Fernandez Fabio (M63001599) – Coraggio Valeria (M63001598)

HOMEWORK SET #1

Esercizi

Esercizio 1.1

È possibile ordinare le funzioni in ordine crescente di complessità asintotica nel modo seguente:

1. $f1(n) < f2(n) < f4(n) < f3(n)$

   f2(n) assume un andamento lineare, f4(n) un andamento quadratico, f3(n) assume un andamento esponenziale. Ne segue che asintoticamente f2<f4<f3. Inoltre, f1(n) < f2(n), in quanto applicando la definizione di big O si ottiene una disuguaglianza del tipo:

   $$\log(n) \leq c * n^{0.000001}$$

   verificata per *n* sufficientemente grande.

2. $f1(n) < f4(n) < f3(n) < f2(n)$

   f1(n) assume, infatti, un andamento costante; f4(n) può essere riscritto come $f4 = n^{\frac{3}{2}}$

   f3(n) ha un andamento quadratico, f2(n) un andamento esponenziale.

3. $f4(n) < f1(n) < f3(n) < f2(n)$

   f4(n) può essere riscritta come $\frac{n(n+1)}{2} + n$, quindi è $O(n^2)$.

   Confrontando f2 con f3 ed applicando la definizione di big-O è possibile ottenere una disequazione del tipo:

   $$n^{10} \leq c * \left(\sqrt{2}\right)^n$$

   verificata per *n* sufficientemente grande.

   Eseguendo il confronto tra f1 e f3 si ottiene:

   $$2^{\sqrt{n}logn} \leq 2^{\log(c*n^{10})+\frac{n}{2}}$$

   $$\sqrt{n} * logn \leq \log(c) + 10 * \log(n) + \frac{n}{2} \rightarrow logn \leq \frac{logc}{\sqrt{n}-10} + \frac{n}{2(\sqrt{n}-10)}$$

   $$logn \leq \frac{logc}{\sqrt{n}-10} + \sqrt{n}\frac{1}{2\left(1-\frac{10}{\sqrt{n}}\right)}$$

   verificata per *n* sufficientemente grande.

Esercizio 1.2

1. $\mathrm{f(n)} = \frac{\mathrm{n2-n}}{2},\ \mathrm{g(n)} = 6\mathrm{n}$

   g(n) è O(f(n)):

   $6n \leq c * \frac{(n^2-n)}{2} \rightarrow 12 \leq c(n-1)$ per $n \geq 13$, scelto c=1 la disequazione risulta verificata

   Non vale il viceversa.

2. $\mathrm{f(n)} = \mathrm{n} + 2\sqrt{\mathrm{n}},\ \mathrm{g(n)} = \mathrm{n}^2$

   f(n) è O(g(n)):

   f(n) è O(n), g(n) è $\theta(\mathrm{n}^2)$, quindi f(n) è O(n^2).

   Non vale il viceversa.

3. $\mathrm{f(n)} = \mathrm{nlogn},\ \mathrm{g(n)} = \frac{\mathrm{n}\sqrt{\mathrm{n}}}{2}$

   Confrontando l'andamento asintotico logaritmico con quello di $\sqrt{\mathrm{n}}$, si evince che f(n) è O(g(n)); non vale il viceversa.

4. $\mathrm{f(n)} = \mathrm{n} + \mathrm{logn},\ \mathrm{g(n)} = \sqrt{\mathrm{n}}$

   Confrontando l'andamento asintotico lineare con quello di $\sqrt{\mathrm{n}}$, si evince che g(n) è O(f(n)); non vale il viceversa.

5. $f(n) = 2(logn)^2,\ \ g(n) = logn + 1$

   $1 + \frac{1}{logn} \leq c * 2logn$

   per $n > 2$, scelto c=1 la disequazione risulta verificata.

   Pertanto, risulta che g(n) è O(f(n)), non vale il viceversa.

6. $\mathrm{f(n)} = 4\mathrm{nlogn} + \mathrm{n},\ \mathrm{g(n)} = \frac{\mathrm{n}^2-\mathrm{n}}{2}$

   f(n) ha una crescita asintotica limitata da O(nlogn), mentre g(n) da O(n^2), quindi risulta che f(n) è O(g(n)).

Esercizio 1.3

La prima affermazione risulta vera, in quanto $2 * 2^n \leq c * 2^n$ per $c = 2$ e per ogni n positivo.

La seconda affermazione risulta falsa, in quanto:

$2^n \leq c$ (con n>0)

non risulta soddisfatta per alcun valore di c.

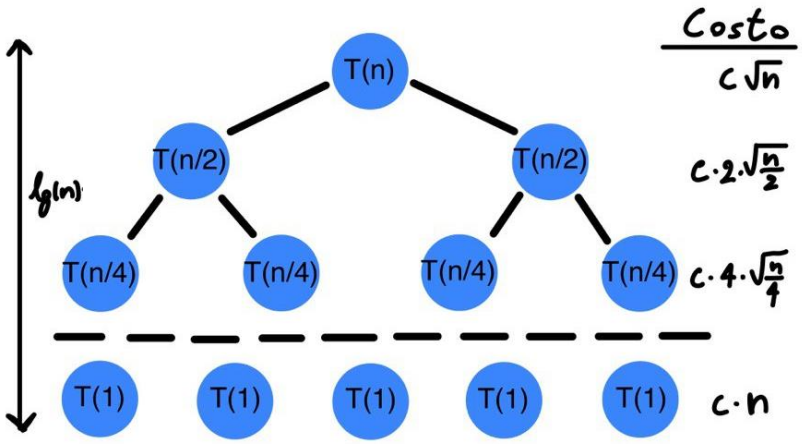
Esercizio 1.4

1. $T(n) = 2T\left(\frac{n}{2}\right) + O(\sqrt{n})$

   Utilizzando il teorema dell'esperto, con f(n)=O$(\sqrt{n})$, a=2 e b=2risulta che O$(\sqrt{n})$=O$(n^{1-\varepsilon})$ per $\frac{1}{2} \leq \varepsilon < 1$, pertanto:

   $$T(n) = \theta(n).$$

   Per ottenere la stessa soluzione è possibile utilizzare il metodo dell'albero delle ricorrenze.



$$T(n) = \sum_{i=0}^{logn-1} 2^i \sqrt{\frac{n}{2^i}} + cn = \sqrt{n} \sum_{i=0}^{logn-1} 2^{\frac{i}{2}} + cn \xrightarrow{j=\frac{i}{2}} \sqrt{n} \sum_{j=0}^{\frac{logn-1}{2}} 2^j + cn =$$

$$= \sqrt{n}\left(2^{\frac{1}{2}log n - \frac{1}{2}} - 1\right) + cn = \sqrt{n}\left(\frac{\sqrt{n}}{\sqrt{2}}\right) + cn = n + \frac{n}{\sqrt{2}} - \sqrt{n} = \theta(n)$$

2. $\mathrm{T(n)} = \mathrm{T}(\sqrt{\mathrm{n}}) + \Theta(\log \log \mathrm{n})$

Operando la sostituzione $m = \log(logn)$ ($\rightarrow n = 2^{2^m}$), è possibile applicare il teorema dell'esperto riconducendosi alla seguente forma:

$$T\left(2^{2^m}\right) = T\left(2^{2^{\frac{m}{2}}}\right) + \Theta(\mathrm{m}) \rightarrow R(m) = T\left(2^{2^m}\right) \rightarrow$$

$$R(m) = R\left(\frac{m}{2}\right) + \Theta(\mathrm{m})$$

La funzione $\Theta(\mathrm{m})$ risulta essere un $\Omega\left(\mathrm{m}^{\log_2 1+\varepsilon}\right)$ con $\varepsilon = 1$. Inoltre, vale che $\Theta\left(\frac{m}{2}\right) \leq c\Theta(m)$ per $\frac{1}{2} \leq c < 1$ ed n sufficientemente grande.

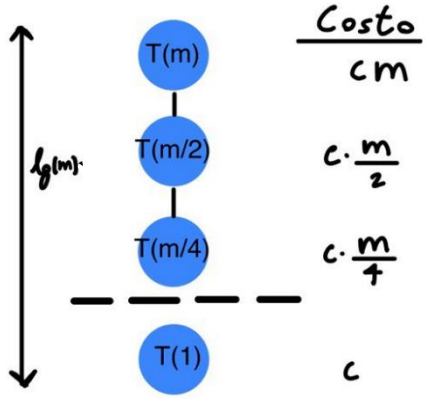
Pertanto, le condizioni del teorema dell'esperto sono rispettate ed è possibile affermare che

$$R(m) = \Theta(\mathrm{m})$$

Sostituendo $m = \log(logn)$, otteniamo anche T(n).

$$T(n) = \Theta(\log(logn)).$$

Per conseguire la stessa soluzione è possibile utilizzare il metodo dell'albero delle ricorrenze. Anche in questo caso eseguiamo la sostituzione:



$$R(m) = R\left(\frac{m}{2}\right) + \Theta(\mathrm{m})$$

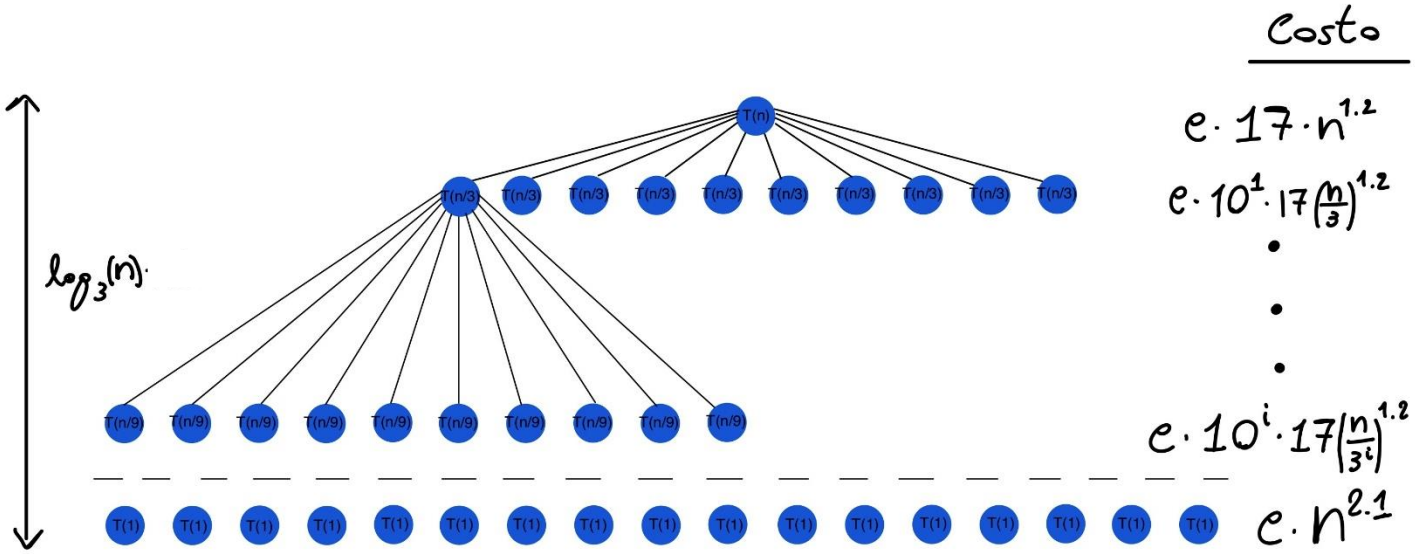
$$R(m) = c + c\sum_{i=0}^{logm-1} \frac{m}{2^i} \leq c + cm\sum_{i=0}^{+\infty} \left(\frac{1}{2}\right)^i = c(1+2m) = \Theta(\mathrm{m})$$

Sostituendo nuovamente $m = \log(logn)$, $T(n) = \Theta(\log(logn))$.

3. $T(n) = 10T\left(\frac{n}{3}\right) + 17n^{1.2}$

Utilizzando il teorema dell'esperto, con f(n)= $17n^{1.2}$, a=10 e b=3 risulta che $17n^{1.2}$=O($n^{2.1-\varepsilon}$), per $\varepsilon \geq 0.9$ pertanto T(n)= $\theta\left(n^{\log_3 10}\right) = \theta(n^{2,1})$

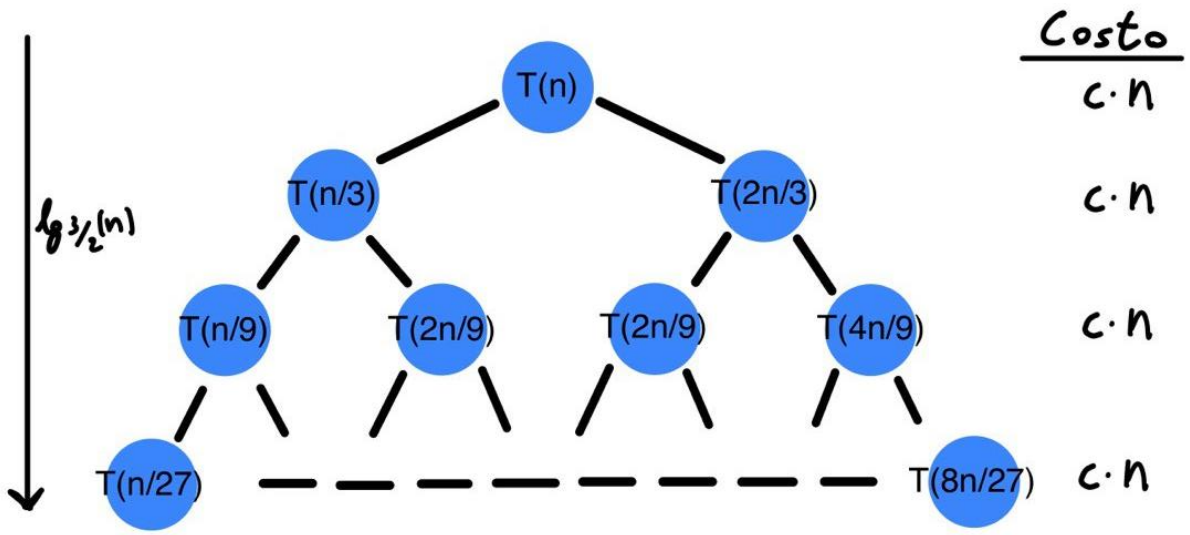
Applicando il metodo dell'albero delle ricorrenze, invece, vale che il costo generalizzato per ogni livello è $c * 10^i * 17\left(\frac{n}{3^i}\right)^{1.2}$, mentre i nodi alla frontiera sono $10^h, ovvero\ 10^{\log_3 n} = n^{\log_3 10}$ e hanno un costo c, costante rispetto ad n.



$$T(n) = c * 17\sum_{i=0}^{\log_3 n-1} 10^i * \left(\frac{n}{3^i}\right)^{1,2} + cn^{2,1} = 17cn^{1,2}\sum_{i=0}^{\log_3 n-1}\left(\frac{10}{3^{1,2}}\right)^i + cn^{2,1} =$$

$$17cn^{1,2} * \frac{\frac{10}{3^{1,2}}^{\log_3 n}-1}{\frac{10}{3^{1,2}}-1} + cn^{2,1} = 17cn^{1,2} * \frac{n^{\log_3 \frac{10}{3^{1,2}}}-1}{\frac{10}{3^{1,2}}-1} + cn^{2,1} = 17cn^{1,2} * \frac{n^{0,9}-1}{\frac{10}{3^{1,2}}-1} + cn^{2,1} = \theta(n^{2,1})$$

4. $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$



Utilizzando il metodo dell'albero di ricorsione, si ottiene un albero non perfettamente bilanciato, per il quale il ramo associato al termine $\left(\frac{2n}{3}\right)$ assume un peso maggiore man mano che si scende per i livelli e ci si allontana dalla radice. Ciò significa che non viene duplicato esattamente il numero di nodi ad ogni livello, come si otterrebbe in un albero perfettamente bilanciato: a causa di questa asimmetria ci si aspetta che il costo totale dell'albero di ricorsione per la nostra ricorrenza sia inferiore a quello di un albero perfettamente bilanciato. Tuttavia, dall'analisi di complessità si ottiene una dipendenza da nlogn, per cui è possibile ipotizzare che la soluzione della ricorrenza sia Ω(nlogn). È possibile, quindi, verificare la soluzione proposta applicando il metodo di sostituzione.

Dobbiamo dimostrare che ∃ d > 0 tale che T(n) ≥ dnlogn

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn \geq d\left(\frac{n}{3}\right)\log\left(\frac{n}{3}\right) + d\left(\frac{2n}{3}\right)\log\left(\frac{2n}{3}\right) + cn$$

$$= \left[d\left(\frac{n}{3}\right)logn - d\left(\frac{n}{3}\right) + log3\right] + \left[d\left(\frac{2n}{3}\right)logn - d\left(\frac{2n}{3}\right)\log\left(\frac{3}{2}\right)\right] + cn$$

$$= dnlogn - d\left[\left(\frac{n}{3}\right)log3 + \left(\frac{2n}{3}\right)\log\left(\frac{3}{2}\right)\right] + cn$$

$$= dnlogn - d\left[\left(\frac{n}{3}\right)log3 + \left(\frac{2n}{3}\right)log3 - \left(\frac{2n}{3}\right)log2\right] + cn$$

$$= dnlogn - dn\left(log3 - \frac{2}{3}\right) + cn$$

$$\text{dnlogn} - \text{dn}\left(\log 3 - \frac{2}{3}\right) + cn \geq dnlogn \leftrightarrow cn - dn\left(log3 - \frac{2}{3}\right) \geq 0 \rightarrow d \leq \frac{c}{\left(log3 - \frac{2}{3}\right)}$$

Di conseguenza è possibile affermare T(n) è Ω(nlogn).

Problemi

Problema 1.1

Si è scelto di implementare l'algoritmo di risoluzione in C (vedi file allegato).

Sono riportati di seguito 4 di casi di test forniti dal programma.

```
Inserisci il numero di test case: 4

Inserire il numero di elementi del vettore:5

9
1
0
5
4

numero inversioni(0): 6
vettore 0: 0 1 4 5 9
Inserire il numero di elementi del vettore:3

1
2
3

numero inversioni(1): 0
vettore 1: 1 2 3
Inserire il numero di elementi del vettore:5

5
4
3
2
1

numero inversioni(2): 10
vettore 2: 1 2 3 4 5
```

```
Inserire il numero di elementi del vettore:6

6
1
2
3
4
5

numero inversioni(3): 5
vettore 3: 1 2 3 4 5 6


...Program finished with exit code 0
Press ENTER to exit console.
```

L'algoritmo fa uso di un approccio divide et impera e ripercorre la strategia del Merge Sort con l'aggiunta della logica per il conteggio delle inversioni, affinché sia rispettata la proprietà richiesta dalla traccia:
"in una sequenza A, la coppia(i,j) è un'inversione se i<je Ai> Aj".
Tale logica ha un costo costante $\theta(1)$, di conseguenza la complessità resta quella dell'algoritmo Merge Sort, $\theta(nlogn)$ nel caso peggiore.

Problema 1.2

Si è scelto di implementare l'algoritmo di risoluzione in C++ (vedi file allegato).

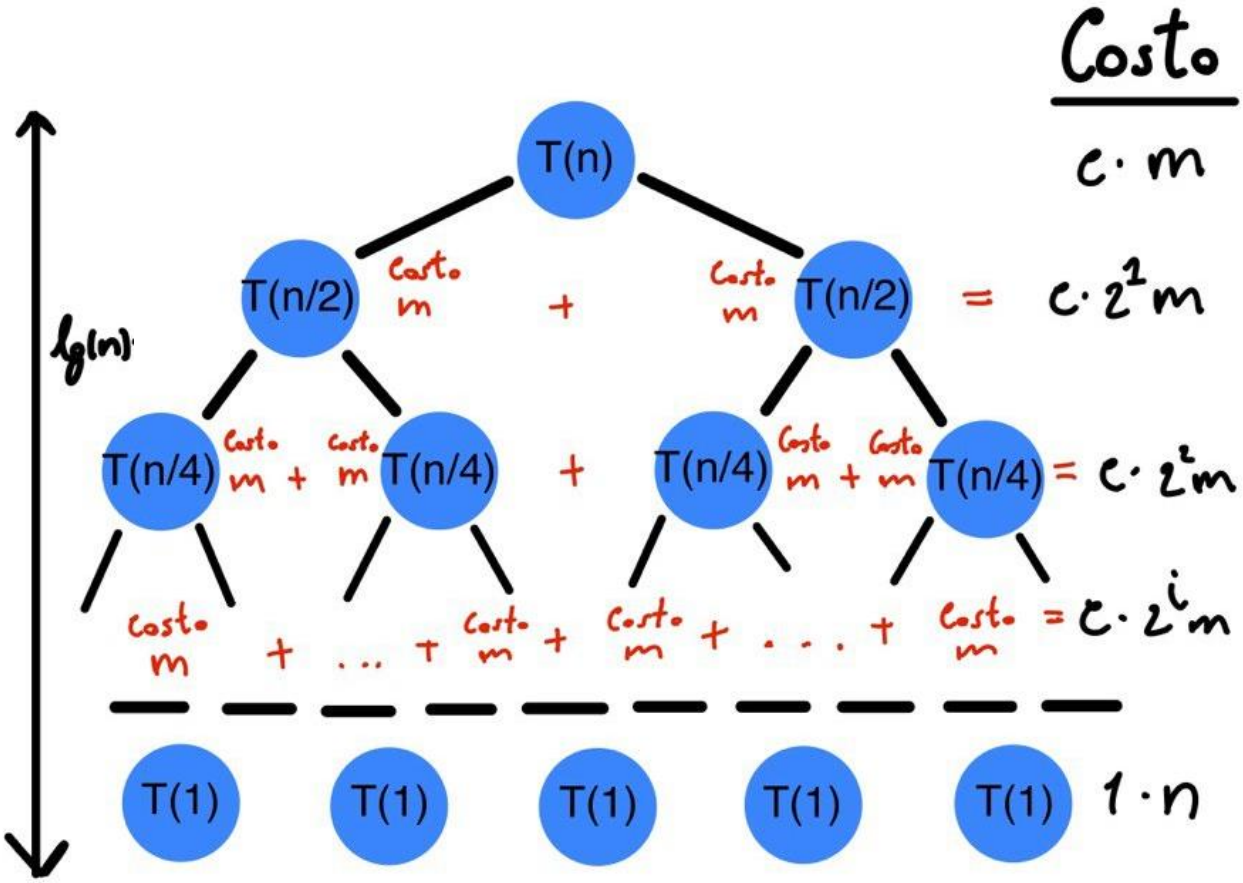
Sono riportati di seguito degli esempi di casi di test forniti dal programma.

```
Inserisci il numero di casi di test: 4
Inserisci le parole per il caso 1 (termina con 0):
apple
ape
april
applied
0
Risultato per il caso 1: ap
Inserisci le parole per il caso 2 (termina con 0):
kiwi
kitchen
kite
kitten
0
Risultato per il caso 2: ki
Inserisci le parole per il caso 3 (termina con 0):
lemon
leopard
lemonade
lemony
0
Risultato per il caso 3: le
Inserisci le parole per il caso 4 (termina con 0):
strawberry
string
stratosphere
structure
0
Risultato per il caso 4: str


...Program finished with exit code 0
Press ENTER to exit console.
```

L'algoritmo fa uso di un approccio divide et impera, di conseguenza il set di stringhe associato al problema iniziale viene suddiviso per ogni caso di test in sottoproblemi di complessità dimezzata. Nel caso peggiore, tutte le stringhe inserite sono identiche e hanno la stessa lunghezza l. Ogni livello di ricorsione ha, quindi, una complessità pari a $\theta(l * 2^i)$. Il numero di livelli di ricorsione è $\theta(logn)$.

Applicando il metodo dell'albero delle ricorrenze, si ottiene che dell'algoritmo:



$$T(n) = \sum_{i=0}^{logn-1} cl * \ 2^i + cn = cl \ \frac{2^{logn}-1}{2-1} + cn = cl(n-1) + cn = \ \theta(\mathrm{l} * \mathrm{n})$$

nel caso peggiore.

Problema 1.3

L'algoritmo è stato implementato in C++ (vedi file allegato).

Per quanto concerne lo studio di complessità, la procedura TreapInsert prevede la somma di due componenti, una dovuta all'inserimento e una dovuta alle rotazioni. Nel caso peggiore, cioè il caso in cui l'albero coincide con una lista e le priorità sono disposte in ordine decrescente rispetto alla radice, l'inserimento ha una complessità pari a $\theta(\mathrm{n})$ e le rotazioni pari a $\theta(\mathrm{n})$, quindi la complessità totale della procedura risulta $\theta(\mathrm{n})$.

Considerando, invece, un caso medio, in cui l'albero differisce da una lista e le priorità sono più bilanciate, l'inserimento ha una complessità pari a $\theta(\mathrm{logn})$ e le rotazioni pari a $\theta(\mathrm{logn})$, quindi la complessità totale della procedura risulta $\theta(\mathrm{logn})$.