# *Algoritmi e Strutture Dati – 19/02/2021*

## Esercizio A1 – Punti $\geq 8$

È facile vedere che la funzione è $\Omega(n^3)$, per via della parte non ricorsiva. Proviamo quindi a vedere se la funzione è anche $O(n^3)$.

- **Caso base**: per $n = 1 \ldots 7$, si ottiene $T(1) = 1 \leq cn^3$. Tutte queste disequazioni sono soddisfatte dal caso $n = 1$, con $c \geq 1$.
- **Ipotesi induttiva**: ipotizziamo che per ogni valore $k$, $1 \leq k < n$: $T(k) \leq k^3$
- **Passo induttivo**: consideriamo tutti i casi con $n \geq 8$; ovviamente $\lfloor n/2 \rfloor$ e $\lfloor n/2\sqrt{2} \rfloor$ sono maggiori o uguali a 2, quindi è possibile applicare l'ipotesi induttiva; applichiamo la sostituzione e otteniamo:

$$
\begin{aligned}
T(n) &= 6T(\lfloor n/2 \rfloor) + T(\lfloor {}^{n}/_{(2\sqrt[3]{2})} \rfloor) + n^3 - n^2 \\
&\leq 6c\lfloor n/2 \rfloor^3 + c\lfloor {}^{n}/_{(2\sqrt[3]{2})} \rfloor^3 + n^3 - n^2 && \text{Sostituzione} \\
&\leq 6cn^3/8 + cn^3/16 + n^3 - n^2 && \text{Eliminazione } \lfloor\,\rfloor \\
&\leq \frac{6}{8}cn^3 + \frac{1}{16}cn^3 + n^3 && \text{Rimozione } -n^2 \text{ in quanto negativo} \\
&= \frac{13}{16}cn^3 + n^3 \stackrel{?}{\leq} cn^3 && \text{Passaggio algebrico}
\end{aligned}
$$

L'ultima disequazione è vera per $c \geq {}^{16}/_{3}$.

Abbiamo quindi dimostrato che $T(n) = O(n^3)$, per $m = 1$ e $c = {}^{16}/_{3}$, e da questo consegue che $T(n) = \Theta(n^3)$.

## Esercizio A2 – Fast sum – Punti $\geq 10$

La richiesta che la soluzione non sia lineare suggerisce di utilizzare un approccio basato su divide-et-impera. È possibile infatti notare che se io conoscessi quanti valori 1, 2, 3 sono presenti nel vettore, potrei ottenere la somma con un calcolo algebrico. Sfruttando una ricerca binaria, è possibile identificare l'ultima posizione di 1 e 2, e ottenere così il numero di elementi. Alternativamente, si potrebbe utilizzare l'algoritmo per ottenere la prima posizione mostrato nel compito del 2/7/2019 e adattare i calcoli.

```
int fastSum(int[] A, int n)
    int pos1 = lastPost(A, 1, n, 1)
    int pos2 = lastPost(A, 1, n, 2)
    return pos1 + (pos2 − pos1) · 2 + (n − pos2) · 3
```

Si noti che l'ultima formula può essere semplificata algebricamente in $3n - pos_1 - pos_2$

```
int lastPos(int[] A, int i, int j, int x)
    if i == j then
        return i
    else
        int m = ⌈(i + j)/2⌉
        if A[m] > x then
            return lastPos(A, i, m − 1, x)
        else
            return lastPos(A, m, j, x)
```

La complessità dell'algoritmo proposto è ovviamente $O(\log n)$.

## Esercizio A3 – Da foglia a foglia – Punti $\geq 12$

Per risolvere questo esercizio, è possibile utilizzare una singola visita in profondità dell'albero, che però raccoglie due informazioni separate.
Per ogni nodo $u$, vogliamo misurare:

- il costo più alto per andare da $u$ ad una qualunque delle sue foglie (*highest*);
- il costo massimo fra tutti i cammini semplici contenuti nel sottoalbero radicato in $u$ che uniscono due foglie (*maxpath*).

Calcolare *highest* di un nodo $u$ è semplice: basta prendere il massimo fra il valore *highest* dei figli sinistro e destro, e sommare il peso $u.weight$. Se $u$ è **nil**, si restituisce 0.
Per calcolare *maxpath* di un albero radicato in $u$, si prende il massimo fra:

- Il *maxpath* del sottoalbero sinistro;
- Il *maxpath* del sottoalbero destro;
- Il costo del cammino massimale che passa attraverso il nodo $u$, calcolato come il costo più alto per andare da $u$.left() ad una delle sue foglie, da $u$.right() ad una delle sue foglie, più il peso del nodo $u$.

Ci sono alcuni casi particolari:

- Nel caso $u$ == **nil**, si restituisce $-\infty$ per entrambi i valori; si assume che in questo modo, se un nodo $u$ non ha figlio destro oppure sinistro, la somma $highest_L + highest_R + T.weight)$ darà comunque $-\infty$ e non verrà selezionata dalla funzione max;
- Nel caso $u$ sia una foglia, si restituisce il peso del nodo sia come *highest* che come *maxpath*.

In ogni caso, il codice può essere reso più chiaro "sviscerando" ognuno dei quattro casi: **nil**, foglia, solo figlio sinistro, solo figlio destro, due figli.

```
int maxLeafLeaf(TREE T)
    highest, maxpath = maxPathRec(T)
    return maxpath
```

```
(int, int) maxPathRec(TREE T)
    if T == nil then
        return (−∞, −∞);
    if T.left() == nil and T.right() == nil then
        return (T.weight, T.weight);
    highest_L, maxpath_L = maxPath(T.left())
    highest_R, maxpath_R = maxPath(T.right())
    highest = max(highest_L, highest_R) + T.weight
    maxpath = max(maxpath_L, maxpath_R, highest_L + highest_R + T.weight)
    return highest, maxpath
```

La complessità dell'algoritmo proposto è quella di una visita in profondità di un albero, ovvero $\Theta(n)$.