Lezione 2

Sommario della Lezione

Primi Esempi di Progettazione di Algoritmi

Siano dati un intero $n \geq 1$ ed una sequenza $a = a[0]a[1] \cdots a[n-1]$ di numeri (che possono essere sia positivi o negativi),

Siano dati un intero $n \geq 1$ ed una sequenza $a = a[0]a[1] \cdots a[n-1]$ di numeri (che possono essere sia positivi o negativi), per ogni coppia di indici $0 \leq i \leq j \leq n-1$, denotiamo con $V(i,j)$ la somma

$$V(i,j) = a[i] + a[i+1] + \ldots + a[j]$$

Siano dati un intero $n \geq 1$ ed una sequenza $a = a[0]a[1] \cdots a[n-1]$ di numeri (che possono essere sia positivi o negativi), per ogni coppia di indici $0 \leq i \leq j \leq n-1$, denotiamo con $V(i,j)$ la somma

$$V(i,j) = a[i] + a[i+1] + \ldots + a[j] = \sum_{k=i}^{j} a[k],$$

Siano dati un intero $n \geq 1$ ed una sequenza $a = a[0]a[1] \cdots a[n-1]$ di numeri (che possono essere sia positivi o negativi), per ogni coppia di indici $0 \leq i \leq j \leq n-1$, denotiamo con $V(i,j)$ la somma

$$V(i,j) = a[i] + a[i+1] + \ldots + a[j] = \sum_{k=i}^{j} a[k],$$

Denotiamo con $V(a)$ la quantità

$$V(a) = \max_{0 \leq i \leq j \leq n-1} V(i,j),$$

Siano dati un intero $n \geq 1$ ed una sequenza $a = a[0]a[1] \cdots a[n-1]$ di numeri (che possono essere sia positivi o negativi), per ogni coppia di indici $0 \leq i \leq j \leq n-1$, denotiamo con $V(i,j)$ la somma

$$V(i,j) = a[i] + a[i+1] + \ldots + a[j] = \sum_{k=i}^{j} a[k],$$

Denotiamo con $V(a)$ la quantità

$$V(a) = \max_{0 \leq i \leq j \leq n-1} V(i,j),$$

cioè $V(a)$ è il *masssimo valore* che possiamo ottenere sommando sottosequenze di elementi *consecutivi* della sequenza a.

Siano dati un intero $n \geq 1$ ed una sequenza $a = a[0]a[1] \cdots a[n-1]$ di numeri (che possono essere sia positivi o negativi), per ogni coppia di indici $0 \leq i \leq j \leq n-1$, denotiamo con $V(i,j)$ la somma

$$V(i,j) = a[i] + a[i+1] + \ldots + a[j] = \sum_{k=i}^{j} a[k],$$

Denotiamo con $V(a)$ la quantità

$$V(a) = \max_{0 \leq i \leq j \leq n-1} V(i,j),$$

cioè $V(a)$ è il *masssimo valore* che possiamo ottenere sommando sottosequenze di elementi *consecutivi* della sequenza a.

Problema computazionale:
Input: sequenza di numeri $a = a[0]a[1] \cdots a[n-1]$

Output: $V(a) = \max_{0 \leq i \leq j \leq n-1} V(i,j) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k]$.

Esempio: $a = a[0]a[1]\cdots a[15]$ dato da

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

Esempio: $a = a[0]a[1]\cdots a[15]$ dato da

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

Abbiamo varie possibili soluzioni, ad es.

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

In questo caso $S_1 = [3, 6]$ con valore $3 + 6 = 9$, e $S_2 = [2, 6, 1]$ con valore $2 + 6 + 1 = 9$.

Esempio: $a = a[0]a[1]\cdots a[15]$ dato da

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

Abbiamo varie possibili soluzioni, ad es.

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

In questo caso $S_1 = [3, 6]$ con valore $3 + 6 = 9$, e $S_2 = [2, 6, 1]$ con valore $2 + 6 + 1 = 9$.
Oppure

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

con valore della soluzione pari a 3+6-1+2-4+7=13.

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a,

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$,

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

1. `MaxConsecutivaSomma1`(a)

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

1. `MaxConsecutivaSomma1`(a)
2. `SommaMassima`$= a[0]$

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

1. `MaxConsecutivaSomma1`(a)
2. `SommaMassima`$= a[0]$
3. `FOR`$(i = 0; i < n; i = i + 1)\{$

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

1. `MaxConsecutivaSomma1`(a)
2. `SommaMassima`$= a[0]$
3. `FOR`$(i = 0; i < n; i = i + 1)\{$
4. `FOR`$(j = i; j < n; j = j + 1)\{$

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.     FOR(j = i; j < n; j = j + 1){
5.         SommaCorrente= 0
```

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.     FOR(j = i; j < n; j = j + 1){
5.         SommaCorrente= 0
6.         FOR(k = i; k < j + 1; k = k + 1){
```

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.     FOR(j = i; j < n; j = j + 1){
5.         SommaCorrente= 0
6.         FOR(k = i; k < j + 1; k = k + 1){
7.             SommaCorrente=SommaCorrente+a[k]
```

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.     FOR(j = i; j < n; j = j + 1){
5.         SommaCorrente= 0
6.         FOR(k = i; k < j + 1; k = k + 1){
7.             SommaCorrente=SommaCorrente+a[k]
8.         }
9.         IF(SommaCorrente>SommaMassima){
10.            SommaMassima=SommaCorrente
        }
    }
}
```

Primo possibile algoritmo

Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a, ovvero calcoliamo tutti i possibili valori $V(i,j) = \sum_{k=i}^{j} a[k]$, per ogni coppia di indici (i,j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima.

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.     FOR(j = i; j < n; j = j + 1){
5.         SommaCorrente= 0
6.         FOR(k = i; k < j + 1; k = k + 1){
7.             SommaCorrente=SommaCorrente+a[k]
8.         }
9.         IF(SommaCorrente>SommaMassima){
10.            SommaMassima=SommaCorrente
        }
    }
}
11. RETURN SommaMassima
```

Analisi dell'Algoritmo

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.       FOR(j = i; j < n; j = j + 1){
5.           SommaCorrente= 0
6.           FOR(k = i; k < j + 1; k = k + 1){
7.               SommaCorrente=SommaCorrente+a[k]
8.           }
9.           IF(SommaCorrente>SommaMassima){
10.             SommaMassima=SommaCorrente
          }
     }
}
11. RETURN SommaMassima
```

Analisi dell'Algoritmo

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.       FOR(j = i; j < n; j = j + 1){
5.           SommaCorrente= 0
6.           FOR(k = i; k < j + 1; k = k + 1){
7.               SommaCorrente=SommaCorrente+a[k]
8.           }
9.           IF(SommaCorrente>SommaMassima){
10.              SommaMassima=SommaCorrente
          }
     }
}
11. RETURN SommaMassima
```

Il ciclo FOR delle istruzioni 6. e 7. calcola $V(i,j) = \sum_{k=i}^{j} a[k]$.

Analisi dell'Algoritmo

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.       FOR(j = i; j < n; j = j + 1){
5.           SommaCorrente= 0
6.           FOR(k = i; k < j + 1; k = k + 1){
7.               SommaCorrente=SommaCorrente+a[k]
8.           }
9.           IF(SommaCorrente>SommaMassima){
10.              SommaMassima=SommaCorrente
          }
     }
}
11. RETURN SommaMassima
```

Il ciclo FOR delle istruzioni 6. e 7. calcola $V(i,j) = \sum_{k=i}^{j} a[k]$. Le istruzioni all'interno del FOR$(k = i; k < j + 1; k = k + 1)$ possono essere eseguite al più n volte,

Analisi dell'Algoritmo

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.       FOR(j = i; j < n; j = j + 1){
5.           SommaCorrente= 0
6.           FOR(k = i; k < j + 1; k = k + 1){
7.               SommaCorrente=SommaCorrente+a[k]
8.           }
9.           IF(SommaCorrente>SommaMassima){
10.              SommaMassima=SommaCorrente
          }
     }
}
11. RETURN SommaMassima
```

Il ciclo FOR delle istruzioni 6. e 7. calcola $V(i,j) = \sum_{k=i}^{j} a[k]$. Le istruzioni all'interno del $\texttt{FOR}(k = i; k < j + 1; k = k + 1)$ possono essere eseguite al più n volte, così come il codice all'interno del ciclo $\texttt{FOR}(j = i; j < n; j = j+1)$

Analisi dell'Algoritmo

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.       FOR(j = i; j < n; j = j + 1){
5.           SommaCorrente= 0
6.           FOR(k = i; k < j + 1; k = k + 1){
7.               SommaCorrente=SommaCorrente+a[k]
8.           }
9.           IF(SommaCorrente>SommaMassima){
10.              SommaMassima=SommaCorrente
          }
     }
}
11. RETURN SommaMassima
```

Il ciclo FOR delle istruzioni 6. e 7. calcola $V(i,j) = \sum_{k=i}^{j} a[k]$. Le istruzioni all'interno del $\texttt{FOR}(k = i; k < j + 1; k = k + 1)$ possono essere eseguite al più n volte, così come il codice all'interno del ciclo $\texttt{FOR}(j = i; j < n; j = j+1)$ ed anche del ciclo $\texttt{FOR}(i = 0; i < n; i = i + 1)$,

Analisi dell'Algoritmo

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.       FOR(j = i; j < n; j = j + 1){
5.           SommaCorrente= 0
6.           FOR(k = i; k < j + 1; k = k + 1){
7.               SommaCorrente=SommaCorrente+a[k]
8.           }
9.           IF(SommaCorrente>SommaMassima){
10.              SommaMassima=SommaCorrente
          }
     }
}
11. RETURN SommaMassima
```

Il ciclo FOR delle istruzioni 6. e 7. calcola $V(i,j) = \sum_{k=i}^{j} a[k]$. Le istruzioni all'interno del $\texttt{FOR}(k = i; k < j + 1; k = k + 1)$ possono essere eseguite al più n volte, così come il codice all'interno del ciclo $\texttt{FOR}(j = i; j < n; j = j+1)$ ed anche del ciclo $\texttt{FOR}(i = 0; i < n; i = i + 1)$, per cui l'algoritmo $\texttt{MaxConsecutivaSomma1}(a)$ eseguirà al più $c \times n \times n \times n = cn^3$ operazioni elementari.

Analisi dell'Algoritmo

```
1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.       FOR(j = i; j < n; j = j + 1){
5.           SommaCorrente= 0
6.           FOR(k = i; k < j + 1; k = k + 1){
7.               SommaCorrente=SommaCorrente+a[k]
8.           }
9.           IF(SommaCorrente>SommaMassima){
10.              SommaMassima=SommaCorrente
          }
     }
}
11. RETURN SommaMassima
```

Il ciclo FOR delle istruzioni 6. e 7. calcola $V(i,j) = \sum_{k=i}^{j} a[k]$. Le istruzioni all'interno del $\texttt{FOR}(k = i; k < j + 1; k = k + 1)$ possono essere eseguite al più n volte, così come il codice all'interno del ciclo $\texttt{FOR}(j = i; j < n; j = j+1)$ ed anche del ciclo $\texttt{FOR}(i = 0; i < n; i = i + 1)$, per cui l'algoritmo $\texttt{MaxConsecutivaSomma1}(a)$ eseguirà al più $c \times n \times n \times n = cn^3$ operazioni elementari.

In altri termini, la complessità dell'algoritmo $\texttt{MaxConsecutivaSomma1}(a)$ su di un input composto da una sequenza di n numeri è $O(n^3)$.

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*.

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] =$$

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] =$$

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$,

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il $\texttt{FOR}(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il `FOR`$(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
```

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il $\texttt{FOR}(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
```

Secondo possibile algoritmo

Calcoliamo sempre la somma di `tutte` le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il `FOR`$(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
```

Secondo possibile algoritmo

Calcoliamo sempre la somma di tutte le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il FOR$(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
```

Secondo possibile algoritmo

Calcoliamo sempre la somma di tutte le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il FOR$(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
         SommaMassima=SommaCorrente
```

Secondo possibile algoritmo

Calcoliamo sempre la somma di tutte le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il FOR$(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
         SommaMassima=SommaCorrente
        } } }
```

Secondo possibile algoritmo

Calcoliamo sempre la somma di tutte le possibili sottosequenze di elementi consecutivi di $a = a[0]a[1]\cdots a[n-1]$, ma in maniera *più furba*. Osserviamo che

$$V(i,j) = \sum_{k=i}^{j} a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i,j-1) + a[j]. \qquad (1)$$

L'osservazione (1) ci permette di risparmiare tempo, in quanto con *una sola* somma possiamo calcolare $V(i,j)$ come $V(i,j) = V(i,j-1) + a[j]$, invece di eseguire il FOR$(k = i; k < j+1; k = k+1)$, come facevamo nel precedente algoritmo.

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
         SommaMassima=SommaCorrente
        } } }
RETURN SommaMassima
```

Analisi dell'Algoritmo

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
          SommaMassima=SommaCorrente
         } } }
RETURN SommaMassima
```

Analisi dell'Algoritmo

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
          SommaMassima=SommaCorrente
         } } }
RETURN SommaMassima
```

Le istruzioni all'interno del FOR($j = i; j < n; j = j + 1$) possono essere eseguite al più n volte,

Analisi dell'Algoritmo

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
          SommaMassima=SommaCorrente
        } } }
RETURN SommaMassima
```

Le istruzioni all'interno del FOR($j = i; j < n; j = j + 1$) possono essere eseguite al più n volte, così come le istruzioni all'interno del FOR($i = 0; i < n; i = i + 1$),

Analisi dell'Algoritmo

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
          SommaMassima=SommaCorrente
         } } }
RETURN SommaMassima
```

Le istruzioni all'interno del FOR($j = i; j < n; j = j + 1$) possono essere eseguite al più n volte, così come le istruzioni all'interno del FOR($i = 0; i < n; i = i + 1$), per cui l'algoritmo MaxConsecutivaSomma2(a) eseguirà al più $d \times n \times n = n^2$ operazioni elementari, per qualche costante d.

Analisi dell'Algoritmo

```
MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
    SommaCorrente= 0
    FOR(j = i; j < n; j = j + 1){
        SommaCorrente=SommaCorrente +a[j]
        IF(SommaCorrente>SommaMassima){
          SommaMassima=SommaCorrente
         } } }
RETURN SommaMassima
```

Le istruzioni all'interno del FOR($j = i; j < n; j = j + 1$) possono essere eseguite al più n volte, così come le istruzioni all'interno del FOR($i = 0; i < n; i = i + 1$), per cui l'algoritmo MaxConsecutivaSomma2(a) eseguirà al più $d \times n \times n = n^2$ operazioni elementari, per qualche costante d.

In altri termini, la complessità dell'algoritmo MaxConsecutivaSomma2(a) su di un input composto da una sequenza di n numeri è $O(n^2)$.

Terzo possibile algoritmo

Applichiamo la tecnica Divide-et-Impera.

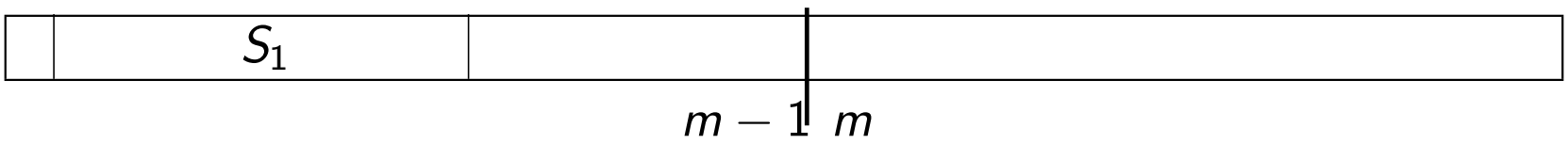
Terzo possibile algoritmo

Applichiamo la tecnica Divide-et-Impera.
Idea: Data la sequenza input di n numeri $a = a[0a[1] \dots a[n-1]$, calcoliamo $m = \lfloor n/2 \rfloor$.

Terzo possibile algoritmo

Applichiamo la tecnica Divide-et-Impera.
Idea: Data la sequenza input di n numeri $a = a[0a[1] \dots a[n-1]$, calcoliamo $m = \lfloor n/2 \rfloor$. La massima somma consecutiva (MSC) della sequenza $a = a[0]a[1] \cdots a[n-1]$ deve necessariamente essere una delle seguenti:

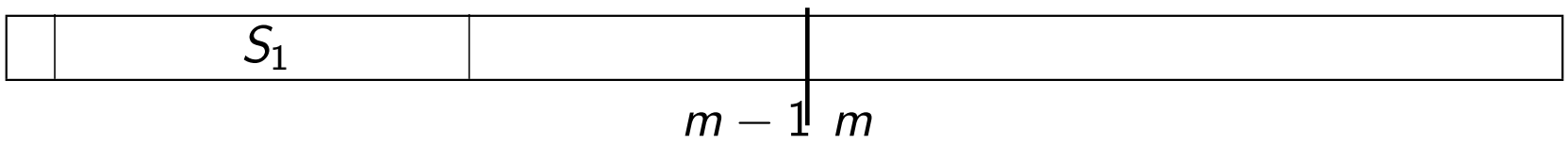
S_1 = la MSC della sottosequenza $a[0]a[1] \cdots a[m-1]$



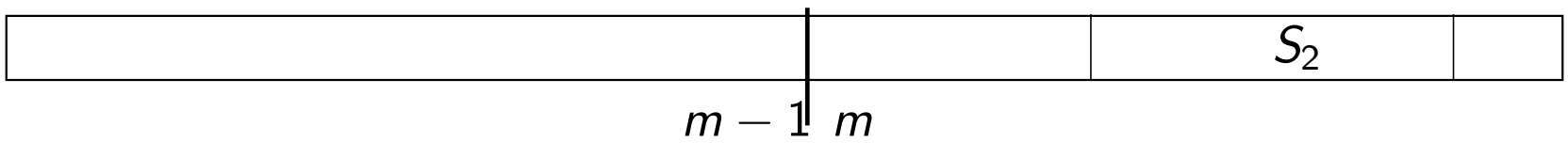
Terzo possibile algoritmo

Applichiamo la tecnica Divide-et-Impera.
Idea: Data la sequenza input di n numeri $a = a[0a[1] \dots a[n-1]$, calcoliamo $m = \lfloor n/2 \rfloor$. La massima somma consecutiva (MSC) della sequenza $a = a[0]a[1] \cdots a[n-1]$ deve necessariamente essere una delle seguenti:

S_1 = la MSC della sottosequenza $a[0]a[1] \cdots a[m-1]$



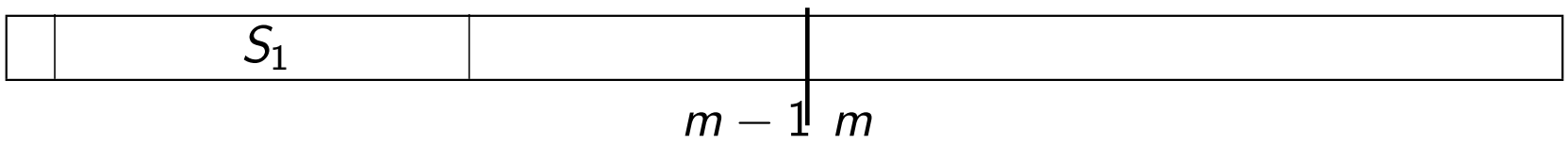
oppure S_2: la MSC della sottosequenza $a[m]a[m+1] \cdots a[n-1]$



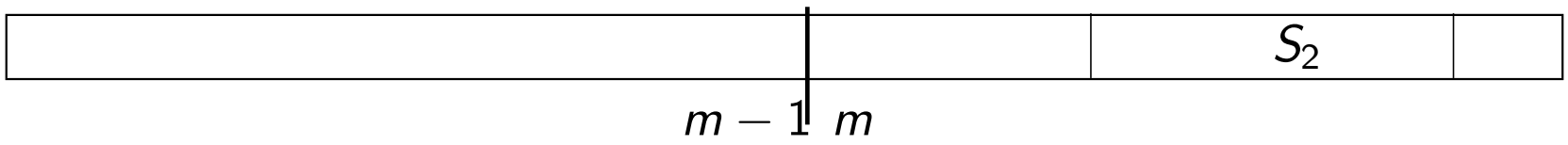
Terzo possibile algoritmo

Applichiamo la tecnica Divide-et-Impera.
Idea: Data la sequenza input di n numeri $a = a[0a[1] \dots a[n-1]$, calcoliamo $m = \lfloor n/2 \rfloor$. La massima somma consecutiva (MSC) della sequenza $a = a[0]a[1] \cdots a[n-1]$ deve necessariamente essere una delle seguenti:

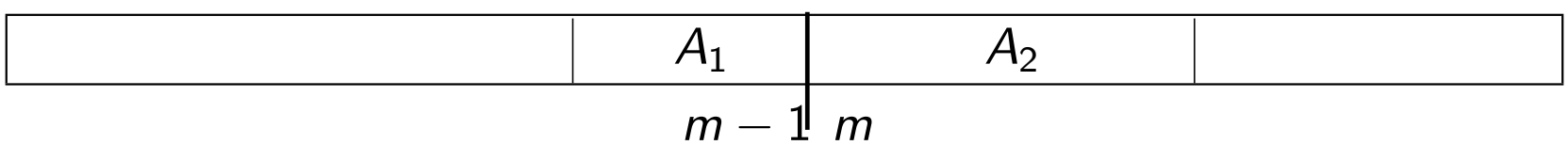
S_1 = la MSC della sottosequenza $a[0]a[1] \cdots a[m-1]$



oppure S_2: la MSC della sottosequenza $a[m]a[m+1] \cdots a[n-1]$



oppure è a "cavallo" di $a[m-1]$, ovvero la massima somma consecutiva di a è della forma $A = A_1 \cup A_2$, con



Esempio

Vediamo un esempio. Sia $a = a[0]a[1]\cdots a[15]$ dato da:

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

Esempio

Vediamo un esempio. Sia $a = a[0]a[1]\cdots a[15]$ dato da:

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

In questo caso $S_1 = [3, 6]$ con valore $3 + 6 = 9$, e $S_2 = [2, 6, 1]$ con valore $2 + 6 + 1 = 9$.

Esempio

Vediamo un esempio. Sia $a = a[0]a[1]\cdots a[15]$ dato da:

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

In questo caso $S_1 = [3, 6]$ con valore $3 + 6 = 9$, e $S_2 = [2, 6, 1]$ con valore $2 + 6 + 1 = 9$.

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

Ma abbiamo anche $A_1 = [3, 6, -1]$, $A_2 = [2, -4, 7]$, con $A = A_1 \cup A_2 = [3, 6, -1, 2, -4, 7]$, di valore totale $3 + 6 - 1 + 2 - 4 + 7 = 13$.

Dobbiamo quindi trovare S_1, S_2 e A e scegliere la migliore

Dobbiamo quindi trovare S_1, S_2 e A e scegliere la migliore

- il valore S_1 lo si trova determinando la MSC di $a[0]a[1]\cdots a[m-1]$, ovvero chiamando l'algoritmo **ricorsivamente** sulla parte sinistra $a[0]a[1]\cdots a[m-1]$ della sequenza $a[0]a[1]\cdots a[n-1]$

Dobbiamo quindi trovare S_1, S_2 e A e scegliere la migliore

- il valore S_1 lo si trova determinando la MSC di $a[0]a[1]\cdots a[m-1]$, ovvero chiamando l'algoritmo **ricorsivamente** sulla parte sinistra $a[0]a[1]\cdots a[m-1]$ della sequenza $a[0]a[1]\cdots a[n-1]$
- il valore S_2 lo si trova determinando la MSC di $a[m]a[m+1]\cdots a[n-1]$, ovvero chiamando l'algoritmo **ricorsivamente** sulla parte destra $a[m]a[m+1]\cdots a[n-1]$ della sequenza $a[0]a[1]\cdots a[n-1]$

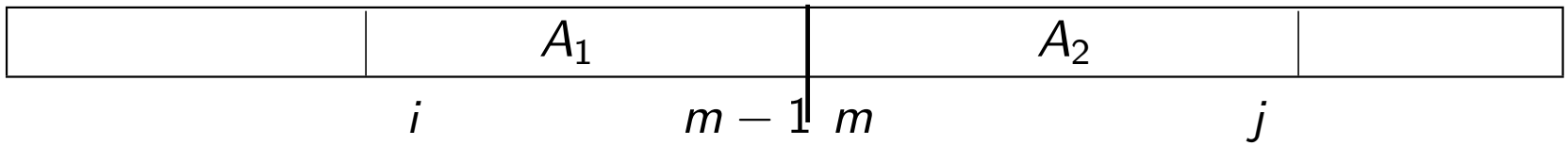
Dobbiamo quindi trovare S_1, S_2 e A e scegliere la migliore

- il valore S_1 lo si trova determinando la MSC di $a[0]a[1]\cdots a[m-1]$, ovvero chiamando l'algoritmo **ricorsivamente** sulla parte sinistra $a[0]a[1]\cdots a[m-1]$ della sequenza $a[0]a[1]\cdots a[n-1]$
- il valore S_2 lo si trova determinando la MSC di $a[m]a[m+1]\cdots a[n-1]$, ovvero chiamando l'algoritmo **ricorsivamente** sulla parte destra $a[m]a[m+1]\cdots a[n-1]$ della sequenza $a[0]a[1]\cdots a[n-1]$
- Come trovare il valore A lo vediamo nella prossima slide...

Dobbiamo quindi trovare S_1, S_2 e A e scegliere la migliore

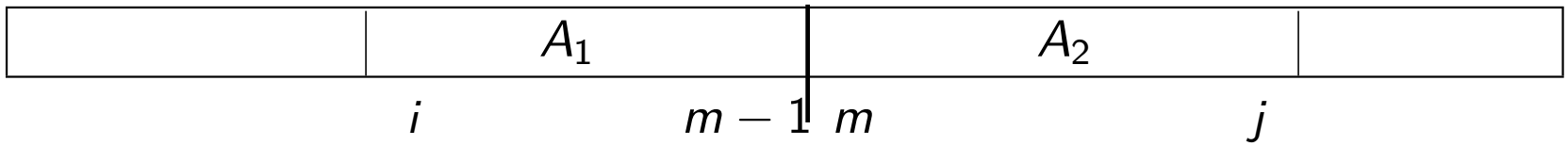
- il valore S_1 lo si trova determinando la MSC di $a[0]a[1]\cdots a[m-1]$, ovvero chiamando l'algoritmo **ricorsivamente** sulla parte sinistra $a[0]a[1]\cdots a[m-1]$ della sequenza $a[0]a[1]\cdots a[n-1]$
- il valore S_2 lo si trova determinando la MSC di $a[m]a[m+1]\cdots a[n-1]$, ovvero chiamando l'algoritmo **ricorsivamente** sulla parte destra $a[m]a[m+1]\cdots a[n-1]$ della sequenza $a[0]a[1]\cdots a[n-1]$
- Come trovare il valore A lo vediamo nella prossima slide...
- La MSC dell'intera sequenza $a[0]a[1]\cdots a[n-1]$ sará quindi quel valore tra S_1, S_2, ed A, che ha valore *massimo*.

Come trovare A:



- A_1 è la massima somma contigua della forma $a[i] \ldots a[m-1]$ per qualche valore di $i \in \{0, \ldots, m-1\}$ (m è *fissato* e *solo* i può variare):

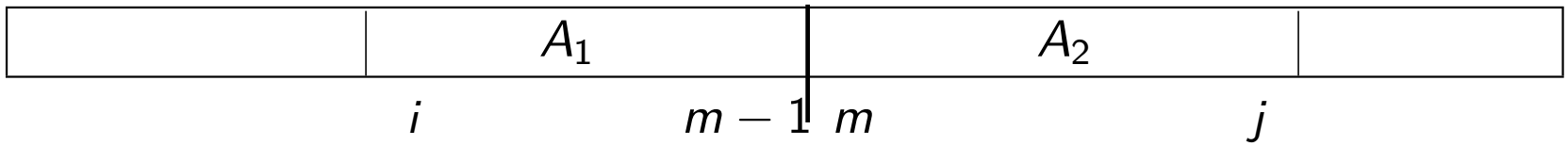
Come trovare A:



- A_1 è la massima somma contigua della forma $a[i] \ldots a[m-1]$ per qualche valore di $i \in \{0, \ldots, m-1\}$ (m è *fissato* e *solo* i può variare):

  ci sono *solo* $m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $i, 0 \leq i \leq m-1$.

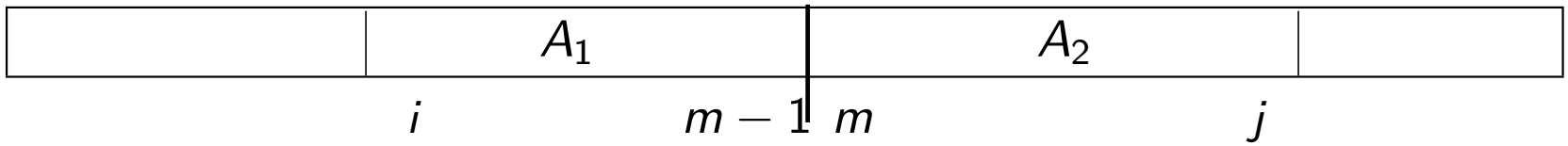
Come trovare A:



- A_1 è la massima somma contigua della forma $a[i] \ldots a[m-1]$ per qualche valore di $i \in \{0, \ldots, m-1\}$ (m è *fissato* e *solo* i può variare):

  ci sono *solo* $m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $i, 0 \leq i \leq m-1$. Pertanto la sequenza contigua A_1 di valore massimo puó essere trovata usando al piú $m \leq n$ operazioni, con un semplice ciclo `FOR`, con l'indice i del `FOR` che varia tra 0 e $m-1$.

Come trovare A:

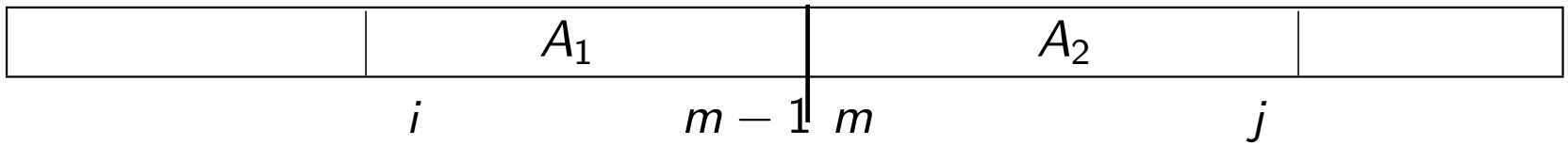


- A_1 è la massima somma contigua della forma $a[i] \ldots a[m-1]$ per qualche valore di $i \in \{0, \ldots, m-1\}$ (m è *fissato* e *solo* i può variare):

  ci sono *solo* $m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $i, 0 \leq i \leq m-1$. Pertanto la sequenza contigua A_1 di valore massimo puó essere trovata usando al piú $m \leq n$ operazioni, con un semplice ciclo `FOR`, con l'indice i del `FOR` che varia tra 0 e $m-1$.

- Analogamente, A_2 é della forma $a[m] \ldots a[j]$. per qualche valore di $j \in \{m, \ldots n-1\}$ (m è *fissato* e *solo* j può variare):

Come trovare A:



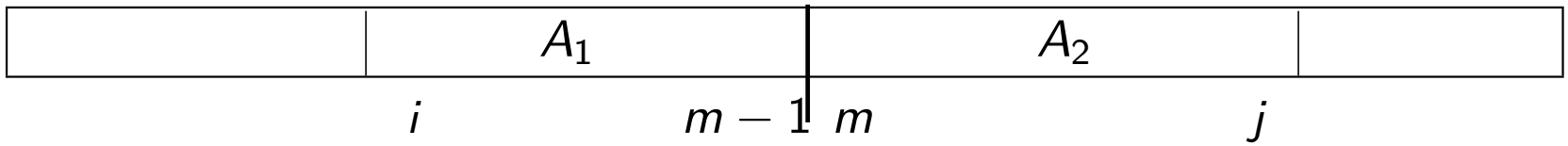
- A_1 è la massima somma contigua della forma $a[i] \ldots a[m-1]$ per qualche valore di $i \in \{0, \ldots, m-1\}$ (m è *fissato* e *solo* i può variare):

  ci sono *solo* $m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $i, 0 \leq i \leq m-1$. Pertanto la sequenza contigua A_1 di valore massimo puó essere trovata usando al piú $m \leq n$ operazioni, con un semplice ciclo `FOR`, con l'indice i del `FOR` che varia tra 0 e $m-1$.

- Analogamente, A_2 é della forma $a[m] \ldots a[j]$. per qualche valore di $j \in \{m, \ldots n-1\}$ (m è *fissato* e *solo* j può variare):

  ci sono solo $n - m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $j, m \leq j \leq n-1$.

Come trovare A:



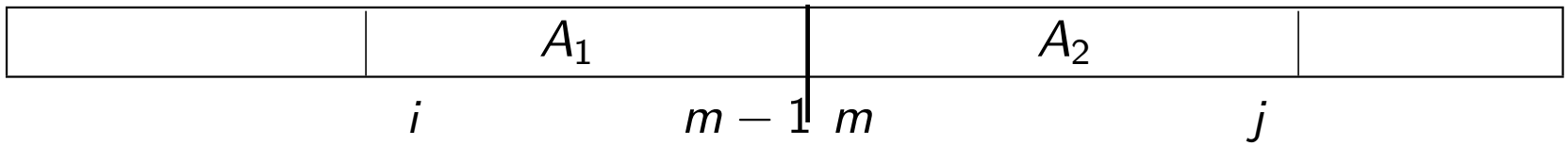
- A_1 è la massima somma contigua della forma $a[i] \ldots a[m-1]$ per qualche valore di $i \in \{0, \ldots, m-1\}$ (m è *fissato* e *solo* i può variare):

  ci sono *solo* $m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $i, 0 \leq i \leq m-1$. Pertanto la sequenza contigua A_1 di valore massimo puó essere trovata usando al piú $m \leq n$ operazioni, con un semplice ciclo `FOR`, con l'indice i del `FOR` che varia tra 0 e $m-1$.

- Analogamente, A_2 é della forma $a[m] \ldots a[j]$. per qualche valore di $j \in \{m, \ldots n-1\}$ (m è *fissato* e *solo* j può variare):

  ci sono solo $n - m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $j, m \leq j \leq n-1$. Pertanto la sequenza contigua A_2 di valore massimo puó essere trovata in $n - m \leq n$ operazioni, sempre con un semplice ciclo `FOR`, con l'indice j del `FOR` che varia tra m ed $n-1$.

Come trovare A:



- A_1 è la massima somma contigua della forma $a[i] \ldots a[m-1]$ per qualche valore di $i \in \{0, \ldots, m-1\}$ (m è *fissato* e *solo* i può variare):

  ci sono *solo* $m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $i, 0 \leq i \leq m-1$. Pertanto la sequenza contigua A_1 di valore massimo puó essere trovata usando al piú $m \leq n$ operazioni, con un semplice ciclo `FOR`, con l'indice i del `FOR` che varia tra 0 e $m-1$.

- Analogamente, A_2 é della forma $a[m] \ldots a[j]$. per qualche valore di $j \in \{m, \ldots n-1\}$ (m è *fissato* e *solo* j può variare):

  ci sono solo $n - m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $j, m \leq j \leq n-1$. Pertanto la sequenza contigua A_2 di valore massimo puó essere trovata in $n - m \leq n$ operazioni, sempre con un semplice ciclo `FOR`, con l'indice j del `FOR` che varia tra m ed $n-1$. Riassumendo

$A = A_1 \cup A_2$ può essere trovato in $O(n)$ operazioni

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

`MaxConsecutivaSomma3`(a, i, j)

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i]
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

`MaxConsecutivaSomma3`(a, i, j)
1. `IF` $i == j$ `RETURN` $a[i]$ `ELSE`
2. $\quad S_1 =$ `MaxConsecutivaSomma3`$(a, i, \lfloor (i+j)/2 \rfloor)$

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

$\texttt{MaxConsecutivaSomma3}(a, i, j)$
1. $\texttt{IF } i == j \texttt{ RETURN } a[i] \texttt{ ELSE}$
2. $\quad S_1 = \texttt{MaxConsecutivaSomma3}(a, i, \lfloor (i+j)/2 \rfloor)$
3. $\quad S_2 = \texttt{MaxConsecutivaSomma3}(a, \lfloor (i+j)/2 \rfloor + 1, j)$

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

`MaxConsecutivaSomma3`(a, i, j)

1. `IF` $i == j$ `RETURN` $a[i]$ `ELSE`
2. $\quad S_1 =$ `MaxConsecutivaSomma3`$(a, i, \lfloor (i+j)/2 \rfloor)$
3. $\quad S_2 =$ `MaxConsecutivaSomma3`$(a, \lfloor (i+j)/2 \rfloor + 1, j)$
4. $\quad A_1 = 0, B_1 = a[\lfloor (i+j)/2 \rfloor]$

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

$\texttt{MaxConsecutivaSomma3}(a, i, j)$
1. $\texttt{IF } i == j \texttt{ RETURN } a[i] \texttt{ ELSE}$
2. $\quad S_1 = \texttt{MaxConsecutivaSomma3}(a, i, \lfloor (i+j)/2 \rfloor)$
3. $\quad S_2 = \texttt{MaxConsecutivaSomma3}(a, \lfloor (i+j)/2 \rfloor + 1, j)$
4. $\quad A_1 = 0, B_1 = a[\lfloor (i+j)/2 \rfloor]$
5. $\texttt{FOR}(s = \lfloor (i+j)/2 \rfloor; s > i-1; s = s-1)\{$

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

$\texttt{MaxConsecutivaSomma3}(a, i, j)$
1. $\texttt{IF } i == j \texttt{ RETURN } a[i] \texttt{ ELSE}$
2. $\quad S_1 = \texttt{MaxConsecutivaSomma3}(a, i, \lfloor (i+j)/2 \rfloor)$
3. $\quad S_2 = \texttt{MaxConsecutivaSomma3}(a, \lfloor (i+j)/2 \rfloor + 1, j)$
4. $\quad A_1 = 0, B_1 = a[\lfloor (i+j)/2 \rfloor]$
5. $\texttt{FOR}(s = \lfloor (i+j)/2 \rfloor; s > i-1; s = s-1)\{$
6. $\qquad\qquad A_1 = A_1 + a[s]$

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.             A1 = A1 + a[s]
7.             IF(A1 > B1) {
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                 B1 = A1
               }
           }
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.             A1 = A1 + a[s]
7.             IF(A1 > B1) {
8.                     B1 = A1
                   }
               }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.             A1 = A1 + a[s]
7.             IF(A1 > B1) {
8.                     B1 = A1
                   }
               }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.        FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                 B1 = A1
                 }
             }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.        FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A2 = A2 + a[t]
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                 B1 = A1
                 }
             }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.        FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.           A2 = A2 + a[t]
12.           IF(A2 > B2) {
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                 B1 = A1
               }
           }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.        FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A2 = A2 + a[t]
12.          IF(A2 > B2) {
13.                B2 = A2
               }
           }
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                 B1 = A1
                 }
             }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.        FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A2 = A2 + a[t]
12.          IF(A2 > B2) {
13.                B2 = A2
                 }
             }
14.    S3 = B1 + B2
```

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                 B1 = A1
               }
           }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.        FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A2 = A2 + a[t]
12.          IF(A2 > B2) {
13.                B2 = A2
               }
           }
14.    S3 = B1 + B2
15.    RETURN MAX(S1, S2, S3)
```

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.     S_1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S_2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A_1 = 0, B_1 = a[⌊(i + j)/2⌋]
5. FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.             A_1 = A_1 + a[s]
7.             IF(A_1 > B_1) {
8.                 B_1 = A_1
                 }
               }
9.     A_2 = 0, B_2 = a[⌊(i + j)/2⌋ + 1]
10.        FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.            A_2 = A_2 + a[t]
12.            IF(A_2 > B_2) {
13.                B_2 = A_2
                 }
               }
14.    S_3 = B_1 + B_2
15.    RETURN MAX(S_1, S_2, S_3)
```

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1.  IF i == j RETURN a[i] ELSE
2.     S_1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S_2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A_1 = 0, B_1 = a[⌊(i + j)/2⌋]
5.  FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A_1 = A_1 + a[s]
7.           IF(A_1 > B_1) {
8.                B_1 = A_1
               }
           }
9.     A_2 = 0, B_2 = a[⌊(i + j)/2⌋ + 1]
10.       FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A_2 = A_2 + a[t]
12.          IF(A_2 > B_2) {
13.               B_2 = A_2
               }
           }
14.    S_3 = B_1 + B_2
15.    RETURN MAX(S_1, S_2, S_3)
```

Sia $T(n)$ il numero di operazioni di MaxConsecutivaSomma3$(a, 0, n-1)$,

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1.  IF i == j RETURN a[i] ELSE
2.     S_1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S_2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A_1 = 0, B_1 = a[⌊(i + j)/2⌋]
5.  FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A_1 = A_1 + a[s]
7.           IF(A_1 > B_1) {
8.                B_1 = A_1
               }
           }
9.     A_2 = 0, B_2 = a[⌊(i + j)/2⌋ + 1]
10.       FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A_2 = A_2 + a[t]
12.          IF(A_2 > B_2) {
13.               B_2 = A_2
               }
           }
14.    S_3 = B_1 + B_2
15.    RETURN MAX(S_1, S_2, S_3)
```

Sia $T(n)$ il numero di operazioni di MaxConsecutivaSomma3$(a, 0, n-1)$, abbiamo:

- l'istruzione 1. richiede tempo $O(1)$.

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1.  IF i == j RETURN a[i] ELSE
2.     S_1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S_2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A_1 = 0, B_1 = a[⌊(i + j)/2⌋]
5.  FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A_1 = A_1 + a[s]
7.           IF(A_1 > B_1) {
8.                B_1 = A_1
               }
           }
9.     A_2 = 0, B_2 = a[⌊(i + j)/2⌋ + 1]
10.       FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A_2 = A_2 + a[t]
12.          IF(A_2 > B_2) {
13.               B_2 = A_2
               }
           }
14.    S_3 = B_1 + B_2
15.    RETURN MAX(S_1, S_2, S_3)
```

Sia $T(n)$ il numero di operazioni di MaxConsecutivaSomma3$(a, 0, n-1)$, abbiamo:

- l'istruzione 1. richiede tempo $O(1)$.
- Le istruzioni 2. e 3. richiedono tempo $T(n/2)$, ciascheduna,

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1.  IF i == j RETURN a[i] ELSE
2.     S_1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S_2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A_1 = 0, B_1 = a[⌊(i + j)/2⌋]
5.  FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A_1 = A_1 + a[s]
7.           IF(A_1 > B_1) {
8.                B_1 = A_1
                }
            }
9.     A_2 = 0, B_2 = a[⌊(i + j)/2⌋ + 1]
10.       FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A_2 = A_2 + a[t]
12.          IF(A_2 > B_2) {
13.               B_2 = A_2
                }
            }
14.    S_3 = B_1 + B_2
15.    RETURN MAX(S_1, S_2, S_3)
```

Sia $T(n)$ il numero di operazioni di `MaxConsecutivaSomma3`$(a, 0, n-1)$, abbiamo:

- l'istruzione 1. richiede tempo $O(1)$.
- Le istruzioni 2. e 3. richiedono tempo $T(n/2)$, ciascheduna, in quanto in ciascuna di esse eseguiamo il nostro algoritmo *MSC* su di una sequenza lunga la *metà* di quella di partenza.

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1.  IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5.  FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                B1 = A1
                }
            }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.       FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A2 = A2 + a[t]
12.          IF(A2 > B2) {
13.               B2 = A2
                }
            }
14.    S3 = B1 + B2
15.    RETURN MAX(S1, S2, S3)
```

Sia $T(n)$ il numero di operazioni di $\texttt{MaxConsecutivaSomma3}(a, 0, n-1)$, abbiamo:

- l'istruzione 1. richiede tempo $O(1)$.
- Le istruzioni 2. e 3. richiedono tempo $T(n/2)$, ciascheduna, in quanto in ciascuna di esse eseguiamo il nostro algoritmo *MSC* su di una sequenza lunga la *metà* di quella di partenza.
- Le istruzioni 4-15 richiedono in totale tempo $O(n)$.

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1.  IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5.  FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                B1 = A1
               }
           }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.       FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A2 = A2 + a[t]
12.          IF(A2 > B2) {
13.               B2 = A2
               }
           }
14.    S3 = B1 + B2
15.    RETURN MAX(S1, S2, S3)
```

Sia $T(n)$ il numero di operazioni di MaxConsecutivaSomma3$(a, 0, n-1)$, abbiamo:

- l'istruzione 1. richiede tempo $O(1)$.
- Le istruzioni 2. e 3. richiedono tempo $T(n/2)$, ciascheduna, in quanto in ciascuna di esse eseguiamo il nostro algoritmo *MSC* su di una sequenza lunga la *metà* di quella di partenza.
- Le istruzioni 4-15 richiedono in totale tempo $O(n)$.

In totale

$$\boxed{T(n) = 2T(n/2) + O(n)}$$

Analisi dell'algoritmo

```
MaxConsecutivaSomma3(a, i, j)
1.  IF i == j RETURN a[i] ELSE
2.     S1 = MaxConsecutivaSomma3(a, i, ⌊(i + j)/2⌋)
3.     S2 = MaxConsecutivaSomma3(a, ⌊(i + j)/2⌋ + 1, j)
4.     A1 = 0, B1 = a[⌊(i + j)/2⌋]
5.  FOR(s = ⌊(i + j)/2⌋; s > i − 1; s = s − 1){
6.           A1 = A1 + a[s]
7.           IF(A1 > B1) {
8.                B1 = A1
               }
           }
9.     A2 = 0, B2 = a[⌊(i + j)/2⌋ + 1]
10.       FOR(t = ⌊(i + j)/2⌋ + 1; t < j + 1; t = t + 1){
11.          A2 = A2 + a[t]
12.          IF(A2 > B2) {
13.               B2 = A2
               }
           }
14.    S3 = B1 + B2
15.    RETURN MAX(S1, S2, S3)
```

Sia $T(n)$ il numero di operazioni di $\texttt{MaxConsecutivaSomma3}(a, 0, n-1)$, abbiamo:

- l'istruzione 1. richiede tempo $O(1)$.
- Le istruzioni 2. e 3. richiedono tempo $T(n/2)$, ciascheduna, in quanto in ciascuna di esse eseguiamo il nostro algoritmo *MSC* su di una sequenza lunga la *metà* di quella di partenza.
- Le istruzioni 4-15 richiedono in totale tempo $O(n)$.

In totale

$$\boxed{T(n) = 2T(n/2) + O(n)}$$

$$\Downarrow$$

$$\boxed{T(n) = O(n \log n)}$$

Riassumiamo...

Dato il problema computazionale:
Input: sequenza di numeri $a = a[0]a[1]\cdots a[n-1]$

Output: $V(a) = \max_{0 \le i \le j \le n-1} V(i,j) = \max_{0 \le i \le j \le n-1} \sum_{k=i}^{j} a[k]$

Riassumiamo...

Dato il problema computazionale:
Input: sequenza di numeri $a = a[0]a[1]\cdots a[n-1]$

Output: $V(a) = \max_{0 \le i \le j \le n-1} V(i,j) = \max_{0 \le i \le j \le n-1} \sum_{k=i}^{j} a[k]$

abbiamo progettato 3 *differenti* algoritmi per la sua soluzione:

Riassumiamo...

Dato il problema computazionale:
Input: sequenza di numeri $a = a[0]a[1]\cdots a[n-1]$

Output: $V(a) = \max_{0\le i\le j\le n-1} V(i,j) = \max_{0\le i\le j\le n-1} \sum_{k=i}^{j} a[k]$

abbiamo progettato 3 *differenti* algoritmi per la sua soluzione:

1. `MaxConsecutivaSomma1`(a) richiede tempo $O(n^3)$

Riassumiamo...

Dato il problema computazionale:
Input: sequenza di numeri $a = a[0]a[1]\cdots a[n-1]$

Output: $V(a) = \max_{0\le i\le j\le n-1} V(i,j) = \max_{0\le i\le j\le n-1} \sum_{k=i}^{j} a[k]$

abbiamo progettato 3 *differenti* algoritmi per la sua soluzione:

1. `MaxConsecutivaSomma1`(a) richiede tempo $O(n^3)$
2. `MaxConsecutivaSomma2`(a) richiede tempo $O(n^2)$

Riassumiamo...

Dato il problema computazionale:
Input: sequenza di numeri $a = a[0]a[1]\cdots a[n-1]$

Output: $V(a) = \max_{0\le i\le j\le n-1} V(i,j) = \max_{0\le i\le j\le n-1} \sum_{k=i}^{j} a[k]$

abbiamo progettato 3 *differenti* algoritmi per la sua soluzione:

1. `MaxConsecutivaSomma1`(a) richiede tempo $O(n^3)$
2. `MaxConsecutivaSomma2`(a) richiede tempo $O(n^2)$
3. `MaxConsecutivaSomma3`(a) richiede tempo $O(n \log n)$

Riassumiamo...

Dato il problema computazionale:
Input: sequenza di numeri $a = a[0]a[1]\cdots a[n-1]$

Output: $V(a) = \max_{0\le i\le j\le n-1} V(i,j) = \max_{0\le i\le j\le n-1} \sum_{k=i}^{j} a[k]$

abbiamo progettato 3 *differenti* algoritmi per la sua soluzione:

1. `MaxConsecutivaSomma1`(a) richiede tempo $O(n^3)$
2. `MaxConsecutivaSomma2`(a) richiede tempo $O(n^2)$
3. `MaxConsecutivaSomma3`(a) richiede tempo $O(n \log n)$

Possiamo fare **ancora** meglio?

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] =$$

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare.

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Se $M(i) = a[i]$, non c'è niente da provare.

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Se $M(i) = a[i]$, non c'è niente da provare. Se $M(i) = x + a[i]$,

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Se $M(i) = a[i]$, non c'è niente da provare. Se $M(i) = x + a[i]$, allora la somma x ha come ultimo termine il valore $a[i-1]$,

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Se $M(i) = a[i]$, non c'è niente da provare. Se $M(i) = x + a[i]$, allora la somma x ha come ultimo termine il valore $a[i-1]$, e *necessariamente* vale che $x = M(i-1)$.

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Se $M(i) = a[i]$, non c'è niente da provare. Se $M(i) = x + a[i]$, allora la somma x ha come ultimo termine il valore $a[i-1]$, e *necessariamente* vale che $x = M(i-1)$. Se non fosse così, ovvero se $x < y = M(i-1)$,

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Se $M(i) = a[i]$, non c'è niente da provare. Se $M(i) = x + a[i]$, allora la somma x ha come ultimo termine il valore $a[i-1]$, e *necessariamente* vale che $x = M(i-1)$. Se non fosse così, ovvero se $x < y = M(i-1)$, allora esisterebbe *un'altra somma* che termina con $a[i]$, di valore $y + a[i] > x + a[i] = M(i)$,

Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la quantità:

$$M(i) = \text{il valore della massima somma consecutiva che ha come } \textit{ultimo} \text{ termine il valore } a[i].$$

Ovviamente vale che

$$V(a) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^{j} a[k] = \max\{M(0), M(1), \ldots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare. Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \tag{2}$$

Se $M(i) = a[i]$, non c'è niente da provare. Se $M(i) = x + a[i]$, allora la somma x ha come ultimo termine il valore $a[i-1]$, e *necessariamente* vale che $x = M(i-1)$. Se non fosse così, ovvero se $x < y = M(i-1)$, allora esisterebbe *un'altra somma* che termina con $a[i]$, di valore $y + a[i] > x + a[i] = M(i)$, contro l'ipotesi che $M(i)$ è il valore della *massima* somma che ha come ultimo termine $a[i]$.

Scriviamo quindi un algoritmo che calcola $V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
```

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
```

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
```

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
       SommaMassimaFAQ=SommaMassimaFAQ+a[i]
```

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
       SommaMassimaFAQ=SommaMassimaFAQ+a[i]
       ELSE SommaMassimaFAQ= a[i]
       }
```

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
       SommaMassimaFAQ=SommaMassimaFAQ+a[i]
       ELSE SommaMassimaFAQ= a[i]
       }
    IF(SommaMassimaFAQ>SommaMassima) {
```

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
       SommaMassimaFAQ=SommaMassimaFAQ+a[i]
       ELSE SommaMassimaFAQ= a[i]
       }
    IF(SommaMassimaFAQ>SommaMassima) {
       SommaMassima=SommaMassimaFAQ
       }
}
```

Scriviamo quindi un algoritmo che calcola
$V(a) = \max\{M(0), M(1), \ldots, M(n-1)\}$, per un'arbitraria sequenza $a = a[0]a[1]\ldots a[n-1]$ di n numeri, utilizzando l'identità

$$M(i) = \max\{M(i-1) + a[i], a[i]\}$$

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
       SommaMassimaFAQ=SommaMassimaFAQ+a[i]
       ELSE SommaMassimaFAQ= a[i]
       }
    IF(SommaMassimaFAQ>SommaMassima) {
       SommaMassima=SommaMassimaFAQ
       }
}
return SommaMassima
```

Analisi

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
        SommaMassimaFAQ=SommaMassimaFAQ+a[i]
        ELSE SommaMassimaFAQ= a[i]
        }
    IF(SommaMassimaFAQ>SommaMassima) {
        SommaMassima=SommaMassimaFAQ
        }
}
return SommaMassima
```

Analisi

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
    IF(SommaMassimaFAQ+a[i] > a[i]){
        SommaMassimaFAQ=SommaMassimaFAQ+a[i]
        ELSE SommaMassimaFAQ= a[i]
        }
    IF(SommaMassimaFAQ>SommaMassima) {
        SommaMassima=SommaMassimaFAQ
        }
}
return SommaMassima
```

Dopo ogni esecuzione del ciclo FOR($i = 1; i < n; i = i + 1$), la variabile SommaMassimaFAQ contiene il valore $M(i)$.

Analisi

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
     IF(SommaMassimaFAQ+a[i] > a[i]){
          SommaMassimaFAQ=SommaMassimaFAQ+a[i]
          ELSE SommaMassimaFAQ= a[i]
          }
     IF(SommaMassimaFAQ>SommaMassima) {
          SommaMassima=SommaMassimaFAQ
          }
}
return SommaMassima
```

Dopo ogni esecuzione del ciclo FOR($i = 1; i < n; i = i + 1$), la variabile SommaMassimaFAQ contiene il valore $M(i)$.

Visto che l'algoritmo consta di *un solo* ciclo FOR, esso eseguirà al più cn operazioni elementari, per qualche costante c.

Analisi

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
     IF(SommaMassimaFAQ+a[i] > a[i]){
          SommaMassimaFAQ=SommaMassimaFAQ+a[i]
          ELSE SommaMassimaFAQ= a[i]
          }
     IF(SommaMassimaFAQ>SommaMassima) {
          SommaMassima=SommaMassimaFAQ
          }
}
return SommaMassima
```

Dopo ogni esecuzione del ciclo FOR($i = 1; i < n; i = i + 1$), la variabile SommaMassimaFAQ contiene il valore $M(i)$.

Visto che l'algoritmo consta di *un solo* ciclo FOR, esso eseguirà al più cn operazioni elementari, per qualche costante c.

In altri termini, la complessità di MaxConsecutivaSomma4(a) su input a di dimensione n è $O(n)$.

Ne è valsa le pena?

Ne è valsa le pena?

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo,

Ne è valsa le pena?

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo, per input di dimensione $10^3, 10^4, 10^5, 10^6, 10^8$.

Ne è valsa le pena?

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo, per input di dimensione $10^3, 10^4, 10^5, 10^6, 10^8$.
Supponendo per semplicità che le costanti all'interno della notazione O siano tutte pari ad 1, avremmo i risultati seguenti:

	n	$n \log n$	n^2	n^3
10^3	0.000001s	0.00000996578s	0.001s	1s

Ne è valsa le pena?

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo, per input di dimensione $10^3, 10^4, 10^5, 10^6, 10^8$.
Supponendo per semplicità che le costanti all'interno della notazione O siano tutte pari ad 1, avremmo i risultati seguenti:

	n	$n \log n$	n^2	n^3
10^3	0.000001s	0.00000996578s	0.001s	1s
10^4	0.00001s	0.00013287712s	0.1s	$\approx$16.6m

Ne è valsa le pena?

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo, per input di dimensione $10^3, 10^4, 10^5, 10^6, 10^8$.
Supponendo per semplicità che le costanti all'interno della notazione O siano tutte pari ad 1, avremmo i risultati seguenti:

	n	$n \log n$	n^2	n^3
10^3	0.000001s	0.00000996578s	0.001s	1s
10^4	0.00001s	0.00013287712s	0.1s	$\approx$16.6m
10^5	0.0001s	0.00166096404s	10s	11.5giorni

Ne è valsa le pena?

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo, per input di dimensione $10^3, 10^4, 10^5, 10^6, 10^8$.
Supponendo per semplicità che le costanti all'interno della notazione O siano tutte pari ad 1, avremmo i risultati seguenti:

	n	$n \log n$	n^2	n^3
10^3	0.000001s	0.00000996578s	0.001s	1s
10^4	0.00001s	0.00013287712s	0.1s	$\approx$16.6m
10^5	0.0001s	0.00166096404s	10s	11.5giorni
10^6	0.001s	0.01993156856s	$\approx$17m	$\approx$31 anni

Ne è valsa le pena?

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo, per input di dimensione $10^3, 10^4, 10^5, 10^6, 10^8$.
Supponendo per semplicità che le costanti all'interno della notazione O siano tutte pari ad 1, avremmo i risultati seguenti:

	n	$n \log n$	n^2	n^3
10^3	0.000001s	0.00000996578s	0.001s	1s
10^4	0.00001s	0.00013287712s	0.1s	$\approx$16.6m
10^5	0.0001s	0.00166096404s	10s	11.5giorni
10^6	0.001s	0.01993156856s	$\approx$17m	$\approx$31 anni
10^8	0.1s	2.65754247591s	$\approx$6 mesi	troppo...

Morale della Lezione

Morale della Lezione

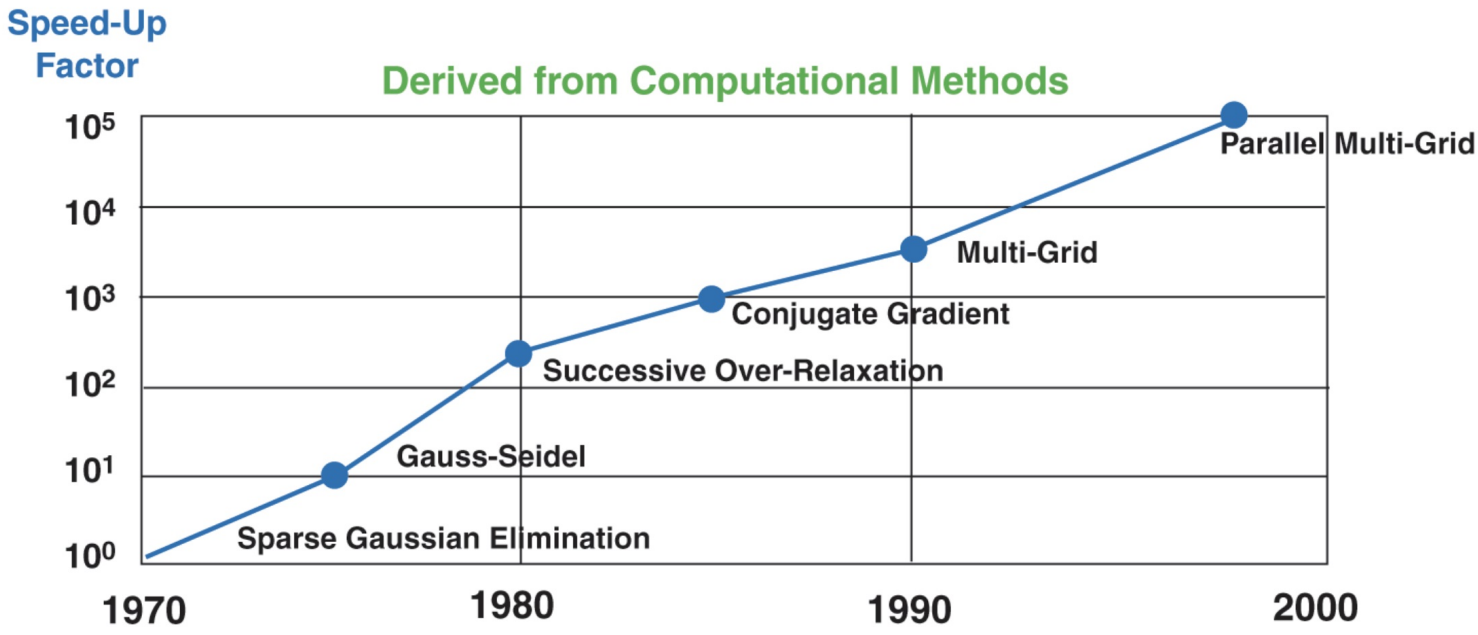
- Ne è valsa la pena!

Morale della Lezione

- Ne è valsa la pena!
- Uno stesso problema algoritmico può essere risolto attraverso varie e differenti tecniche, le quali produrranno vari e differenti algoritmi per la risoluzione dello stesso problema;

Morale della Lezione

- Ne è valsa la pena!
- Uno stesso problema algoritmico può essere risolto attraverso varie e differenti tecniche, le quali produrranno vari e differenti algoritmi per la risoluzione dello stesso problema;
- É importante avere algoritmi efficienti.



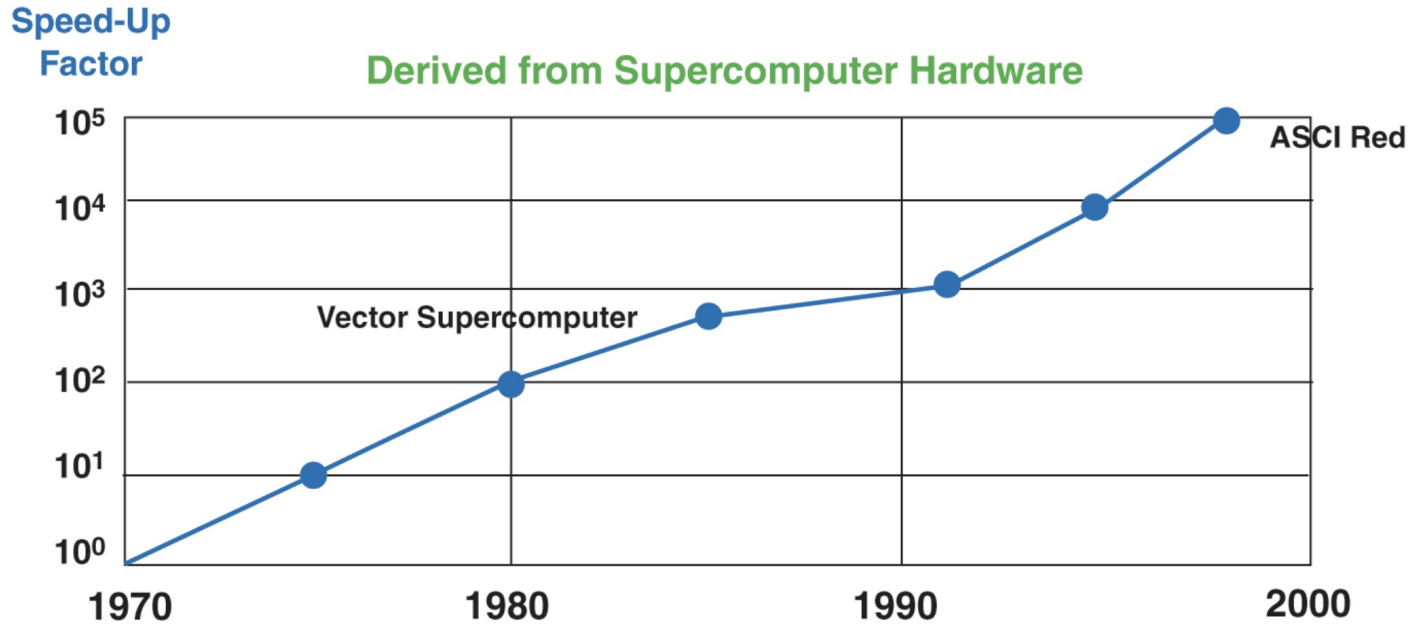


Fig. 2 *Comparison of the contributions of mathematical algorithms and computer hardware.*

In generale, si è visto che algoritmi **moderni**, eseguiti su hardware di 15 anni fà sono **più veloci** di algoritmi di 15 anni fà eseguiti su hardware **moderni**.

In generale, si è visto che algoritmi **moderni**, eseguiti su hardware di 15 anni fà sono **più veloci** di algoritmi di 15 anni fà eseguiti su hardware **moderni**.

A Giovedì...