Selezione per distribuzione

Problema: selezione dell'elemento con rango r in un array a di n elementi distinti.

- Si vuole evitare di ordinare a
- NB: Il problema diventa quello di trovare il minimo quando $r = 1$ e il massimo quando $r = n$.

Osservazione: la funzione `Distribuzione` permette di trovare il rango del pivot, posizionando tutti gli elementi di rango inferiore alla sua sinistra e tutti quelli di rango superiore alla sua destra.

Possiamo modificare il codice del quicksort procedendo ricorsivamente nel *solo* segmento dell'array contenente l'elemento da selezionare.

La ricorsione ha termine quando il segmento è composto da un solo elemento.

Selezione per distribuzione

```
1  QuickSelect( a, sinistra, r, destra ):
2    IF (sinistra == destra) {
3      RETURN a[sinistra];
4    } ELSE {
5      scegli pivot nell'intervallo [sinistra...destra];
6      indiceFinalePivot = Distribuzione(a, sinistra, pivot, destra);
7      IF (r-1 == indiceFinalePivot) {
8        RETURN a[indiceFinalePivot];
9      } ELSE IF (r-1 < indiceFinalePivot) {
10       RETURN QuickSelect( a, sinistra, r, indiceFinalePivot-1 );
11     } ELSE {
12       RETURN QuickSelect( a, indiceFinalePivot+1, r, destra );
13     }
14   }
```

Analisi di QuickSelect mediante relazione di ricorrenza

- Caso base:
  Se il segmento sul quale opera l'algoritmo contiene un solo elemento allora l'algoritmo esegue un numero costante di operazioni $\leq c_0$;
  Se l'indice restituito da *Distribuzione*(a, *sinistra*, *pivot*, *destra*) è uguale a $r-1$ (in altre parole se il pivot ha rango r), l'algoritmo termina e il suo costo è dominato dal costo di Distribuzione che esegue $\leq c_1 n$ operazioni, dove c_1 è una certa costante.
- Passo ricorsivo: il numero di operazioni eseguite è al più pari a cn (c costante) più il numero di operazioni richieste per effettuare la selezione nel segmento degli elementi minori o uguali del pivot **oppure** in quello degli elementi maggiori o uguali del pivot.

Analisi di QuickSelect mediante relazione di ricorrenza

Relazione di ricorrenza per il tempo $T(n)$ di esecuzione dell'algoritmo.
Indichiamo con r_p il rango del pivot

- Caso base:
  $T(n) \leq c_0$ per $n \leq 1$ e
  $T(n) \leq c_1 n$ se $r_p = r$.
  In entrambi i casi $T(n) \leq c_1 n$.
- Passo ricorsivo: Ci sono $r_p - 1$ elementi a sinistra del pivot e $n - r_p$ elementi a destra, per cui $T(n) \leq \max\{T(r_p - 1), T(n - r_p)\} + cn$.

$$T(n) \leq \begin{cases} c_1 n & \text{se } n \leq 1 \text{ o } r_p = r - 1 \\ \max\{T(r_p - 1), T(n - r_p)\} + cn & \text{altrimenti} \end{cases}$$

Analisi di QuickSelect mediante relazione di ricorrenza

caso pessimo

- Il pivot è tutto a sinistra ($r_p = 1$) e $r > r_p$ oppure tutto a destra ($r_p = n$) e $r < r_p$. In entrambi i casi, la relazione diventa $T(n) \leq T(n-1) + cn$.
- Applichiamo iterativamente la relazione di ricorrenza:

$$T(n) \leq T(n-1)+cn \leq T(n-2)+c(n-1)+cn \leq \ldots \leq T(n-i)+\sum_{j=n-i+1}^{n} c\, i.$$

- Sositutendo $i = n-1$ nell'ultima disequazione, otteniamo

$$T(n) \leq T(1) + \sum_{j=2}^{n} c\, i \leq c_0 + \sum_{j=2}^{n} c\, i = c_0 + c\, n(n+1)/2 - c = O(n^2).$$

Analisi di QuickSelect mediante relazione di ricorrenza

caso ottimo

- L'elemento di rango r è proprio il pivot ($r_p = r$), per cui si esce dalla procedura senza effettuare la ricorsione e si ha che $T(n) = O(n)$.
- Il caso ottimo si verifica anche quando ad ogni chiamata ricorsiva viene dimezzata la lunghezza del segmento in cui effettuare la selezione.

$$T(n) \leq T(n/2) + cn \leq T(n/4) + c(n/2) + cn \leq \ldots \leq T(\frac{n}{2^i}) + \sum_{j=0}^{i-1} c\frac{n}{2^j}.$$

Dopo $\log n$ applicazioni della relazione di ricorrenza otteniamo

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2^{\log n}}\right) + \sum_{j=0}^{\log n - 1} c\frac{n}{2^j} = T(1) + cn \sum_{j=0}^{\log n - 1} \frac{1}{2^j} \\ &\leq c_0 + cn\left(\frac{1 - 1/2^{\log n}}{1/2}\right) = c_0 + 2cn(1 - 1/n) = O(n) \end{aligned}$$

Analisi di QuickSelect mediante relazione di ricorrenza

caso medio

- Tempo $O(n)$ in quanto nelle chiamate ricorsive si alternano situazioni in cui gli elementi si distribuiscono in modo bilanciato a sinistra e a destra del pivot e situazioni in cui gli elementi si distribuiscono in modo non bilanciato. L'algoritmo è veloce in pratica.

Moltiplicazione di interi

- Algoritmo che usiamo comunemente ha tempo di esecuzione $O(n^2)$, dove n e` il numero di cifre di ciascun numero

```
      2345 x
      5382 =

      _____
      4690
    18760
    7035
  11725

_____________
  12620790
```

Moltiplicazione veloce di interi

Ogni numero intero w di n cifre può essere scritto come $10^{n/2} \times w_s + w_d$

- w_s indica il numero formato dalle $n/2$ cifre più significative di w
- w_d denota il numero formato dalle $n/2$ cifre meno significative.

Ad esempio 124100 può essere scritto come $10^3 \times 124 + 100$

Per moltiplicare due numeri x e y, vale l'uguaglianza

$$\begin{aligned} x\,y &= (10^{n/2}\,x_s + x_d)(10^{n/2}\,y_s + y_d) \\ &= 10^n\,x_s\,y_s + 10^{n/2}(x_s\,y_d + x_d\,y_s) + x_d\,y_d \end{aligned}$$

Decomposizione: se x e y hanno almeno due cifre, dividili come numeri x_s, x_d, y_s e y_d aventi ciascuno la metà delle cifre.

Ricorsione: calcola ricorsivamente le moltiplicazioni $x_s\,y_s$, $x_s\,y_d$, $x_d\,y_s$ e $x_d\,y_d$.

Ricombinazione: combina i numeri risultanti usando l'uguaglianza riportata sopra.

Moltiplicazione veloce di interi

- l'algoritmo esegue quattro moltiplicazioni di due numeri di $n/2$ cifre (ad un costo di $T(n/2)$), e tre somme di due numeri di n cifre (a un costo $O(n)$)
- la moltiplicazione per il valore 10^k può essere realizzata spostando le cifre di k posizioni verso sinistra e riempiendo di 0 la parte destra
- il costo della decomposizione e della ricombinazione è $c\,n$

Vale la relazione di ricorrenza

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 4\,T(n/2) + cn & \text{altrimenti} \end{cases}$$

Moltiplicazione veloce di interi

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 4T(n/2) + cn & \text{altrimenti} \end{cases}$$

Assumiamo per semplicità $n = 2^k$ per un certo k e applichiamo iterativamente la relazione di ricorrenza:

$$\begin{aligned} T(n) &\leq cn + 4T(n/2) \leq cn + 4(cn/2 + 4T(n/2^2)) = cn + 2cn + 4^2 T(n/2^2) \\ &\leq cn + 2cn + 4^2(cn/2^2 + 4T(n/2^3)) = cn + 2cn + 2^2 cn + 4^3 T(n/2^3) \\ &\leq \cdots \\ &\leq cn + 2cn + 2^2 cn + \ldots + 2^{i-1} cn + 4^i T(n/2^i) \\ &= cn \sum_{j=0}^{i-1} 2^j + 4^i T(n/2^i) = cn2^i - cn + 4^i T(n/2^i) \end{aligned}$$

Ponendo $i = k = \log_2 n$ si ha $T(n) \leq cn^2 - cn + n^2 T(1) = O(n^2)$.

Moltiplicazione veloce di interi

- È possibile progettare un algoritmo più veloce?
- Osserviamo che sommando e sottraendo $x_s y_s + x_d y_d$ a $x_s y_d + x_d y_s$ si ha

$$\begin{aligned} x_s y_d + x_d y_s &= x_s y_d + x_d y_s + x_s y_s + x_d y_d - x_s y_s - x_d y_d \\ &= x_s y_s + x_d y_d + (x_s y_d + x_d y_s - x_s y_s - x_d y_d) \end{aligned}$$

- Poiché $x_s y_d + x_d y_s - x_s y_s - x_d y_d = -(x_s - x_d) \times (y_s - y_d)$ allora possiamo scrivere

$$x_s y_d + x_d y_s = x_s y_s + x_d y_d - (x_s - x_d) \times (y_s - y_d)$$

- quindi il valore $x_s y_d + x_d y_s$ può essere calcolato facendo uso di $x_s y_s$, $x_d y_d$ e $(x_s - x_d) \times (y_s - y_d)$
- Quindi per computare il prodotto xy sono necessarie tre moltiplicazioni e non più quattro come prima

Moltiplicazione veloce di interi

Si ha quindi la relazione di ricorrenza

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 3T(n/2) + cn & \text{altrimenti} \end{cases}$$

Assumiamo per semplicità $n = 2^k$, per un certo k, e applichiamo iterativamente la relazione di ricorrenza:

$$\begin{aligned}
T(n) &\leq cn + 3T(n/2) \leq cn + 3(cn/2 + 3T(n/2^2)) = cn + (3/2)cn + 3^2 T(n/2^2) \\
&\leq cn + (3/2)cn + 3^2(cn/2^2 + 3T(n/2^3)) = cn + (3/2)cn + (3/2)^2 cn + 3^3 T(n/2^3) \\
&\leq \dots \\
&\leq cn + (3/2)cn + (3/2)^2 cn + \dots + (3/2)^{i-1} cn + 3^i T(n/2^i) \\
&= cn \sum_{j=0}^{i-1} (3/2)^j + 3^i T(n/2^i) = cn \left(\frac{(3/2)^i - 1}{3/2 - 1} \right) + 3^i T(n/2^i) \\
&= 2cn((3/2)^i - 1) + 3^i T(n/2^i) = 2cn(3/2)^i - 2cn + 3^i T(n/2^i)
\end{aligned}$$

Continua nella prossima slide

Ponendo $i = k = \log_2 n$ si ha

$$\begin{aligned}
T(n) &\leq 2cn(3/2)^{\log_2 n} - 2cn + 3^{\log_2 n} T(1) \\
&= 2cn \left(2^{\log_2(3/2)}\right)^{\log_2 n} - 2cn + \left(2^{\log_2 3}\right)^{\log_2 n} T(1) \\
&= 2cn \left(2^{\log_2 n}\right)^{\log_2(3/2)} - 2cn + \left(2^{\log_2 n}\right)^{\log_2 3} T(1) \\
&= 2cn\, n^{\log_2(3/2)} - 2cn + n^{\log_2 3} T(1) \\
&= 2cn\, n^{\log_2 3 - 1} - 2cn + n^{\log_2 3} T(1) \\
&= 2cn^{\log_2 3} - 2cn + n^{\log_2 3} T(1) \\
&\leq 2cn^{\log_2 3} - 2cn + n^{\log_2 3} c_0 \\
&= O(n^{\log_2 3}) = O(n^{1,585})
\end{aligned}$$

Il problema della coppia più vicina

Problema: vogliamo trovare la coppia di punti più vicina tra un insieme di punti del piano.

La distanza tra due punti $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ si calcola con la formula $\sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2}$ in tempo $O(1)$

Il problema può essere risolto in tempo $O(n^2)$ calcolando le distanze tra tutte le coppie di punti.

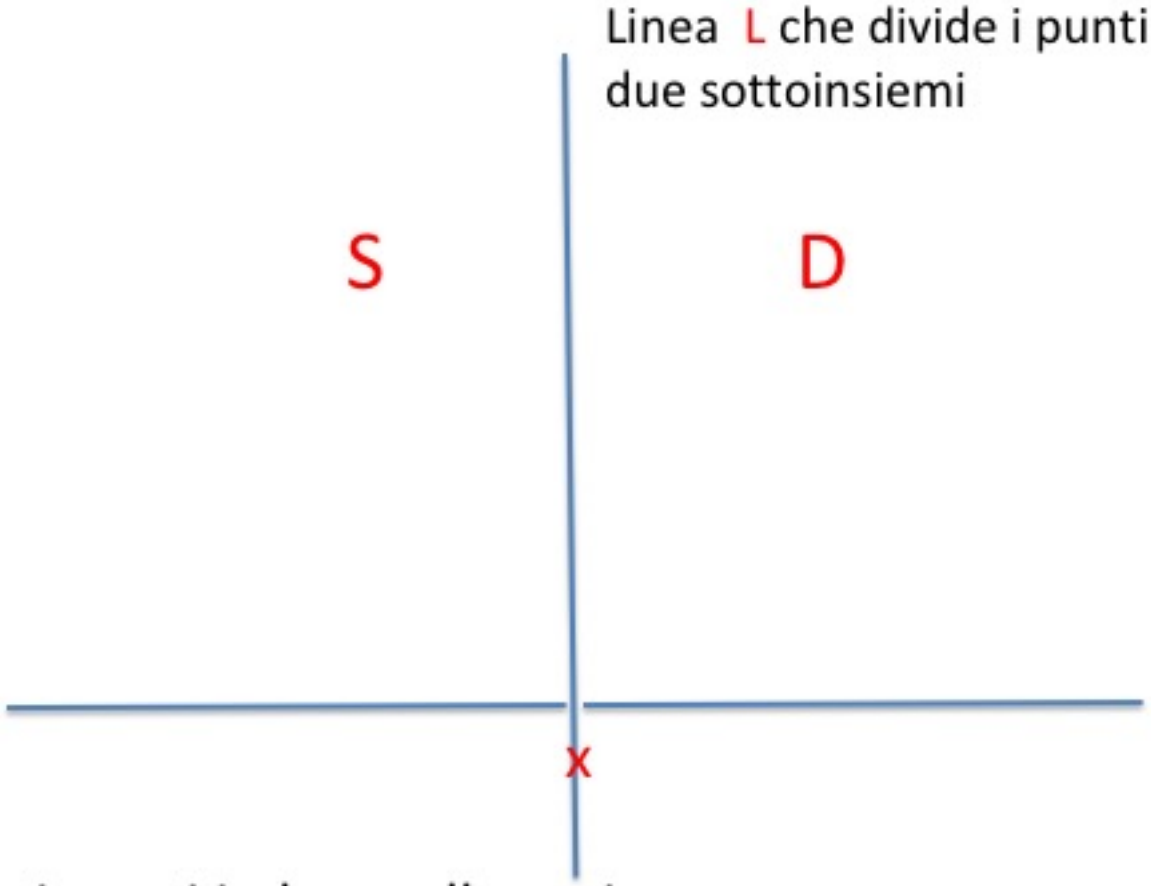
Utilizzando la tecnica del divide et impera, il problema può essere risolto in tempo $O(n \log n)$.

Il problema della coppia più vicina

Idea intuitiva.

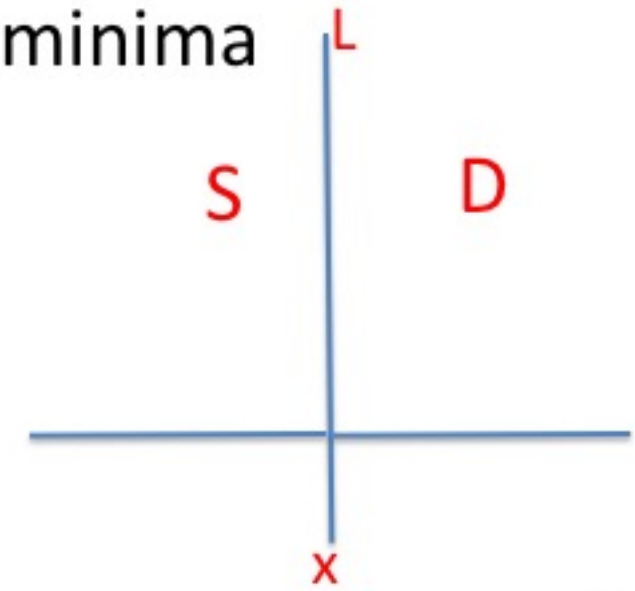
- l'insieme ha cardinalià costante: usiamo la ricerca esaustiva.
- altrimenti: lo dividiamo in due parti uguali S e D, per esempio quelli a sinistra e quelli a destra di una fissata linea verticale
  - troviamo ricorsivamente le soluzioni per l'istanza per S e quella per D individuando due coppie di punti a distanza minima, d_S e d_D
- soluzione finale: o una delle due coppie già individuate oppure può essere formata da un punto in S e uno in D
- se d_{SD} è la minima distanza tra punti aventi estremi in S e D, la soluzione finale è data dalla coppia di punti a distanza $\min\{d_{SD}, d_S, d_D\}$.

Partizione in due sottoinsiemi di n/2 punti ciascuno



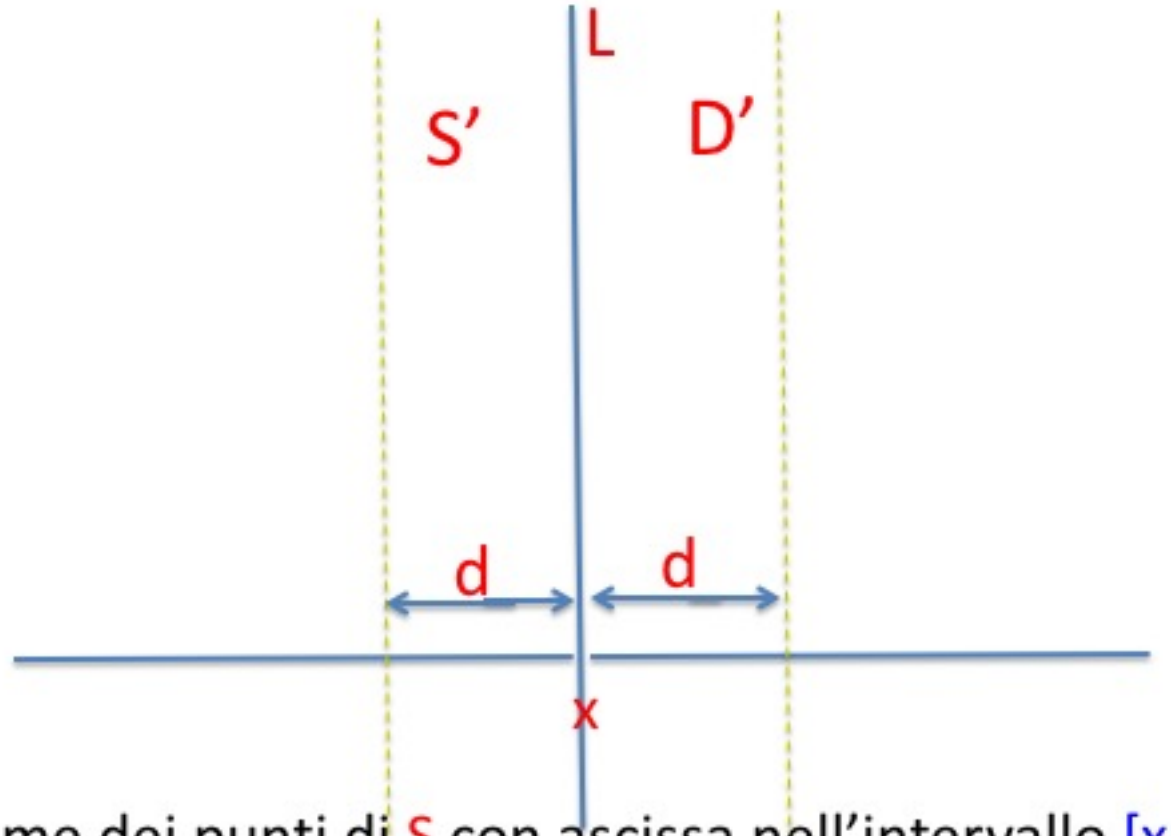
- Ordina i punti in base alle ascisse
- x= ascissa punto **pc** centrale nell'ordinamento
- S= insieme dei punti a sinistra di **pc** nell'ordinamento
- D= insieme dei punti a destra di **pc** nell'ordinamento

Individuazione della coppia di punti a distanza minima



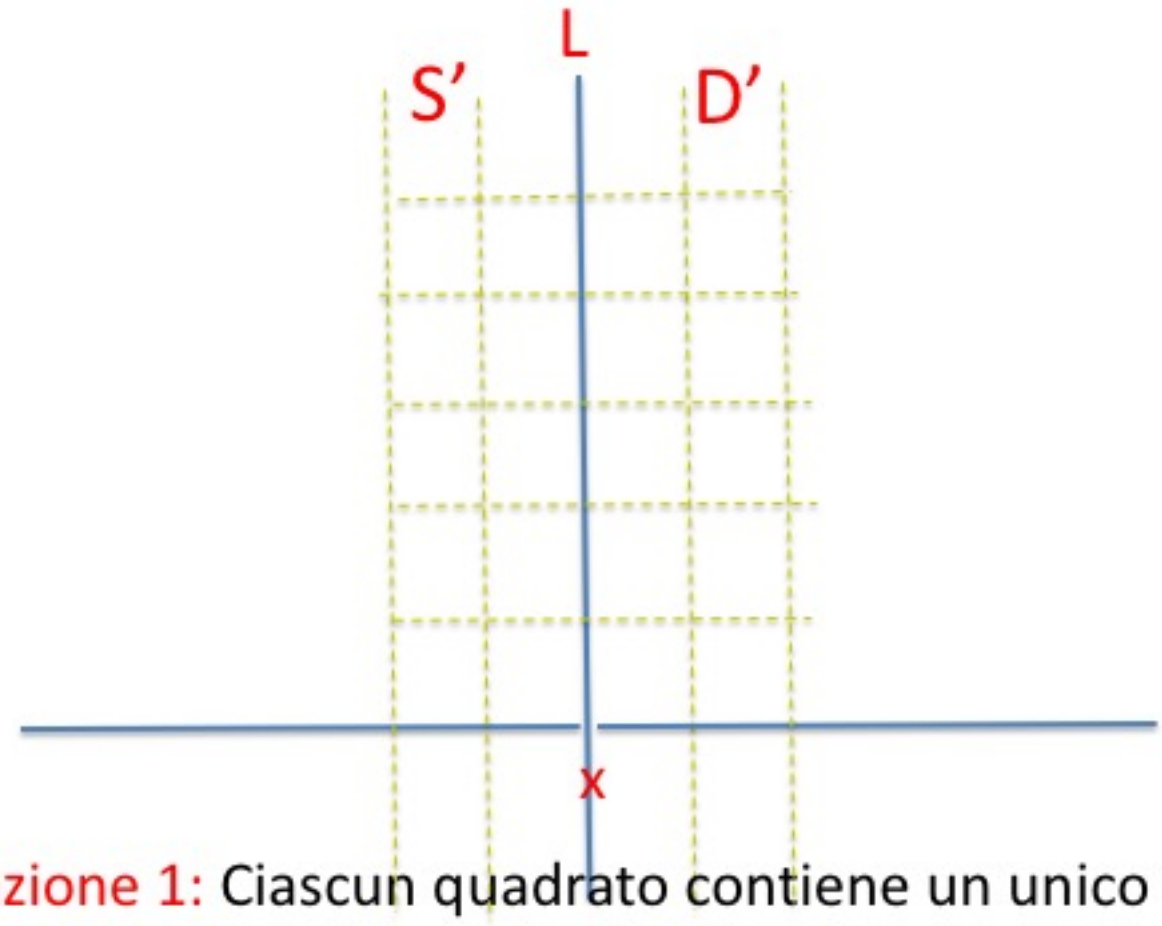
- I 2 punti a distanza minima o sono entrambi in S, o sono entrambi in D, o uno dei due si trova in S e l'altro in D
- Divide et impera:
- Decomposizione: partiziona l'insieme di punti in S e D
- Soluzione sottoproblemi: cerca la coppia a distanza minima d_S in S e la coppia a distanza minima d_D in D. $d=\min\{d_S, d_D\}$
- Ricombinazione: Cerca tra le coppie (p,q) con p in S e q in D quella a distanza minima d_{SD} e resituisce $\min\{d, d_{SD}\}$

Ricerca della coppia (p,q) a distanza minima con p in S e q in D



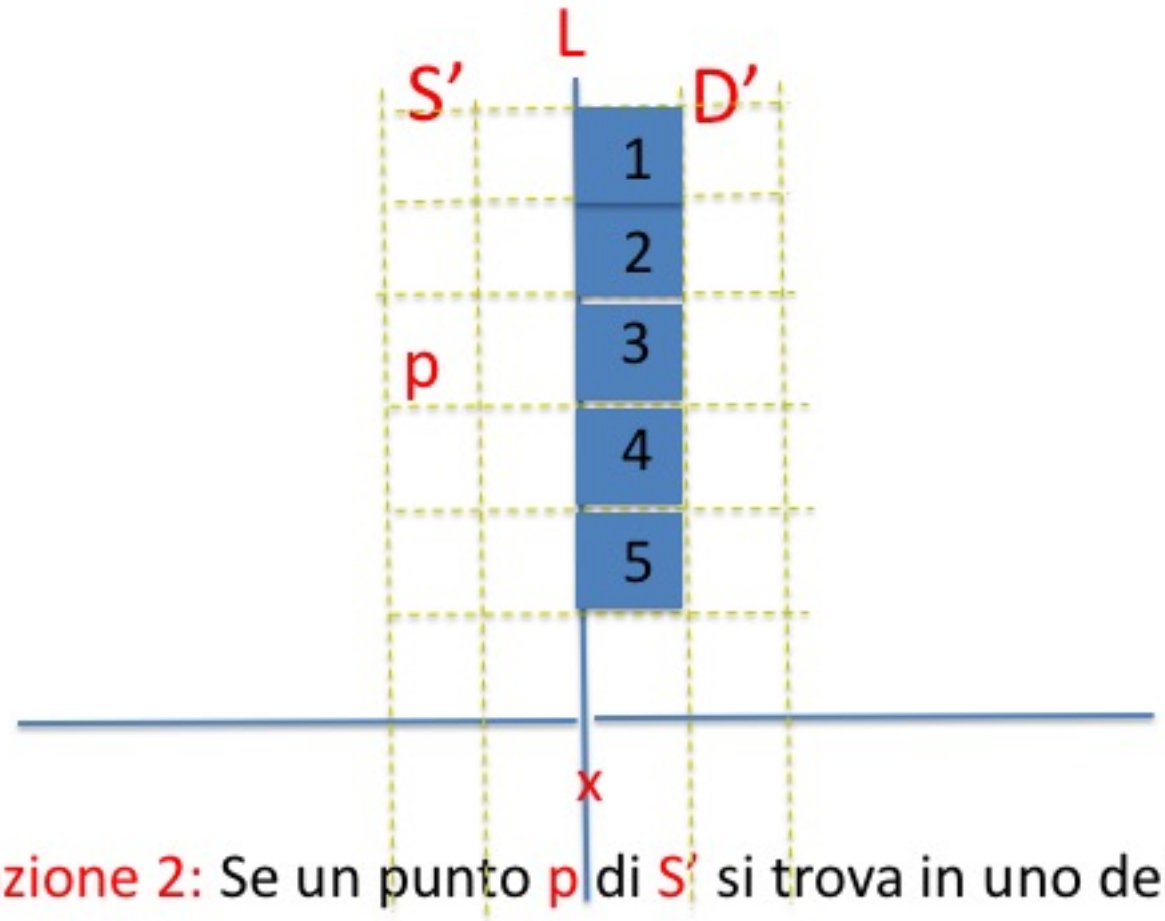
- S'= insieme dei punti di S con ascissa nell'intervallo [x-d,x]
- D'= insieme dei punti di D con ascissa nell'intervallo [x,x+d]
- E` sufficiente considerare coppie (p,q) con p in S' e q in D' ' in quanto le altre coppie (p,q) con p in S e q in D sono a distanza maggiore di d

Dividiamo S’ e D’ in tanti quadrati di lato uguale a d/2



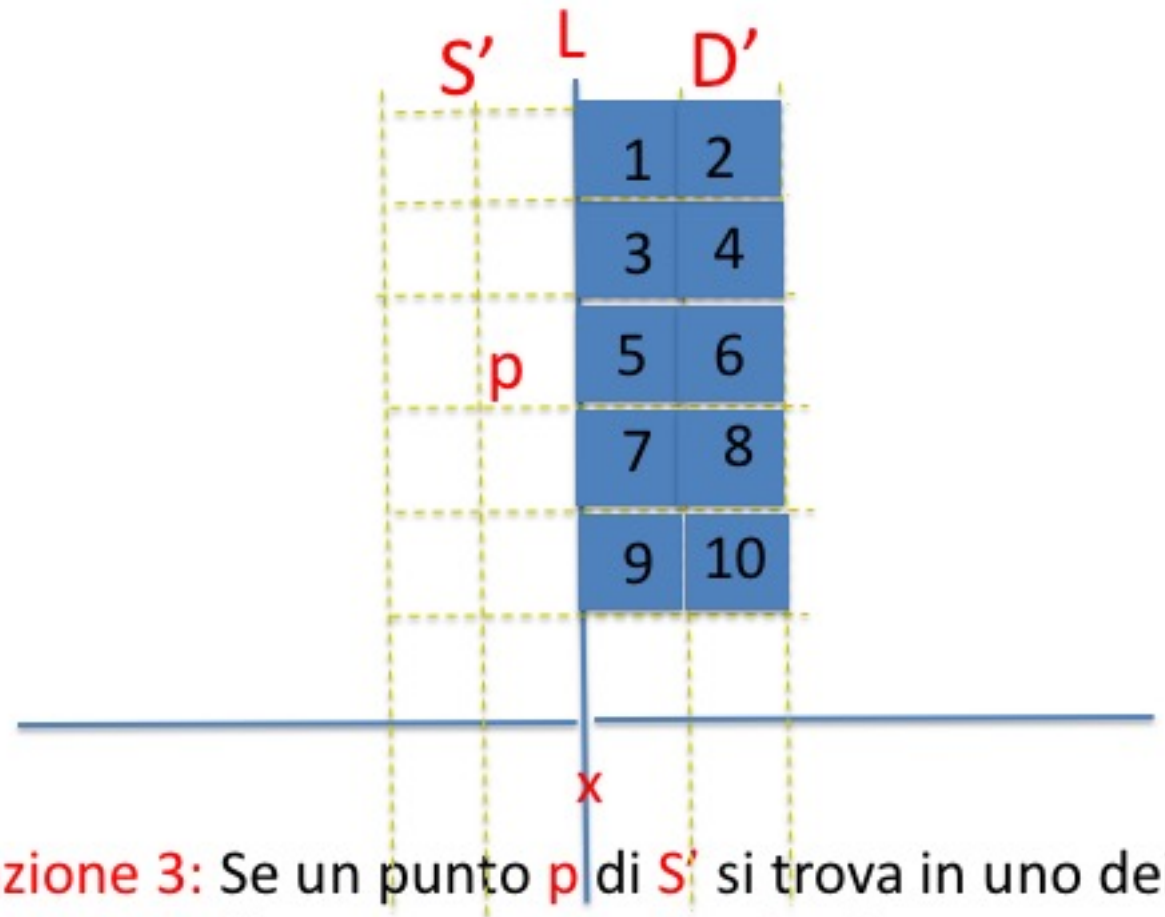
- Osservazione 1: Ciascun quadrato contiene un unico punto altrimenti esisterebbe una coppia di punti entrambi in S’ o entrambi in D’, a distanza minore di d

Dividiamo S’ e D’ in una griglia di quadrati di lato uguale a d/2

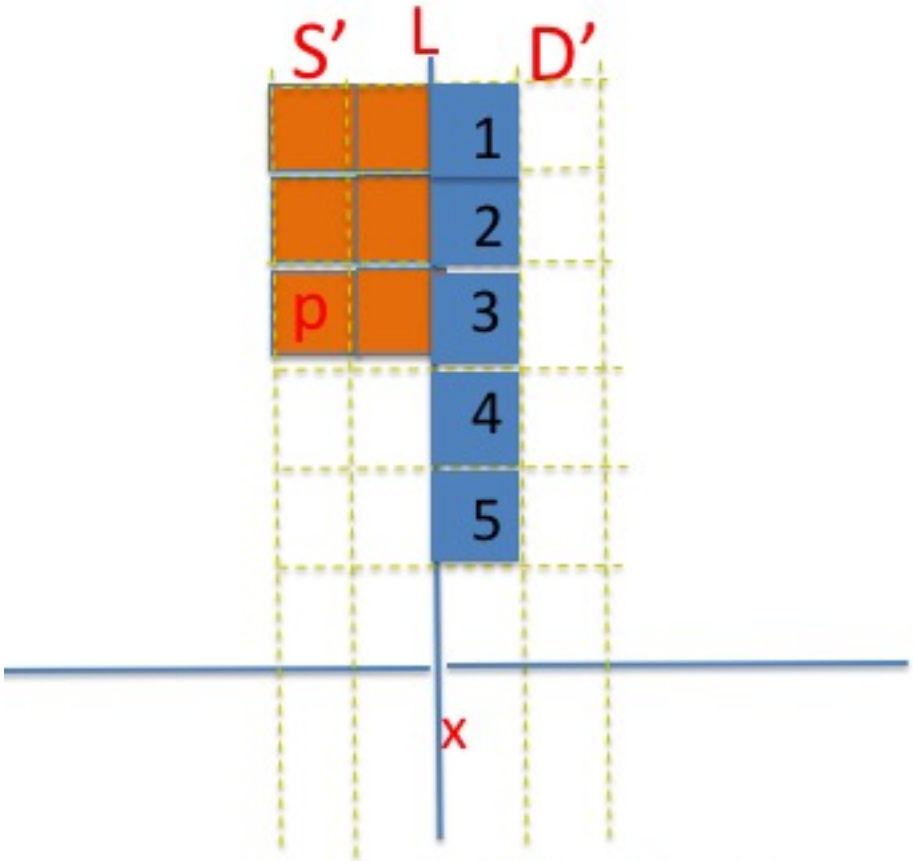


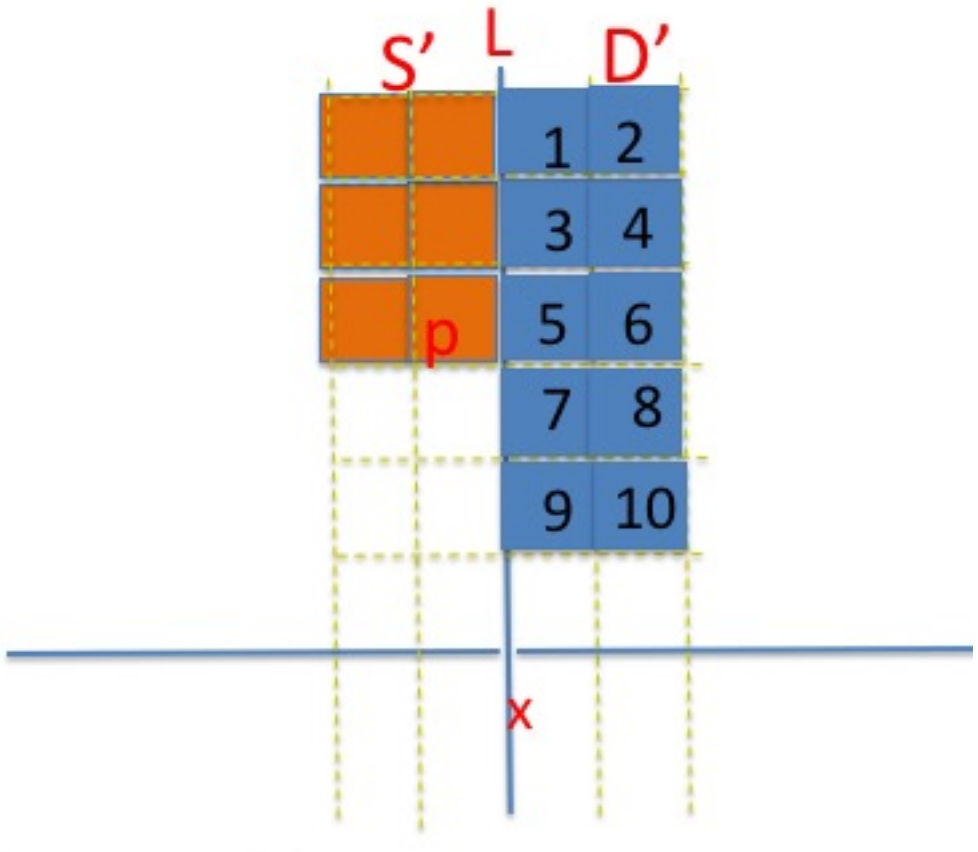
- Osservazione 2: Se un punto p di S’ si trova in uno dei quadrati piu` a sinistra allora un punto q di D’ a distanza minore di d da p puo` trovarsi solo in uno dei quadrati in D’ colorati di azzurro
  - Se q e` piu` in alto rispetto a p allora q si trova in uno dei quadrati 1, 2, 3; altrimenti si trova in uno dei quadrati 3, 4, 5.

Dividiamo S’ e D’ in tanti quadrati di lato uguale a d/2



- Osservazione 3: Se un punto p di S’ si trova in uno dei quadrati confinati con L allora un punto q di D’ a distanza minore di d da p puo` trovarsi solo in uno dei quadrati in D’ colorati di azzurro
  - Se q e` piu` in alto rispetto a p allora q si trova in uno dei quadrati 1-6; altrimenti q si trova in uno dei quadrati 5-10





- P_d = array dei punti di S' e D' in ordine non decrescente di altezza
- Ciascun quadrato contiene al piu` 1 punto →un punto q di D' a distanza al piu` d da p si trova al piu` 11 locazioni in avanti nell'array P_d rispetto a p
  - tra p e q possono esserci al piu` 5 punti di D' e 5 punti di S' (II figura ): ad esempio se q e` piu` in alto rispetto a p allora tra p e q puo` esserci al piu` un punto di D' per ciascuno dei quadrati 1-6 (meno quello contente q) e un punto di S' per ciascuno dei quadrati arancioni (meno quello contente p)

L'algoritmo che trova la coppia più vicina

Input: P_x = array dei punti ordinato in modo non decrescente rispetto alle ascisse; P_y = array dei punti ordinato in modo non decrescente rispetto alle ordinate, n dimensione degli array P_x e P_y

1. Se $n <= 3$, calcola le distanze tra le tre coppie di punti per trovare la coppia a distanza minima.
2. Se $n > 3$, esegue i seguenti passi:
3. Inserisce nell'array S_x i primi $\lfloor n/2 \rfloor$ punti di P_x e nell'array D_x gli ultimi $\lceil n/2 \rceil$ punti di P_x
4. Inserisce nell'array S_y i primi $\lfloor n/2 \rfloor$ punti di P_x nell'ordine in cui appaiono in P_y e nell'array D_y gli ultimi $\lceil n/2 \rceil$ punti di P_x nell'ordine in cui appaiono in P_y
5. Effettua una chiamata ricorsiva con input S_x, S_y e $\lfloor n/2 \rfloor$ e una chiamata ricorsiva con input D_x, D_y e $\lceil n/2 \rceil$. Siano d_S e d_D i valori delle distanze delle coppie di punti restituite dalla prima e dalla seconda chiamata rispettivamente. Pone $d = \min\{d_S, d_D\}$ e (p, q) uguale alla coppia a distanza d.
6. Copia in P_d i punti a distanza minore di d dalla retta verticale passante per l'elemento centrale di P_x nello stesso ordine in cui appaiono in P_y
7. Per ciascun punto p' in P_d esamina gli 11 punti che seguono p' in P_d; per ciascun punto q' (tra questi 11) computa la sua distanza da p' e se questa risulta minore di d, aggiorna il valore di d e pone $(p, q) = (p', q')$
8. Restituisce la coppia (p, q)

Analisi del costo dell'algoritmo che trova coppia più vicina

Assumiamo per semplicità che n sia un potenza di 2

1. Se n è $<= 3$, il costo è limitato superiormente da una certa costante c_0
2. Se n è > 3, il costo dell'algoritmo è così computato:
3. il costo del passo 3 è $O(n)$
4. il costo del passo 4 è $O(n)$: i punti di P_y vengono scanditi a partire dalla prima locazione.e vengono man mano inseriti in S_y o in D_y a seconda che si trovino in locazioni di P_x di indice minore di $\lfloor n/2 \rfloor$ oppure in locazioni di P_x di indice maggiore o uguale di $\lfloor n/2 \rfloor$
5. Il costo delle due chiamate ricorsive è $2T(n/2)$; il costo delle altre operazioni eseguite al passo 5 è costante
6. Il passo 6 richiede tempo $O(n)$: i punti di P_y vengono scanditi a partire dalla prima locazione e quelli la cui ascissa differisce al più d dall'ascissa dell'elemento centrale di S_x vengono man mano inseriti in P_d
7. il passo 7 richiede tempo $O(n)$ perché P_d contiene al più n punti e per ciascuno di essi vengono computate al più 11 distanze, 11 confronti e 11 aggiornamenti di d, p e q.
8. il passo 8 richiede tempo $O(1)$

Costo computazionale dell'algoritmo per la coppia più vicina definito mediante relazione di ricorrenza

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 2 \\ 2T\left(\frac{n}{2}\right) + cn & \text{altrimenti} \end{cases}$$

dove c_0, c sono costanti.

Abbiamo $T(n) = O(n \log n)$.

```
1.  CoppiaPiuVicina(Px,Py,n):
2.  IF(n<=3){Return RicercaEsaustiva(Px,Py,n);}
3.   ELSE{ p=Px[n/2];}
4.  j=k=0;
5.  FOR(i=0;i<n/2;i=i+1){
6.  Sx[i]=Px[i]; Dx[i]=Px[i+n/2];}
7.  FOR(i=0;i<n;i=i+1){
8.  IF(Py[i].x<=p.x){Sy[j]=Py[i]; j=j+1;}
9.  ELSE {Dy[k]=Py[i]; k=k+1;}
10. }
11. (ps,qs)= CoppiaPiuVicina(Sx,Sy,n/2);
12. (pd,qd)=CoppiaPiuVicina(Dx,Dy,(n+1)/2);
13. IF(Dist(ps,qs)<Dist(pd,qd)){d=Dist(ps,qs); (p,q)=(ps,qs);}
14. ELSE {d=Dist(pd,qd); (p,q)=(pd,qd);}
15. FOR(i=m=0;i<n;i=i+1){
16.  IF(|Py[i].x-p.x|<=d){Pd[m]=Py[i]; m=m+1;}
17. }
18. FOR(i=0;i<m;i=i+1){
19.   FOR(j=i+1;j<=min{i+11,m};j=j+1){
20.      IF(Dist(Pd[i],Pd[j])<d){ d=Dist(Pd[i],Pd[j]); (p,q)=(Pd[i],Pd[j]);}
21.  }
22. }
23. RETURN(p,q);
24. }
```

Sottosequenza di somma massima di un array di numeri

Dato un array a di n numeri positivi e negativi trovare la sottosequenza di numeri consecutivi la cui somma è massima. N.B. Se l'array contiene solo numeri positivi, il massimo si ottiene banalmente prendendo come sequenza quella di tutti i numeri dell'array; se l'array contiene solo numeri negativi il massimo si ottiene prendendo come sottosequenza quella formata dalla locazione contenente il numero più grande .

- I soluzione: Per ogni coppia di indici (i, j) con $i \leq j$ dell'array computa la somma degli elementi nella sottosequenza degli elementi di indice compreso tra i e j e restituisci la sottosequenza per cui questa somma è max.
- Costo della I soluzione: $O(n^3)$ perché

$$\begin{aligned}
\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1) &= \sum_{i=0}^{n-1}\sum_{k=1}^{n-i} k = \sum_{i=0}^{n-1}(n-i+1)(n-i)/2 \\
&= \sum_{i=0}^{n-1}((n-i)^2/2+(n-i)/2) = \sum_{a=1}^{n}(a^2/2+a/2) \\
&= \sum_{a=1}^{n} a^2/2 + \sum_{a=1}^{n} a/2 \\
&= 1/2(n(n+1)(2n+1)/6) + 1/2(n(n+1)/2) = \theta(n^3).
\end{aligned}$$

Sottosequenza di somma massima di un array di numeri

- II soluzione Osserviamo che la somma degli elementi di indice compreso tra i e j può essere ottenuta sommando $a[j]$ alla somma degli elementi di indice compreso tra i e j. Di conseguenza, per ogni i, la somma degli elementi in tutte le sottosequenze che partono da i possono essere computate con un costo totale pari a $\theta(n-i)$. Il costo totale è quindi

$$\sum_{i=0}^{n-1} \theta(n-i) = \sum_{i=1}^{n} \theta(i) = \theta(\sum_{i=1}^{n} i) = \theta(n^2)$$

Sottosequenza di somma massima di un array di numeri

- III soluzione: Divide et Impera
  Algoritmo A:
  1. Se $i = j$ viene restituita la sottosequenza formata da $a[i]$
  2. Se $i < j$ si invoca ricorsivamente $A(i, (i+j)/2)$ e $A((i+j)/2+1, j)$: la sottosequenza cercata o è una di quelle restituite dalle 2 chiamate ricorsive o si trova a cavallo delle due metà dell'array
  3. La sottosequenza di somma massima tra quelle che intersecano entrambe le metà dell'array si trova nel seguente modo:
     - si scandisce l'array a partire dall'indice $(i+j)/2$ andando a ritroso fino a che si arriva all'inizio dell'array sommando via via gli elementi scanditi: ad ogni iterazione si confronta la somma ottenuta fino a quel momento con il valore max s_1 delle somme ottenute in precedenza e nel caso aggiorna il max s_1 e l'indice in corrispondenza del quale è stato ottenuto.
     - si scandisce l'array a partire dal'indice $(i+j)/2+1$ andando in avanti fino a che o si raggiunge la fine dell'array sommando gli elementi scanditi: ad ogni iterazione si confronta la somma ottenuta fino a quel momento con il valore max s_2 delle somme ottenute in precedenza e nel caso aggiorna il max s_2 e l'indice in corrispondenza del quale è stato ottenuto.
     - La sottosequenza di somma massima tra quelle che intersecano le due metà dell'array è quella di somma $s_1 + s_2$.
  4. L'algoritmo restituisce la sottosequenza massima tra quella restituita dalla prima chiamata ricorsiva, quella restituita dalla seconda chiamata ricorsiva e quella di somma $s_1 + s_2$

Sottosequenza di somma massima di un array di numeri

- Tempo di esecuzione dell'algoritmo Divide et Impera

$$T(n) \leq \begin{cases} c_0 & \text{se } n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{altrimenti} \end{cases}$$

  Il tempo di esecuzione quindi è $O(n \log n)$.

Sottosequenza di somma massima di un array di numeri

- IV soluzione: Chiamiamo s_j la sottosequenza di somma massima che tra quelle che terminano in j. Si ha $s_{j+1} = \max\{s_j + a[j+1], a[j+1]\}$. Questo valore si calcola in tempo costante per ogni j. L'algoritmo calcola questi valori per ogni j e prende il massimo degli n valori computati. Il tempo dell'algoritmo quindi è $O(n)$.

Teorema fondamentale delle ricorrenze

Data la relazione di ricorrenza

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq n_0 \\ \alpha T(n/\beta) + cf(n) & \text{altrimenti} \end{cases}$$

dove $f(n)$ è una funzione non decrescente e $\alpha \geq 1$, $\beta > 1$,$n_0 \geq 1$ e $c_0, c > 0$.

Se esistono due costanti positive γ e n_0' tali che $\alpha f(n/\beta) = \gamma f(n)$ per ogni $n \geq n_0'$, allora la relazione di ricorrenza ha le seguenti soluzioni per ogni n:

1. $T(n) = O(f(n))$ se $\gamma < 1$;
2. $T(n) = O(f(n) \log_\beta n)$ se $\gamma = 1$;
3. $T(n) = O(n^{\log_\beta \alpha})$ se $\gamma > 1$.

Teorema fondamentale delle ricorrenze

Dimostrazione. Assumiamo che n sia potenza di β ($n = \beta^k$, per qualche k) e applichiamo iterativamente la relazione di ricorrenza.

$$\begin{aligned}
T(n) &\leq cf(n) + \alpha T(n/\beta) \leq cf(n) + \alpha(cf(n/\beta) + \alpha T(n/\beta^2)) \\
&= cf(n) + \alpha cf(n/\beta) + \alpha^2 T(n/\beta^2) \\
&\leq cf(n) + \alpha cf(n/\beta) + \alpha^2(cf(n/\beta^2) + \alpha T(n/\beta^3)) \\
&= cf(n) + \alpha cf(n/\beta) + \alpha^2 cf(n/\beta^2) + \alpha^3 T(n/\beta^3) \\
&\leq \ldots \\
&\leq cf(n) + \alpha cf(n/\beta) + \alpha^2(cf(n/\beta^2) + \ldots + \alpha^{i-1}(cf(n/\beta^{i-1}) + \alpha^i T(n/\beta^i))
\end{aligned}$$

Per $i = \log_\beta n$ si ha

$$\begin{aligned}
T(n) &\leq cf(n) + \alpha cf(n/\beta)) + \ldots + \alpha^{(\log_\beta n - 1)}(cf(n/\beta^{(\log_\beta n - 1)}) + \alpha^{\log_\beta n} T(n/\beta^{\log_\beta n})) \\
&= \sum_{j=0}^{\log_\beta n - 1} \alpha^j(cf(n/\beta^j) + \alpha^{\log_\beta n} T(1) \leq \sum_{j=0}^{\log_\beta n - 1} \alpha^j(cf(n/\beta^j) + \alpha^{\log_\beta n} c_0.
\end{aligned}$$

L'ipotesi $\alpha\, f(n/\beta) = \gamma\, f(n)$ implica

$$\alpha^i\, f(n/\beta^i) = \gamma^i\, f(n) \tag{1}$$

(quest'ultima uguaglianza è dimostrata per induzione alla fine della dimostrazione del Teorema). Si noti che nell'analisi di un algoritmo Divide et Impera, $\alpha^i\, f(n/\beta^i)$ rappresenta il tempo per eseguire il lavoro di decomposizione e ricombinazione nelle chiamate ricorsive a profondità $i-1$.

L'ultima disuguaglianza nella slide precedente e la (1) implicano che

$$T(n) \leq c \sum_{j=0}^{\log_\beta n - 1} \alpha^j (f(n/\beta^j) + \alpha^{\log_\beta n} c_0 = c \sum_{j=0}^{\log_\beta n - 1} \gamma^j\, f(n) + \alpha^{\log_\beta n} c_0.$$

Siccome $f(n/\beta^{\log_\beta n}) = f(1)$ allora possiamo scrivere $\alpha^{\log_\beta n} = \frac{\alpha^{\log_\beta n}\, f(n/\beta^{\log_\beta n})}{f(1)}$.

Si noti che per $i = \log_\beta n$ la (1) implica $\alpha^{\log_\beta n}\, f(n/\beta^{\log_\beta n}) = \gamma^{\log_\beta n}\, f(n)$. Si ha quindi

$$\alpha^{\log_\beta n} = \frac{\alpha^{\log_\beta n}\, f(n/\beta^{\log_\beta n})}{f(1)} = \frac{\gamma^{\log_\beta n}\, f(n)}{f(1)}$$

e quindi

$$c_0 \alpha^{\log_\beta n} = O(\gamma^{\log_\beta n}\, f(n)).$$

Ne consegue che

$$T(n) \leq c_0 f(n) \sum_{j=0}^{\log_\beta n - 1} \gamma^j + O(f(n)\gamma^{\log_\beta n}) = O\left(f(n) \sum_{j=0}^{\log_\beta n} \gamma^j\right). \qquad (2)$$

Esaminiamo i tre casi $\gamma < 1$, $\gamma = 1$ e $\gamma > 1$:

1. Caso 1: $\gamma < 1$

$$\sum_{j=0}^{\log_\beta n} \gamma^j = \frac{\gamma^{\log_\beta n + 1} - 1}{\gamma - 1} = \frac{1 - \gamma^{\log_\beta n + 1}}{1 - \gamma} < \frac{1}{1 - \gamma} = O(1)$$

   per cui dalla (2) si ha che

$$T(n) = O(f(n))$$

2. Caso 2: $\gamma = 1$

$$\sum_{j=0}^{\log_\beta n} \gamma^j = 1 + \log_\beta n$$

   per cui dalla (2) si ha che

$$T(n) = O(f(n) \log_\beta n)).$$

1. Caso 3: $\gamma > 1$.

$$\sum_{j=0}^{\log_\beta n} \gamma^j = \frac{\gamma^{\log_\beta n+1} - 1}{\gamma - 1}$$

Si noti che $\frac{\gamma^{\log_\beta n+1}-1}{\gamma-1} < \frac{\gamma^{\log_\beta n+1}}{\gamma-1} = \frac{\gamma}{\gamma-1}\gamma^{\log_\beta n} = O(\gamma^{\log_\beta n})$, dal momento che $\frac{\gamma}{\gamma-1}$ è una costante positiva.

Si ha quindi

$$\sum_{j=0}^{\log_\beta n} \gamma^j = \frac{\gamma^{\log_\beta n+1} - 1}{\gamma - 1} = O(\gamma^{\log_\beta n})$$

per cui dalla (2) si ha che

$$T(n) = O(\gamma^{\log_\beta n} f(n)).$$

Inoltre dalla (1) si ha che

$$\gamma^{\log_\beta n} f(n) = \alpha^{\log_\beta n} f(n/\beta^{\log_\beta n}) = \alpha^{\log_\beta n} f(1)$$

e quindi

$$T(n) \quad = \quad O(\alpha^{\log_\beta n} f(1)) = O(\alpha^{\log_\beta n}) = O(\alpha^{\log_\alpha n \log_\beta \alpha}) = O(n^{\log_\beta \alpha}).$$

Dimostrazione dell'uguaglianza $\alpha^i f(n/\beta^i) = \gamma^i f(n)$

- Base dell'induzione $0 \leq i \leq 1$: per $i = 0$ l'uguaglianza vale banalmente; per $i = 1$ discende immediatamente dall'ipotesi $\alpha f(n/\beta) = \gamma f(n)$.
- Passo Induttivo: assumiamo che l'uguaglianza valga per i, con $i \geq 1$, e dimostriamo che vale per $i + 1$:

  $$\alpha^{i+1} f(n/\beta^{i+1}) = \alpha^i \alpha f((n/\beta^i)/\beta).$$

  Applicando l'ipotesi del teorema $\alpha f(n/\beta) = \gamma f(n)$ con n/β^i al posto di n si ha $\alpha f((n/\beta^i)/\beta) = \gamma f(n/\beta^i)$, da cui si ha

  $$\alpha^{i+1} f(n/\beta^{i+1}) = \alpha^i \alpha f((n/\beta^i)/\beta) = \alpha^i \gamma f(n/\beta^i)$$

  e applicando l'ipotesi induttiva si ha $\alpha^i f(n/\beta^i) = \gamma^i f(n)$ per cui $\alpha^i \gamma f(n/\beta^i) = \gamma\gamma^i f(n) = \gamma^{i+1} f(n)$.

Dimostrazione del teorema fondamentale quando n non è potenza di β

- Si k l'intero più piccolo per cui $n \leq \beta^k$. In altre parole, $\beta^{k-1} < n \leq \beta^k$.
- Siccome $f(n)$ è non decrescente allora

$$f(n) \geq f(\beta^{k-1}) = f\left(\frac{\beta^k}{\beta}\right). \tag{3}$$

L'uguaglianza $\alpha f(n/\beta) = \gamma f(n)$, $\forall n \geq n_0'$, implica

$$\alpha f\left(\frac{\beta^k}{\beta}\right) = \gamma f(\beta^k), \ \ \forall \beta^k \geq n_0'. \tag{4}$$

Le (3) e (4) implicano

$$f(n) \geq \frac{\gamma}{\alpha} f(\beta^k), \quad \forall \beta^k \geq n_0'. \tag{5}$$

Siccome $n \geq n_0'$ implica $\beta^k \geq n_0'$ allora si ha che la disuguaglianza nella (5) vale per ogni $n \geq n_0'$. In altre parole

$$f(\beta^k) = O(f(n)).$$

Dimostrazione del teorema fondamentale quando n non è potenza di β

- Nelle slide precedenti, abbiamo dimostrato che il teorema vale quando l'argomento di $T()$ è potenza di β per cui si ha

$$T(\beta^k) = \begin{cases} O(f(\beta^k)) & \text{se } \gamma < 1 \\ O(f(\beta^k)\log_\beta \beta^k) = O(f(\beta^k)k) & \text{se } \gamma = 1 \\ O((\beta^k)^{\log_\beta \alpha}) & \text{se } \gamma > 1 \end{cases}$$

- Siccome abbiamo appena visto che $f(\beta^k) = O(f(n))$ e siccome $\beta^{k-1} < n$, allora

$$T(\beta^k) = \begin{cases} O(f(\beta^k)) = O(f(n)) & \text{se } \gamma < 1 \\ O(f(\beta^k)\log_\beta \beta^k) = O(f(n)k) = O(f(n)(k-1)) = O(f(n)\log_\beta n) & \text{se } \gamma = 1 \\ O((\beta^k)^{\log_\beta \alpha}) = O((\beta\beta^{k-1})^{\log_\beta \alpha}) = O(\beta^{\log_\beta \alpha}(\beta^{k-1})^{\log_\beta \alpha}) = O((\beta^{k-1})^{\log_\beta \alpha}) = O(n^{\log_\beta \alpha}) & \text{se } \gamma > 1 \end{cases}$$

- Siccome $T()$ è decrescente allora $T(n) \leq T(\beta^k)$ e le relazione asintotiche sopra riportate valgono anche per $T(n)$.

Teorema fondamentale: esempi

- Ricerca binaria

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \text{ oppure } k \text{ è l'elemento centrale} \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

Si ha $\alpha = 1, \beta = 2$, $f(n) = 1$.

Siccome $1 \cdot f(\frac{n}{2}) = 1 = 1 \cdot f(n)$, siamo nel caso $\gamma = 1$ (con $n_0' = 1$) e si ha

$$T(n) = O(f(n) \log_\beta n)) = O(\log n).$$

Teorema fondamentale: esempi

Nell'ordinamento per fusione,

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 2T(n/2) + cn & \text{altrimenti} \end{cases}$$

Quindi,

- $\alpha = 2$, $\beta = 2$ e $f(n) = n$
- siamo nel caso $\gamma = 1$ (con $n_0' = 1$), in quanto $\alpha f(n/\beta) = 2(n/2) = n = f(n)$.
- il numero di passi è $O(n \log_2 n) = O(n \log n)$.

Teorema fondamentale: esempi

- Moltiplicazione veloce di interi: primo algoritmo

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 4T(n/2) + cn & \text{altrimenti} \end{cases}$$

  Applicazione del teorema fondamentale delle ricorrenze

  - si ha che $\alpha = 4$, $\beta = 2$ e $f(n) = n$
  - $\alpha f(n/\beta) = 4(n/2) = 2n = 2f(n)$, quindi si applica il terzo caso del teorema con $\gamma = 2$ (ed $n_0' = 1$).
  - ne deriva $O(n^{\log_2 4}) = O(n^2)$

- Moltiplicazione veloce di interi: secondo algoritmo

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 3T(n/2) + cn & \text{altrimenti} \end{cases}$$

  Applicando il teorema fondamentale delle ricorrenze,

  - si ha che $\alpha = 3$, $\beta = 2$ e $f(n) = n$
  - $\alpha f(n/\beta) = 3(n/2) = \frac{3}{2}n = \frac{3}{2}f(n)$, quindi si applica il terzo caso del teorema con $\gamma = \frac{3}{2}$ ed $n_0' = 1$
  - ne deriva $O(n^{\log_2 3}) = O(n^{1,585})$