

## 2.1 Algoritmo della somma massimale di un sottovettore

Date le nostre nuove conoscenze possiamo calcolare con precisione la complessità delle varie versioni degli algoritmi proposti per la soluzione al problema della somma massimale di un sottovettore.

### Complessità della prima versione

---

```
int maxsum1(int[] A, int n) {
    int maxSoFar = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int sum = 0;
            for (int k = i; k <= j; k++) {
                sum = sum + A[k];
            }
            maxSoFar = max(maxSoFar, sum);
        }
    }
    return maxSoFar;
}
```

---

La complessità dell'algoritmo può essere approssimata come segue (contando il numero di esecuzioni della riga più interna):

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1)$$

Vogliamo provare che  $T(n) = \mathcal{O}(n^3)$ .

*Dimostrazione. limite superiore:*  $\exists c_2 > 0, \exists m \geq 0 : T(n) \leq c_2 n^3, \forall n \geq m$ .

$$\begin{aligned} T(n) &= \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1) \\ &\leq \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} n \\ &\leq \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n \\ &= \sum_{i=0}^{n-1} n^2 \\ &= n^3 \leq c_2 n^3 \end{aligned}$$

spiegazione  
spiegazione  
spiegazione  
spiegazione  
spiegazione

Questa disequazione è vera per  $n \geq m = 0$  and  $c_2 \geq 1$ .  $\square$

Vogliamo provare che  $T(n) = \Omega(n^3)$ .

*Dimostrazione.* **limite inferiore:**  $\exists c_1 > 0, \exists m \geq 0 : T(n) \leq c_1 n^3, \forall n \geq m$ .

$$\begin{aligned}
T(n) &= \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1) \\
&\geq \sum_{i=0}^{n/2} \sum_{j=i}^{n+n/2-1} (j - i + 1) \\
&= \sum_{i=0}^{n/2} \sum_{j=i}^{n+n/2-1} n/2 \\
&= \sum_{i=0}^{n/2} n^2/4 \geq n^3/8 \geq c_1 n^3
\end{aligned}$$

*spiegazione*

*spiegazione*

Questa disequazione è vera per  $n \geq m = 0$  and  $c_1 \geq 8$ . □

### Complessità della seconda versione

---

```

int maxsum2(int[] A, int n) {
    int maxSoFar = 0;
    for (int i=0; i < n; i++) {
        int sum = 0;
        for (int j=i; j < n; j++) {
            sum = sum + A[j];
            maxSoFar = max(maxSoFar, sum);
        }
    }
    return maxSoFar;
}

```

---

La complessità di questo algoritmo può essere approssimata come segue (stiamo contando il numero di passi nel ciclo più interno):

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

Vogliamo provare che  $T(n) = \Theta(n^2)$

*Dimostrazione.*

$$\begin{aligned}
T(n) &= \sum_{i=0}^{n-1} n - i \\
&= \sum_{i=1}^n i \\
&= \frac{n(n+1)}{2} = \Theta(n^2)
\end{aligned}$$

*spiegazione*

*spiegazione*

Questo non richiede ulteriori spiegazioni. □

### Complessità della terza versione

---

```

int maxsum_rec(int[] A, int i, int j) {
    if (i == j)
        return max(0, A[i]);

    int m = (i + j) / 2;
    int maxs = maxsum_rec(A, i, m);
    int maxd = maxsum_rec(A, m + 1, j);

```

---

```

int maxss = 0;
int sum = 0;

for (int k = m; k >= i; k--) {
    sum = sum + A[k];
    maxss = max(maxss, sum);
}

int maxdd = 0;
sum = 0;
for (int k = m + 1; k <= j; k++) {
    sum = sum + A[k];
    maxdd = max(maxdd, sum);
}

return max(max(maxs, maxd), maxss + maxdd);
}

```

---

Per questo, definiamo la equazione di ricorrenza:

$$T(n) = 2T(n/2) + n$$

Utilizzando il teorema, possiamo vedere che  $\alpha = \log_2 2 = 1$  e  $\beta = 1$ , quindi  $T(n) = \Theta(n \log n)$ .

### Complessità della quarta versione

```

int maxsum4(int A[], int n) {
    int maxSoFar = 0;
    int maxHere = 0;
    for (int i = 0; i < n; i++) {
        maxHere = max(maxHere + A[i], 0);
        maxSoFar = max(maxSoFar, maxHere);
    }

    return maxSoFar;
}

```

---

È facile vedere che la complessità di questa versione è  $\Theta(n)$ .