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

- 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

- 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 $j-1$. 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

- 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 somma degli elementi della sottosequenza di somma massima tra quelle che terminano in j. Si ha $s_{j+1} = \max\{s_j + a[j+1], a[j+1]\}$. Se s_j è noto , questo valore si calcola in tempo costante per ogni j. Possiamo calcolare i valori $s_0, s_1, \ldots, s_{n-1}$ in tempo O(n) in uno dei seguenti modi:
  - in modo iterativo partendo da $s_0 = A[0]$ e memorizzando via via i valori computati in un array s
  - in modo ricorsivo: l'algoritmo prende in input A e un intero $k \geq 0$ e
    - se $k = 0$, pone $s[0] = A[0]$ restituendolo in ountput;
    - se $k > 0$, invoca ricorsivamente se stesso su A e $k - 1$ e, una volta ottenuto s_{k-1} dalla chiamata ricorsiva, calcola il valore di s_k con la formula in alto e pone $s[k] = s_k$ restituendolo in output.

  Una volta calcolati i valori s_j, prende il massimo degli n valori computati.
  Il tempo dell'algoritmo quindi è $O(n)$.