# Lezione 9

# Sommario della lezione:

# Sommario della lezione:

- Applicazioni della tecnica Divide et Impera

# Sommario della lezione:

- Applicazioni della tecnica Divide et Impera
- Esempio di analisi di complessità nel caso medio

- Dato un array `a=a[0]...a[n-1]`, definiamo il **rango** di un generico elemento `x` in `a` come il numero di elementi che sono $\leq$ di `x` nel vettore `a`.

- Dato un array `a=a[0]...a[n-1]`, definiamo il **rango** di un generico elemento `x` in `a` come il numero di elementi che sono $\leq$ di `x` nel vettore `a`.
- Ad esempio, l'elemento di rango 1 sarà il minimo di `a`,

- Dato un array `a=a[0]...a[n-1]`, definiamo il **rango** di un generico elemento `x` in `a` come il numero di elementi che sono $\leq$ di `x` nel vettore `a`.
- Ad esempio, l'elemento di rango 1 sarà il minimo di `a`, l'elemento di rango 2 sarà il secondo minimo di `a`,

- Dato un array `a=a[0]...a[n-1]`, definiamo il **rango** di un generico elemento `x` in `a` come il numero di elementi che sono $\leq$ di `x` nel vettore `a`.
- Ad esempio, l'elemento di rango 1 sarà il minimo di `a`, l'elemento di rango 2 sarà il secondo minimo di `a`, ... , l'elemento di rango $n$ sarà l'elemento di valore massimo in `a`.

- Dato un array `a=a[0]...a[n-1]`, definiamo il **rango** di un generico elemento `x` in `a` come il numero di elementi che sono $\leq$ di `x` nel vettore `a`.
- Ad esempio, l'elemento di rango 1 sarà il minimo di `a`, l'elemento di rango 2 sarà il secondo minimo di `a`, ... , l'elemento di rango $n$ sarà l'elemento di valore massimo in `a`.

Consideriamo il seguente problema, che chiameremo `Selezione`:

**Input**: array `a=a[0]...a[n-1]` di interi *differenti* tra di loro,

- Dato un array `a=a[0]...a[n-1]`, definiamo il **rango** di un generico elemento `x` in `a` come il numero di elementi che sono $\leq$ di `x` nel vettore `a`.
- Ad esempio, l'elemento di rango 1 sarà il minimo di `a`, l'elemento di rango 2 sarà il secondo minimo di `a`, ... , l'elemento di rango $n$ sarà l'elemento di valore massimo in `a`.

Consideriamo il seguente problema, che chiameremo `Selezione`:

**Input**: array `a=a[0]...a[n-1]` di interi *differenti* tra di loro, intero $k \in \{1, 2, \ldots, n\}$

- Dato un array `a=a[0]...a[n-1]`, definiamo il **rango** di un generico elemento `x` in `a` come il numero di elementi che sono $\leq$ di `x` nel vettore `a`.
- Ad esempio, l'elemento di rango 1 sarà il minimo di `a`, l'elemento di rango 2 sarà il secondo minimo di `a`, ... , l'elemento di rango $n$ sarà l'elemento di valore massimo in `a`.

Consideriamo il seguente problema, che chiameremo `Selezione`:

**Input**: array `a=a[0]...a[n-1]` di interi *differenti* tra di loro, intero $k \in \{1, 2, \ldots, n\}$
**Output**: l'elemento di rango $k$ in `a`.

# Un semplice algoritmo:

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

Possiamo far meglio?

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

Possiamo far meglio?

- Nel caso in cui $k$ è una costante $c$ ($c$ =1, 2, ...) basterà cercare il minimo se $c = 1$,

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

Possiamo far meglio?

- Nel caso in cui $k$ è una costante $c$ ($c$ =1, 2, ...) basterà cercare il minimo se $c = 1$, il secondo minimo se $c = 2$, etc...)

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

Possiamo far meglio?

- Nel caso in cui $k$ è una costante $c$ ($c$ =1, 2, ...) basterà cercare il minimo se $c = 1$, il secondo minimo se $c = 2$, etc...) e quindi risolvere il problema in tempo $\Theta(n)$.

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

Possiamo far meglio?

- Nel caso in cui $k$ è una costante $c$ ($c$ =1, 2, ...) basterà cercare il minimo se $c = 1$, il secondo minimo se $c = 2$, etc...) e quindi risolvere il problema in tempo $\Theta(n)$.
- Analogamente se $k = n - c$ per qualche costante $c = 0, 1, \ldots$ basterà cercare il massimo se $c = 1$, etc.,

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

Possiamo far meglio?

- Nel caso in cui $k$ è una costante $c$ ($c$ =1, 2, ...) basterà cercare il minimo se $c = 1$, il secondo minimo se $c = 2$, etc...) e quindi risolvere il problema in tempo $\Theta(n)$.
- Analogamente se $k = n - c$ per qualche costante $c = 0, 1, ...$ basterà cercare il massimo se $c = 1$, etc., e di nuovo il problema lo si potrà risolvere $\Theta(n)$.

# Un semplice algoritmo:

1. **Ordina** `a=a[0]...a[n-1]`
2. **Restituisci** `a[k-1]`.

Complessità: $O(n \log n)$, usando Mergesort (ad esempio) per implementare il passo 1.

Possiamo far meglio?

- Nel caso in cui $k$ è una costante $c$ ($c$ =1, 2, ...) basterà cercare il minimo se $c = 1$, il secondo minimo se $c = 2$, etc...) e quindi risolvere il problema in tempo $\Theta(n)$.
- Analogamente se $k = n - c$ per qualche costante $c = 0, 1, ...$ basterà cercare il massimo se $c = 1$, etc., e di nuovo il problema lo si potrà risolvere $\Theta(n)$.

Ed in generale?

# Questione preliminare:

Come stabilire che un dato elemento `x` di `a=a[0]...a[n-1]` è l'elemento di rango $k$ di `a=a[0]...a[n-1]`?

# Questione preliminare:

Come stabilire che un dato elemento `x` di `a=a[0]...a[n-1]` è l'elemento di rango $k$ di `a=a[0]...a[n-1]`?

Potremmo dividere (in qualche modo) l'array `a=a[0]...a[n-1]` in due sottoarray $A_1$ ed $A_2$,

# Questione preliminare:

Come stabilire che un dato elemento `x` di `a=a[0]...a[n-1]` è l'elemento di rango $k$ di `a=a[0]...a[n-1]`?

Potremmo dividere (in qualche modo) l'array `a=a[0]...a[n-1]` in due sottoarray $A_1$ ed $A_2$, dove $A_1$ contiene tutti gli elementi `y` di `a` che sono `<x`

# Questione preliminare:

Come stabilire che un dato elemento `x` di `a=a[0]...a[n-1]` è l'elemento di rango $k$ di `a=a[0]...a[n-1]`?

Potremmo dividere (in qualche modo) l'array `a=a[0]...a[n-1]` in due sottoarray $A_1$ ed $A_2$, dove $A_1$ contiene tutti gli elementi `y` di `a` che sono `<x` e $A_2$ contiene tutti gli elementi `y` di `a` che sono `>x`.

# Questione preliminare:

Come stabilire che un dato elemento `x` di `a=a[0]...a[n-1]` è l'elemento di rango $k$ di `a=a[0]...a[n-1]`?

Potremmo dividere (in qualche modo) l'array `a=a[0]...a[n-1]` in due sottoarray $A_1$ ed $A_2$, dove $A_1$ contiene tutti gli elementi `y` di `a` che sono `<x` e $A_2$ contiene tutti gli elementi `y` di `a` che sono `>x`. Così:

| | | |
|---|---|---|
| $A_1$ =tutti gli elementi `y<x` | `x` | $A_2$ = tutti gli elementi `y>x` |

# Questione preliminare:

Come stabilire che un dato elemento `x` di `a=a[0]...a[n-1]` è l'elemento di rango $k$ di `a=a[0]...a[n-1]`?

Potremmo dividere (in qualche modo) l'array `a=a[0]...a[n-1]` in due sottoarray $A_1$ ed $A_2$, dove $A_1$ contiene tutti gli elementi `y` di `a` che sono `<x` e $A_2$ contiene tutti gli elementi `y` di `a` che sono `>x`. Così:

| $A_1$ =tutti gli elementi y<x | x | $A_2$ = tutti gli elementi y>x |
|---|---|---|

Se il numero di elementi $|A_1|$ in $A_1$ è tale che $|A_1| = k - 1$, l'elemento `x` è **proprio** l'elemento di rango $k$ che cercavamo,

# Questione preliminare:

Come stabilire che un dato elemento `x` di `a=a[0]...a[n-1]` è l'elemento di rango $k$ di `a=a[0]...a[n-1]`?

Potremmo dividere (in qualche modo) l'array `a=a[0]...a[n-1]` in due sottoarray $A_1$ ed $A_2$, dove $A_1$ contiene tutti gli elementi `y` di `a` che sono `<x` e $A_2$ contiene tutti gli elementi `y` di `a` che sono `>x`. Così:

| $A_1$ =tutti gli elementi `y<x` | `x` | $A_2$ = tutti gli elementi `y>x` |
|---|---|---|

Se il numero di elementi $|A_1|$ in $A_1$ è tale che $|A_1| = k - 1$, l'elemento `x` è **proprio** l'elemento di rango $k$ che cercavamo, mentre se $|A_1| \neq k - 1$ l'elemento `x` **non** è l'elemento di rango $k$ che cercavamo.

Il test ci dice *molto di più* del solo fatto che l'elemento `x` è o non è l'elemento di rango $k$ di `a=[0]...a[n-1]`.

Il test ci dice *molto di più* del solo fatto che l'elemento `x` è o non è l'elemento di rango $k$ di `a=[0]...a[n-1]`.

Ricordiamo:

| | | |
|---|---|---|
| $A_1$ =tutti gli elementi `y<x` | `x` | $A_2$ = tutti gli elementi `y>x` |

Il test ci dice *molto di più* del solo fatto che l'elemento `x` è o non è l'elemento di rango $k$ di `a=[0]...a[n-1]`.

Ricordiamo:

| | | |
|---|---|---|
| $A_1$ =tutti gli elementi `y<x` | `x` | $A_2$ = tutti gli elementi `y>x` |

Infatti:

- Se $|A_1| = k - 1$, allora l'elemento di rango $k$ è **proprio** `x`.

Il test ci dice *molto di più* del solo fatto che l'elemento `x` è o non è l'elemento di rango $k$ di `a=[0]...a[n-1]`.

Ricordiamo:

| | | |
|---|---|---|
| $A_1$ =tutti gli elementi `y<x` | `x` | $A_2$ = tutti gli elementi `y>x` |

Infatti:

- Se $|A_1| = k - 1$, allora l'elemento di rango $k$ è **proprio** `x`.
- Se $|A_1| \geq k$, allora l'elemento di rango $k$ **sarà** in $A_1$

Il test ci dice *molto di più* del solo fatto che l'elemento `x` è o non è l'elemento di rango $k$ di `a=[0]...a[n-1]`.

Ricordiamo:

| | | |
|---|---|---|
| $A_1$ =tutti gli elementi `y<x` | `x` | $A_2$ = tutti gli elementi `y>x` |

Infatti:

- Se $|A_1| = k - 1$, allora l'elemento di rango $k$ è **proprio** `x`.
- Se $|A_1| \geq k$, allora l'elemento di rango $k$ **sarà** in $A_1$
- Se $|A_1| < k - 1$, allora l'elemento di rango $k$ **sarà** in $A_2$.

# Possibile algoritmo basato su Divide et Impera:

Partiamo dall'array `a[0]...a[n-1]`

# Possibile algoritmo basato su Divide et Impera:

Partiamo dall'array `a=[0]...a[n-1]`

a =

# Possibile algoritmo basato su Divide et Impera:

Partiamo dall'array `a=[0]...a[n-1]`

a =

e vogliamo trovare l'elemento di rango $k$ in `a=a[0]...a[n-1]`.

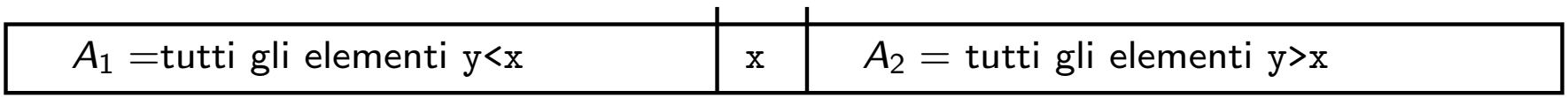
# Possibile algoritmo basato su Divide et Impera:

Partiamo dall'array `a=[0]...a[n-1]`

a =

e vogliamo trovare l'elemento di rango $k$ in `a=a[0]...a[n-1]`.

Scegliamo un elemento `x` in `a=a[0]...a[n-1]` e dividiamo `a=[0]...a[n-1]` in $A_1$ e $A_2$ come prima detto:

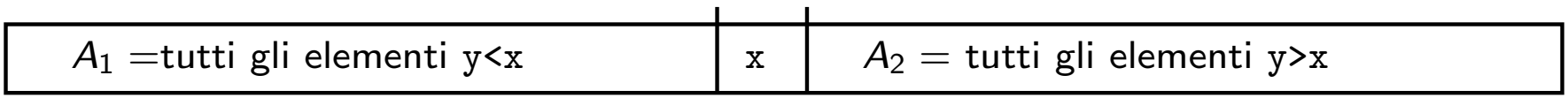


# Possibile algoritmo basato su Divide et Impera:

Partiamo dall'array `a=[0]...a[n-1]`
a =

e vogliamo trovare l'elemento di rango $k$ in `a=a[0]...a[n-1]`.

Scegliamo un elemento `x` in `a=a[0]...a[n-1]` e dividiamo `a=[0]...a[n-1]` in $A_1$ e $A_2$ come prima detto:




- Se $|A_1| = k - 1$, allora l'elemento di rango $k$ é proprio `x` e lo ritorniamo.

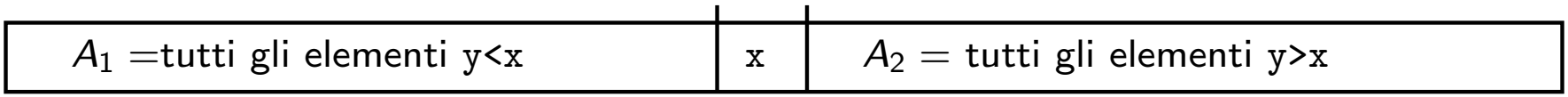
# Possibile algoritmo basato su Divide et Impera:

Partiamo dall'array `a=[0]...a[n-1]`

a =

e vogliamo trovare l'elemento di rango $k$ in `a=a[0]...a[n-1]`.

Scegliamo un elemento `x` in `a=a[0]...a[n-1]` e dividiamo `a=[0]...a[n-1]` in $A_1$ e $A_2$ come prima detto:



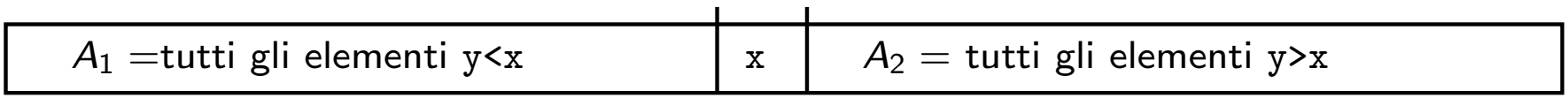

- Se $|A_1| = k - 1$, allora l'elemento di rango $k$ é proprio `x` e lo ritorniamo.
- Se $|A_1| \geq k$, allora l'elemento di rango $k \in A_1$ e quindi ricorriamo in $A_1$.

# Possibile algoritmo basato su Divide et Impera:

Partiamo dall'array `a=[0]...a[n-1]`
a =

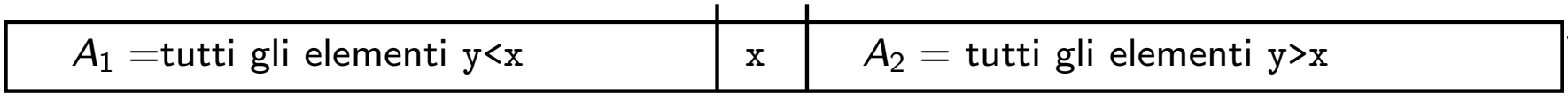
e vogliamo trovare l'elemento di rango $k$ in `a=a[0]...a[n-1]`.

Scegliamo un elemento `x` in `a=a[0]...a[n-1]` e dividiamo `a=[0]...a[n-1]` in $A_1$ e $A_2$ come prima detto:

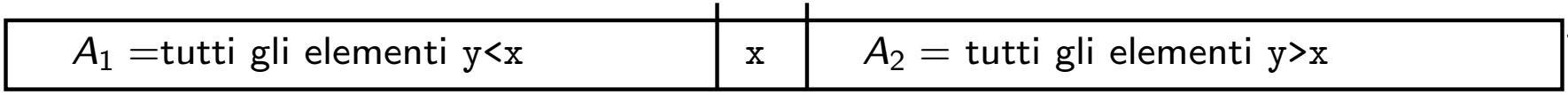



- Se $|A_1| = k-1$, allora l'elemento di rango $k$ é proprio `x` e lo ritorniamo.
- Se $|A_1| \geq k$, allora l'elemento di rango $k \in A_1$ e quindi ricorriamo in $A_1$.
- Se $|A_1| < k-1$, allora l'elemento di rango $k \in A_2$ e quindi ricorriamo in $A_2$.

Come suddividere `a=[0]...a[n-1]` in

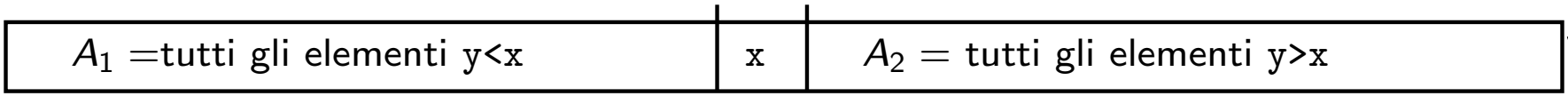


Come suddividere a=[0]...a[n-1] in




Potremmo scorrere a da sinistra a destra e confrontando uno ad uno tutti gli elementi di a con x.

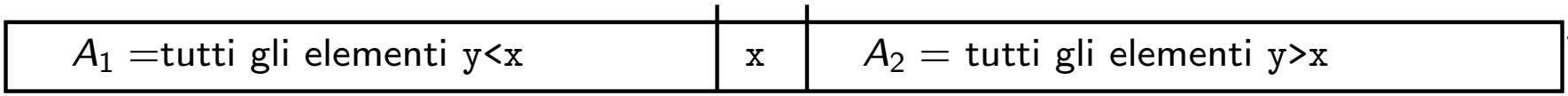
Come suddividere a=[0]...a[n-1] in




Potremmo scorrere a da sinistra a destra e confrontando uno ad uno tutti gli elementi di a con x.

- Se l'elemento confrontato è <x allora lo si mette in $A_1$,

Come suddividere `a=[0]...a[n-1]` in




Potremmo scorrere a da sinistra a destra e confrontando uno ad uno tutti gli elementi di a con x.

- Se l'elemento confrontato è <x allora lo si mette in $A_1$,
- Se l'elemento confrontato è >x allora lo si mette in $A_2$.

# Algoritmo `Distribuzione(a, sx, px, dx)`

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame

`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE (i<= j) {
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE (i<= j) {
4.    WHILE ((i<= j) && (a[i]<= a[dx])) {  i=i+1  }
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE (i<= j) {
4.     WHILE ((i<= j) && (a[i]<= a[dx])) {  i=i+1  }
5.     WHILE ((i<= j) && (a[j]>= a[dx])) {   j=j-1  }
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE (i<= j) {
4.     WHILE ((i<= j) && (a[i]<= a[dx])) {  i=i+1  }
5.     WHILE ((i<= j) && (a[j]>= a[dx])) {   j=j-1  }
6. IF (i<j) Scambia (i, j)   }
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE (i<= j) {
4.    WHILE ((i<= j) && (a[i]<= a[dx])) {  i=i+1  }
5.    WHILE ((i<= j) && (a[j]>= a[dx])) {   j=j-1  }
6. IF (i<j) Scambia (i, j)   }
7. IF (i!=dx) Scambia (i,dx)
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE (i<= j) {
4.    WHILE ((i<= j) && (a[i]<= a[dx])) {  i=i+1  }
5.    WHILE ((i<= j) && (a[j]>= a[dx])) {   j=j-1  }
6. IF (i<j) Scambia (i, j)   }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

# Algoritmo `Distribuzione(a, sx, px, dx)`

`sx`=indice dell'estremità sinistra della parte di a correntemente sotto esame
`dx`=indice dell'estremità destra della parte di a correntemente sotto esame

`px`=indice dell'elemento `x` di a che useremo per suddividere a in $A_1$ ed $A_2$ (cioè `x=a[px]`), che chiameremo **pivot**

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE (i<= j) {
4.     WHILE ((i<= j) && (a[i]<= a[dx])) {  i=i+1  }
5.     WHILE ((i<= j) && (a[j]>= a[dx])) {   j=j-1  }
6. IF (i<j) Scambia (i, j)   }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

```
Scambia (i, j)
temp=a[j]
a[j]=a[i]
a[i]=temp
```

# Eseguiamo `Distribuzione(a,0,1,7)`

su `a=a[0]...a[7]= 6 4 2 10 9 3 5 8`.

# Eseguiamo `Distribuzione(a,0,1,7)`

su `a=a[0]...a[7]= 6 4 2 10 9 3 5 8`. Quindi, `sx=0, px=1, dx=7`.

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8,

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4.

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5.

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5. Poichè a[6]=5 > 4, ciò ci va bene (infatti vogliamo che a destra compaiano elementi più grandi di 4)

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5. Poichè a[6]=5 > 4, ciò ci va bene (infatti vogliamo che a destra compaiano elementi più grandi di 4) e continuiamo, decrementando il valore di j a 5.

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5. Poichè a[6]=5 > 4, ciò ci va bene (infatti vogliamo che a destra compaiano elementi più grandi di 4) e continuiamo, decrementando il valore di j a 5. Confrontiamo a[5]=3 con a[7]=4 e ci fermiamo, in quanto 3<4.

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5. Poichè a[6]=5 > 4, ciò ci va bene (infatti vogliamo che a destra compaiano elementi più grandi di 4) e continuiamo, decrementando il valore di j a 5. Confrontiamo a[5]=3 con a[7]=4 e ci fermiamo, in quanto 3<4.
- Avendo eseguito il WHILE dei passi 4. e 5,

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5. Poichè a[6]=5 > 4, ciò ci va bene (infatti vogliamo che a destra compaiano elementi più grandi di 4) e continuiamo, decrementando il valore di j a 5. Confrontiamo a[5]=3 con a[7]=4 e ci fermiamo, in quanto 3<4.
- Avendo eseguito il WHILE dei passi 4. e 5, verifichiamo che i=0<5

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5. Poichè a[6]=5 > 4, ciò ci va bene (infatti vogliamo che a destra compaiano elementi più grandi di 4) e continuiamo, decrementando il valore di j a 5. Confrontiamo a[5]=3 con a[7]=4 e ci fermiamo, in quanto 3<4.
- Avendo eseguito il WHILE dei passi 4. e 5, verifichiamo che i=0<5 e scambiamo gli elementi a[0]=6 con a[5]=3 tra di loro,

# Eseguiamo Distribuzione(a,0,1,7)

su a=a[0]...a[7]= 6 4 2 10 9 3 5 8. Quindi, sx=0, px=1, dx=7.

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
           }
6. IF (i<j) Scambia (i, j)
       }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- px$\neq$7 $\Rightarrow$ scambiamo tra di loro le posizioni di a[1]=4 con a[7]=8, per ottenere 6 8 2 10 9 3 5 4
- Poniamo i=0 e j=6 ed iniziamo ad eseguire il WHILE al passo 4. Poichè a[0]=6>a[7]=4 ci fermiamo immediatamente senza incrementare il valore di i
- eseguiamo il WHILE al punto 5. Poichè a[6]=5 > 4, ciò ci va bene (infatti vogliamo che a destra compaiano elementi più grandi di 4) e continuiamo, decrementando il valore di j a 5. Confrontiamo a[5]=3 con a[7]=4 e ci fermiamo, in quanto 3<4.
- Avendo eseguito il WHILE dei passi 4. e 5, verifichiamo che i=0<5 e scambiamo gli elementi a[0]=6 con a[5]=3 tra di loro, per ottenere la sequenza
3 8 2 10 9 6 5 4.

3 8 2 10 9 6 5 4 i=0 j=5

3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.    WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
       }
6. IF (i<j) Scambia (i, j)
           }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
          i=i+1
          }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
          j=j-1
       }
6. IF (i<j) Scambia (i, j)
          }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
          i=i+1
          }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
          j=j-1
       }
6. IF (i<j) Scambia (i, j)
          }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
          i=i+1
          }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
          j=j-1
       }
6. IF (i<j) Scambia (i, j)
          }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2,

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
         i=i+1
         }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
         j=j-1
       }
6. IF (i<j) Scambia (i, j)
         }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
         i=i+1
         }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
         j=j-1
       }
6. IF (i<j) Scambia (i, j)
         }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`.

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
          i=i+1
          }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
          j=j-1
       }
6. IF (i<j) Scambia (i, j)
          }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`)

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
        }
6. IF (i<j) Scambia (i, j)
           }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`) e `j=1` e ci fermiamo

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
         }
6. IF (i<j) Scambia (i, j)
           }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`) e `j=1` e ci fermiamo (in quanto `a[1]=2<a[7]=4`),

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
          i=i+1
          }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
          j=j-1
       }
6. IF (i<j) Scambia (i, j)
          }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`) e `j=1` e ci fermiamo (in quanto `a[1]=2<a[7]=4`), NON eseguiremo lo scambio in quanto adesso `i=2>1=j`.

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
          i=i+1
          }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
          j=j-1
       }
6. IF (i<j) Scambia (i, j)
          }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`) e `j=1` e ci fermiamo (in quanto `a[1]=2<a[7]=4`), NON eseguiremo lo scambio in quanto adesso `i=2>1=j`. Termineremo il `WHILE` al passo 5.

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
          i=i+1
          }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
          j=j-1
       }
6. IF (i<j) Scambia (i, j)
          }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`) e `j=1` e ci fermiamo (in quanto `a[1]=2<a[7]=4`), NON eseguiremo lo scambio in quanto adesso `i=2>1=j`. Termineremo il `WHILE` al passo 5.
- Poichè $i \neq 7$, scambiamo `a[2]=8` con `a[7]=4`, otteniamo la sequenza

  3 2 4 10 9 6 5 8

e restituiamo `i=2`.

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.   WHILE ((i<=j)&&
          (a[i]<=a[dx])){
         i=i+1
         }
5.    WHILE ((i<=j) &&
          (a[j]>= a[dx])){
         j=j-1
       }
6. IF (i<j) Scambia (i, j)
         }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`) e `j=1` e ci fermiamo (in quanto `a[1]=2<a[7]=4`), NON eseguiremo lo scambio in quanto adesso `i=2>1=j`. Termineremo il `WHILE` al passo 5.
- Poichè $i \neq 7$, scambiamo `a[2]=8` con `a[7]=4`, otteniamo la sequenza

  3 2 4 10 9 6 5 8

e restituiamo `i=2`.

Osserviamo che tutti i valori a "sinistra" di `a[2]` sono piùpiccoli di `a[2]` e tutti i valori a "destra" di `a[2]` sono più grandi di `a[2]`.

# 3 8 2 10 9 6 5 4 i=0 j=5

```
Distribuzione(a, sx, px, dx)
1. IF (px!= dx) Scambia (px, dx)
2. i=sx, j=dx-1
3. WHILE(i<= j){
4.    WHILE ((i<=j)&&
            (a[i]<=a[dx])){
           i=i+1
           }
5.     WHILE ((i<=j) &&
            (a[j]>= a[dx])){
           j=j-1
         }
6. IF (i<j) Scambia (i, j)
           }
7. IF (i!=dx) Scambia (i,dx)
8. RETURN i
```

- Effettuiamo un'altra esecuzione del `WHILE` al passo 5. e aumentiamo `i` da 0 a 1 e ci fermiamo in quanto `a[1]=8> 4 = a[7]`.
- Eseguiremo il `WHILE` al passo 5. fin quando `j` si decrementa fino a 2, in quanto **solo** in questo caso `a[j]<a[7]=4`.
- ora `i=1` e `j=2`. Poichè `i<j`, effettuiamo lo scambio e la nuova sequenza sarà

  3 2 8 10 9 6 5 4.

- Poniamo `i=2` (in quanto `a[2]=8 > a[7]=4`) e `j=1` e ci fermiamo (in quanto `a[1]=2<a[7]=4`), NON eseguiremo lo scambio in quanto adesso `i=2>1=j`. Termineremo il `WHILE` al passo 5.
- Poichè $i \neq 7$, scambiamo `a[2]=8` con `a[7]=4`, otteniamo la sequenza

  3 2 4 10 9 6 5 8

  e restituiamo `i=2`.

Osserviamo che tutti i valori a "sinistra" di `a[2]` sono piùpiccoli di `a[2]` e tutti i valori a "destra" di `a[2]` sono più grandi di `a[2]`.

`Distribuzione` richiede tempo $\Theta(n)$ su di una sequenza composta di $n$ elementi

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra)
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      }
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
6.         rango=Distribuzione (a, sinistra, pivot, destra)
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
6.         rango=Distribuzione (a, sinistra, pivot, destra)
7.         IF (k-1 == rango)
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
6.         rango=Distribuzione (a, sinistra, pivot, destra)
7.         IF (k-1 == rango) {
8.          RETURN a[rango]
9.         }
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
6.         rango=Distribuzione (a, sinistra, pivot, destra)
7.         IF (k-1 == rango) {
8.          RETURN a[rango]
9.         } ELSE IF (k-1< rango)
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
6.         rango=Distribuzione (a, sinistra, pivot, destra)
7.         IF (k-1 == rango) {
8.          RETURN a[rango]
9.         } ELSE IF (k-1< rango) {
10.          RETURN QuickSelect (a,sinistra,k,rango-1)
11.         }
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
6.         rango=Distribuzione (a, sinistra, pivot, destra)
7.         IF (k-1 == rango) {
8.          RETURN a[rango]
9.         } ELSE IF (k-1< rango) {
10.          RETURN QuickSelect (a,sinistra,k,rango-1)
11.         } ELSE {
12.          RETURN QuickSelect (a,rango+1,k,destra)
```

# L'algoritmo

**Osservazione**: `Distribuzione` trova **il rango del pivot scelto**, posizionando tutti gli elementi inferiori al pivot alla sua sinistra e tutti gli elementi più grandi del pivot alla sua destra.

L'algoritmo per calcolare l'elemento di rango $k$ in `a=a[0]...a[n-1]` è

```
1.  QuickSelect (a,sinistra,k,destra)
2.  IF (sinistra == destra) {
3.      RETURN a[sinistra]
4.      } ELSE {
5.         scegli pivot nell'intervallo [sinistra...  destra]
6.         rango=Distribuzione (a, sinistra, pivot, destra)
7.         IF (k-1 == rango) {
8.          RETURN a[rango]
9.         } ELSE IF (k-1< rango) {
10.          RETURN QuickSelect (a,sinistra,k,rango-1)
11.         } ELSE {
12.          RETURN QuickSelect (a,rango+1,k,destra)
      }
}
```

# Analisi di `QuickSelect(a,0,k,n-1)`

Assumiamo che l'elemento che cerchiamo si trovi sempre nella parte di array a "più grande",

# Analisi di `QuickSelect(a,0,k,n-1)`

Assumiamo che l'elemento che cerchiamo si trovi sempre nella parte di array a "più grande", per cui avremmo un'equazione di ricorrenza del tipo

$$T(n) \leq \begin{cases} c & \text{se } n \leq 1 \\ T(\max(r-1, n-r)) + dn & \text{altrimenti} \end{cases} \quad (1)$$

dove $r =$`rango +1` (ovvero la posizione occupata dall'elemento `a[pivot]` dopo `Distribuzione`).

# Analisi di `QuickSelect(a,0,k,n-1)`

Assumiamo che l'elemento che cerchiamo si trovi sempre nella parte di array a "più grande", per cui avremmo un'equazione di ricorrenza del tipo

$$T(n) \leq \begin{cases} c & \text{se } n \leq 1 \\ T(\max(r-1, n-r)) + dn & \text{altrimenti} \end{cases} \quad (1)$$

dove $r =$`rango +1` (ovvero la posizione occupata dall'elemento `a[pivot]` dopo `Distribuzione`). La (1) ha diverse soluzione, a seconda del valore di $r$.

# Analisi di `QuickSelect(a,0,k,n-1)`

Assumiamo che l'elemento che cerchiamo si trovi sempre nella parte di array a "più grande", per cui avremmo un'equazione di ricorrenza del tipo

$$T(n) \leq \begin{cases} c & \text{se } n \leq 1 \\ T(\max(r-1, n-r)) + dn & \text{altrimenti} \end{cases} \quad (1)$$

dove $r =$`rango +1` (ovvero la posizione occupata dall'elemento `a[pivot]` dopo `Distribuzione`). La (1) ha diverse soluzione, a seconda del valore di $r$.

• Se la ricorsione fosse **sempre** del tipo $T(n) \leq T(n-1) + dn$ (e ciò accade se ad **ogni passo** scegliamo un pivot di rango 1) avremmo una soluzione $T(n) = O(n^2)$,

# Analisi di `QuickSelect(a,0,k,n-1)`

Assumiamo che l'elemento che cerchiamo si trovi sempre nella parte di array a "più grande", per cui avremmo un'equazione di ricorrenza del tipo

$$T(n) \leq \begin{cases} c & \text{se } n \leq 1 \\ T(\max(r-1, n-r)) + dn & \text{altrimenti} \end{cases} \quad (1)$$

dove $r =$`rango +1` (ovvero la posizione occupata dall'elemento `a[pivot]` dopo `Distribuzione`). La (1) ha diverse soluzione, a seconda del valore di $r$.

- Se la ricorsione fosse **sempre** del tipo $T(n) \leq T(n-1) + dn$ (e ciò accade se ad **ogni passo** scegliamo un pivot di rango 1) avremmo una soluzione $T(n) = O(n^2)$,
- se **fosse** sempre del tipo $T(n) \leq T(n/2) + dn$ (e ciò accade se ad **ogni passo** scegliamo un pivot di rango $n/2$)

# Analisi di `QuickSelect(a,0,k,n-1)`

Assumiamo che l'elemento che cerchiamo si trovi sempre nella parte di array a "più grande", per cui avremmo un'equazione di ricorrenza del tipo

$$T(n) \leq \begin{cases} c & \text{se } n \leq 1 \\ T(\max(r-1, n-r)) + dn & \text{altrimenti} \end{cases} \quad (1)$$

dove $r =$`rango +1` (ovvero la posizione occupata dall'elemento `a[pivot]` dopo `Distribuzione`). La (1) ha diverse soluzione, a seconda del valore di $r$.

- Se la ricorsione fosse **sempre** del tipo $T(n) \leq T(n-1) + dn$ (e ciò accade se ad **ogni passo** scegliamo un pivot di rango 1) avremmo una soluzione $T(n) = O(n^2)$,
- se **fosse** sempre del tipo $T(n) \leq T(n/2) + dn$ (e ciò accade se ad **ogni passo** scegliamo un pivot di rango $n/2$) avremmo una soluzione $T(n) = O(n)$.

# Analisi di `QuickSelect(a,0,k,n-1)`

Assumiamo che l'elemento che cerchiamo si trovi sempre nella parte di array a "più grande", per cui avremmo un'equazione di ricorrenza del tipo

$$T(n) \leq \begin{cases} c & \text{se } n \leq 1 \\ T(\max(r-1, n-r)) + dn & \text{altrimenti} \end{cases} \quad (1)$$

dove $r =$`rango +1` (ovvero la posizione occupata dall'elemento `a[pivot]` dopo `Distribuzione`). La (1) ha diverse soluzione, a seconda del valore di $r$.

- Se la ricorsione fosse **sempre** del tipo $T(n) \leq T(n-1) + dn$ (e ciò accade se ad **ogni passo** scegliamo un pivot di rango 1) avremmo una soluzione $T(n) = O(n^2)$,
- se **fosse** sempre del tipo $T(n) \leq T(n/2) + dn$ (e ciò accade se ad **ogni passo** scegliamo un pivot di rango $n/2$) avremmo una soluzione $T(n) = O(n)$. Ma trovare in maniera efficiente un pivot di un *dato* rango è proprio il problema che vogliamo risolvere, e quindi non lo possiamo dare per già risolto!

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

- Per la equazione che governa la complessità $T(n)$ di `QuickSelect(a,0,k,n-1)` varrà la relazione $T(n) \leq T(\max(r-1, n-r)) + dn$

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

- Per la equazione che governa la complessità $T(n)$ di `QuickSelect(a,0,k,n-1)` varrà la relazione $T(n) \leq T(\max(r-1, n-r)) + dn$ **se e solo se** la nostra scelta a caso del pivot ci ha restituito un pivot di rango $r$,

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

- Per la equazione che governa la complessità $T(n)$ di `QuickSelect(a,0,k,n-1)` varrà la relazione $T(n) \leq T(\max(r-1, n-r)) + dn$ **se e solo se** la nostra scelta a caso del pivot ci ha restituito un pivot di rango $r$, e ciò avverrà con probabilità $1/n$.

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

• Per la equazione che governa la complessità $T(n)$ di `QuickSelect(a,0,k,n-1)` varrà la relazione $T(n) \leq T(\max(r-1, n-r)) + dn$ **se e solo se** la nostra scelta a caso del pivot ci ha restituito un pivot di rango $r$, e ciò avverrà con probabilità $1/n$.

• Infatti, se calcoliamo la probabilità come =numero di casi favorevoli)/(numero di casi possibili),

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

• Per la equazione che governa la complessità $T(n)$ di `QuickSelect(a,0,k,n-1)` varrà la relazione $T(n) \leq T(\max(r-1, n-r)) + dn$ **se e solo se** la nostra scelta a caso del pivot ci ha restituito un pivot di rango $r$, e ciò avverrà con probabilità $1/n$.

• Infatti, se calcoliamo la probabilità come =numero di casi favorevoli)/(numero di casi possibili), poichè esiste un **unico** elemento di rango $r$, per ogni dato $r$, otteniamo che la probabilità è appunto pari a $1/n$.

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

• Per la equazione che governa la complessità $T(n)$ di `QuickSelect(a,0,k,n-1)` varrà la relazione $T(n) \leq T(\max(r-1, n-r)) + dn$ **se e solo se** la nostra scelta a caso del pivot ci ha restituito un pivot di rango $r$, e ciò avverrà con probabilità $1/n$.

• Infatti, se calcoliamo la probabilità come =numero di casi favorevoli)/(numero di casi possibili), poichè esiste un **unico** elemento di rango $r$, per ogni dato $r$, otteniamo che la probabilità è appunto pari a $1/n$.

• Ci troviamo quindi di fronte ad una quantità (il tempo di esecuzione di `QuickSelect`) che assume *differenti* valori con certe probabilità .

# La nuova idea!

Che succede se scegliessimo il pivot **a caso**?

• Per la equazione che governa la complessità $T(n)$ di `QuickSelect(a,0,k,n-1)` varrà la relazione $T(n) \leq T(\max(r-1, n-r)) + dn$ **se e solo se** la nostra scelta a caso del pivot ci ha restituito un pivot di rango $r$, e ciò avverrà con probabilità $1/n$.

• Infatti, se calcoliamo la probabilità come =numero di casi favorevoli)/(numero di casi possibili), poichè esiste un **unico** elemento di rango $r$, per ogni dato $r$, otteniamo che la probabilità è appunto pari a $1/n$.

• Ci troviamo quindi di fronte ad una quantità (il tempo di esecuzione di `QuickSelect`) che assume *differenti* valori con certe probabilità . Quindi, non ha più senso parlare di tempo di esecuzione nel caso peggiore, ma occorrerà valutare il tempo di esecuzione nel caso medio.

# E valutiamola, allora....

Ricordiamo che se abbiamo una generica "quantità " $T$ che può assumere differenti valori $t_1, t_2, \ldots, t_n$,

# E valutiamola, allora....

Ricordiamo che se abbiamo una generica "quantità " $T$ che può assumere differenti valori $t_1, t_2, \ldots, t_n$, con rispettive probabilità $\Pr\{T = t_i\} = p_i$, per $i = 1, \ldots, n$,

# E valutiamola, allora....

Ricordiamo che se abbiamo una generica "quantità" $T$ che può assumere differenti valori $t_1, t_2, \ldots, t_n$, con rispettive probabilità $\Pr\{T = t_i\} = p_i$, per $i = 1, \ldots, n$, allora il valore medio di $T$ è

$$E[T] = \sum_{i=1}^{n} t_i \times p_i.$$

# E valutiamola, allora....

Ricordiamo che se abbiamo una generica "quantità" $T$ che può assumere differenti valori $t_1, t_2, \ldots, t_n$, con rispettive probabilità $\Pr\{T = t_i\} = p_i$, per $i = 1, \ldots, n$, allora il valore medio di $T$ è

$$E[T] = \sum_{i=1}^{n} t_i \times p_i.$$

Occorre quindi valutare la seguente quantità per `QuickSelect(a,0,k,n-1)`:

$$T(n) = \sum_{\text{su tutti i tempi di esecuzione}} (\text{tempo di esecuzione}) \times \Pr\{\text{di avere quel tempo di esec.}\}$$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

il tempo medio di `QuickSelect(a,0,k,n-1)` sarà pari a:

$$T(n) \leq (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 1\}$$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

il tempo medio di `QuickSelect(a,0,k,n-1)` sarà pari a:

$$\begin{aligned} T(n) \leq & (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 1\} \\ & + (T(\max(1, n-2)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 2\} \end{aligned}$$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

il tempo medio di `QuickSelect(a,0,k,n-1)` sarà pari a:

$$\begin{aligned} T(n) \leq & (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 1\} \\ & + (T(\max(1, n-2)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 2\} \\ & + (T(\max(2, n-3)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 3\} \end{aligned}$$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

il tempo medio di `QuickSelect(a,0,k,n-1)` sarà pari a:

$$\begin{aligned} T(n) \leq & (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 1\} \\ & + (T(\max(1, n-2)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 2\} \\ & + (T(\max(2, n-3)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 3\} \\ & \vdots \\ & + (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } n\} \end{aligned}$$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

il tempo medio di `QuickSelect(a,0,k,n-1)` sarà pari a:

$$\begin{aligned}
T(n) \leq & (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 1\} \\
& + (T(\max(1, n-2)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 2\} \\
& + (T(\max(2, n-3)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 3\} \\
& \vdots \\
& + (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } n\} \\
= & \frac{1}{n} \sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n).
\end{aligned}$$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

il tempo medio di `QuickSelect(a,0,k,n-1)` sarà pari a:

$$\begin{aligned}
T(n) \leq & (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 1\} \\
& + (T(\max(1, n-2)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 2\} \\
& + (T(\max(2, n-3)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } 3\} \\
& \vdots \\
& + (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento } \texttt{a[pivot]} \text{ di rango } n\} \\
= & \frac{1}{n} \sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n).
\end{aligned}$$

dove abbiamo assunto che $T(0) = 0$

Ricordando

$$\forall r \quad T(n) \leq T(\max(r-1, n-r)) + \Theta(n) \text{ con probabilità } \frac{1}{n}$$

il tempo medio di `QuickSelect(a,0,k,n-1)` sarà pari a:

$$\begin{aligned}
T(n) \leq & (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento a[pivot] di rango } 1\} \\
& + (T(\max(1, n-2)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento a[pivot] di rango } 2\} \\
& + (T(\max(2, n-3)) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento a[pivot] di rango } 3\} \\
& \vdots \\
& + (T(n-1) + \Theta(n)) \cdot \Pr\{\text{scegliere un elemento a[pivot] di rango } n\} \\
= & \frac{1}{n} \sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n).
\end{aligned}$$

dove abbiamo assunto che $T(0) = 0$ ed abbiamo sfruttato il fatto che

$$\Pr\{\text{scegliere un elemento a[pivot] di rango } k\} = 1/n$$

per ogni possibile valore di $k$.

Partendo da

$$\frac{1}{n}\sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n)$$

Partendo da

$$\frac{1}{n}\sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n)$$

osserviamo che

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil \\ \end{cases}$$

Partendo da

$$\frac{1}{n}\sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n)$$

osserviamo che

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil \\ n-k & \text{se } k \leq \lceil n/2 \rceil. \end{cases}$$

Partendo da

$$\frac{1}{n}\sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n)$$

osserviamo che

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil \\ n-k & \text{se } k \leq \lceil n/2 \rceil. \end{cases}$$

• Se $n$ è pari, nella espressione di $T(n)$ di sopra ogni termine da $T(\lceil n/2 \rceil)$ fino a $T(n-1)$ compare **due** volte

Partendo da

$$\frac{1}{n}\sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n)$$

osserviamo che

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil \\ n-k & \text{se } k \leq \lceil n/2 \rceil. \end{cases}$$

- Se $n$ è pari, nella espressione di $T(n)$ di sopra ogni termine da $T(\lceil n/2 \rceil)$ fino a $T(n-1)$ compare **due** volte
- se $n$ è dispari tutti questi termini appaiono due volte mentre il termine $T(\lfloor n/2 \rfloor)$ appare una sola volta.

Partendo da

$$\frac{1}{n}\sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n)$$

osserviamo che

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil \\ n-k & \text{se } k \leq \lceil n/2 \rceil. \end{cases}$$

- Se $n$ è pari, nella espressione di $T(n)$ di sopra ogni termine da $T(\lceil n/2 \rceil)$ fino a $T(n-1)$ compare **due** volte
- se $n$ è dispari tutti questi termini appaiono due volte mentre il termine $T(\lfloor n/2 \rfloor)$ appare una sola volta.

Pertanto, possiamo dire che

$$T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n). \tag{2}$$

Partendo da

$$\frac{1}{n}\sum_{k=1}^{n} T(\max(k-1, n-k)) + \Theta(n)$$

osserviamo che

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil \\ n-k & \text{se } k \leq \lceil n/2 \rceil. \end{cases}$$

- Se $n$ è pari, nella espressione di $T(n)$ di sopra ogni termine da $T(\lceil n/2 \rceil)$ fino a $T(n-1)$ compare **due** volte
- se $n$ è dispari tutti questi termini appaiono due volte mentre il termine $T(\lfloor n/2 \rfloor)$ appare una sola volta.

Pertanto, possiamo dire che

$$T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n). \tag{2}$$

Risolveremo questa ricorrenza per **induzione** su $n$.

Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$.

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an$$

$$= \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$
\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
&= \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an \\
&\leq \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2} \right) + an
\end{aligned}
$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$
\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
&= \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an \\
&\leq \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2} \right) + an \\
&= \frac{2c}{n} \left( \frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an
\end{aligned}
$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$
\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
&= \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an \\
&\leq \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2} \right) + an \\
&= \frac{2c}{n} \left( \frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \\
&= \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an
\end{aligned}
$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$
\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
&= \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an \\
&\leq \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2} \right) + an \\
&= \frac{2c}{n} \left( \frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \\
&= \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an = c \left( \frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an
\end{aligned}
$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$
\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
&= \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an \\
&\leq \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2} \right) + an \\
&= \frac{2c}{n} \left( \frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \\
&= \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an = c \left( \frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \\
&\leq \frac{3cn}{4} + \frac{c}{2} + an
\end{aligned}
$$

# Risolviamo $T(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$.

**Assumiamo** che $T(k) \leq ck$, per qualche costante $c$, e per tutti i valori di $k < n$. Sostituendo, otterremo, per opportuna costante a, che vale:

$$
\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
&= \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an \\
&\leq \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2} \right) + an \\
&= \frac{2c}{n} \left( \frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \\
&= \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an = c \left( \frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \\
&\leq \frac{3cn}{4} + \frac{c}{2} + an = cn - \left( \frac{cn}{4} - \frac{c}{2} - an \right).
\end{aligned}
$$

# E per terminare...

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Per terminare la prova, ovvero che $T(n) \leq cn$,

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Per terminare la prova, ovvero che $T(n) \leq cn$, occorre far vedere che possiamo sempre scegliere $c$ tale che $(cn/4 - c/2 - an) > 0$.

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Per terminare la prova, ovvero che $T(n) \leq cn$, occorre far vedere che possiamo sempre scegliere $c$ tale che $(cn/4 - c/2 - an) > 0$. A tale scopo, basta scegliere $c > 8a$.

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Per terminare la prova, ovvero che $T(n) \leq cn$, occorre far vedere che possiamo sempre scegliere $c$ tale che $(cn/4 - c/2 - an) > 0$. A tale scopo, basta scegliere $c > 8a$.

E chi è $a$?

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Per terminare la prova, ovvero che $T(n) \leq cn$, occorre far vedere che possiamo sempre scegliere $c$ tale che $(cn/4 - c/2 - an) > 0$. A tale scopo, basta scegliere $c > 8a$.

E chi è $a$? $a$ è la costante per cui il tempo di esecuzione di `Distribuzione` è $\leq an$, su input di dimensione $n$.

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Per terminare la prova, ovvero che $T(n) \leq cn$, occorre far vedere che possiamo sempre scegliere $c$ tale che $(cn/4 - c/2 - an) > 0$. A tale scopo, basta scegliere $c > 8a$.

E chi è $a$? $a$ è la costante per cui il tempo di esecuzione di `Distribuzione` è $\leq an$, su input di dimensione $n$.

**Morale della lezione**: effettuando scelte casuali all'interno dell'algoritmo, possiamo trasformare `QuickSelect` (che ha complessità $\Theta(n^2)$ nel caso **peggiore**) in un algoritmo di complessità $\Theta(n)$ nel caso **medio**.

# E per terminare...

Abbiamo quindi provato che

$$T(n) \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Per terminare la prova, ovvero che $T(n) \leq cn$, occorre far vedere che possiamo sempre scegliere $c$ tale che $(cn/4 - c/2 - an) > 0$. A tale scopo, basta scegliere $c > 8a$.

E chi è $a$? $a$ è la costante per cui il tempo di esecuzione di `Distribuzione` è $\leq an$, su input di dimensione $n$.

**Morale della lezione**: effettuando scelte casuali all'interno dell'algoritmo, possiamo trasformare `QuickSelect` (che ha complessità $\Theta(n^2)$ nel caso **peggiore**) in un algoritmo di complessità $\Theta(n)$ nel caso **medio**.

La stessa idea è applicabile ad altri problemi e algoritmi.