

Docente: Maria Rita Di Berardini

Prova Parziale del 18/12/2009 – soluzioni

Rispondere alle seguenti domande:

1. Perché è utile la notazione asintotica?

Perché ci consente (1) di stimare gli ordini di grandezza delle funzioni, ignorando dettagli non rilevanti (come costanti e termini di ordine inferiore) e (2) di classificare gli algoritmi in base al loro comportamento asintotico.

2. Quali sono i vantaggi derivanti dall'analisi di algoritmi in termini di tempo di esecuzione?

L'analisi di algoritmi in termini di tempo di esecuzione consente di: (1) stimare il tempo necessario all'esecuzione di un algoritmo, (2) stimare il più grande input gestibile in termini ragionevoli, (3) confrontare l'efficienza di due algoritmi diversi, ma che risolvono lo stesso problema, (4) ottimizzare le parti critiche.

3. Vero o falso (giustificare le risposte fornite):

(a)  $2^n = \Theta(2^{3n})$ ;

Falso. Infatti  $2^{3n} = (2^3)^n = 8^n$  e  $2^n \neq \Omega(2^{3n})$ . Infatti, se fosse  $2^n = \Omega(8^n)$  allora esisterebbero delle costanti positive  $c$  e  $n_0$  tali che  $2^n \geq c8^n$  per ogni  $n \geq n_0$ , e quindi  $(1/4)^n \geq c$  per ogni  $n \geq n_0$ . Il che è impossibile.

(b)  $5n^2 + 9n\sqrt{n} = O(n^2)$ ;

Vero. Infatti per ogni  $n \geq 1$ ,  $5n^2 + 9n\sqrt{n} \leq 5n^2 + 9n^2 = 14n^2$ .

(c)  $\log n = \Omega(\log \log n^4)$ .

Vero. Infatti, per ogni  $n \geq 4$ ,  $\log \log n^4 = \log(4 \log n) \leq \log(4n) = \log 4 + \log n = 2 + \log n \leq \log n + \log n = 2 \log n$ . In altri termini, per ogni  $n \geq 4$ ,  $\log n \geq (1/2) \log \log n^4$ .

4. Considerate la seguente ricorrenza  $T(n) = 4T(n/2) + 2n^2 + n$ .

(a) Si risolva  $T(n)$  applicando il metodo del master;

In questo caso  $\log_b a = \log_2 4 = 2$  e  $f(n) = n^2 + n = \Theta(n^2) = \Theta(n^{\log_b a})$  (Caso 2 del teorema del master). Quindi  $T(n) = \Theta(n^{\log_b a} \log n) = n^2 \log n$ .

(b) Si dica (motivando la risposta fornita) se  $T(n) = \Omega(n^3)$ .

Se fosse  $T(n) = \Omega(n^3)$  avremmo anche  $n^2 \log n = \Omega(n^3)$  il che è impossibile.

5. Usate il metodo iterativo per risolvere la ricorrenza  $T(n) = 2T(n/2) + \sqrt{n}$ .

$$\begin{aligned} T(n) &= \sqrt{n} + 2T\left(\frac{n}{2}\right) \\ &= \sqrt{n} + 2\sqrt{\frac{n}{2}} + 4T\left(\frac{n}{4}\right) \\ &= \sqrt{n} + 2\sqrt{\frac{n}{2}} + 4\sqrt{\frac{n}{4}} + 8T\left(\frac{n}{8}\right) \\ &= \sqrt{n} + 2\sqrt{\frac{n}{2}} + 4\sqrt{\frac{n}{4}} + 8\sqrt{\frac{n}{8}} + 16T\left(\frac{n}{16}\right) \\ &\vdots \\ &= \sum_{i=0}^{k-1} 2^i \sqrt{\frac{n}{2^i}} + 2^k T\left(\frac{n}{2^k}\right) \end{aligned}$$

Ci fermiamo quando  $k = \log n$ . Quindi

$$T(n) = \sum_{i=0}^{\log n - 1} 2^i \sqrt{\frac{n}{2^i}} + 2^{\log n} = \sum_{i=0}^{\log n - 1} 2^i \sqrt{\frac{n}{2^i}} + n$$

Ora  $2^i \sqrt{\frac{n}{2^i}} = 2^i \frac{\sqrt{n}}{\sqrt{2^i}} = 2^i \frac{\sqrt{n}}{\sqrt{2}^i} = \sqrt{n} \left(\frac{2}{\sqrt{2}}\right)^i$ . Allora:

$$\begin{aligned} \sum_{i=0}^{\log n - 1} 2^i \sqrt{\frac{n}{2^i}} &= \sqrt{n} \sum_{i=0}^{\log n - 1} \left(\frac{2}{\sqrt{2}}\right)^i = \sqrt{n} \frac{1 - \left(\frac{2}{\sqrt{2}}\right)^{\log n}}{1 - \frac{2}{\sqrt{2}}} = \sqrt{n} \frac{\left(\frac{2}{\sqrt{2}}\right)^{\log n} - 1}{\frac{2}{\sqrt{2}} - 1} = \\ &= \sqrt{n} \frac{n^{\log \frac{2}{\sqrt{2}}} - 1}{\frac{2 - \sqrt{2}}{\sqrt{2}}} = \frac{\sqrt{2}}{2 - \sqrt{2}} \sqrt{n} (\sqrt{n} - 1) = \frac{\sqrt{2}}{2 - \sqrt{2}} n - \frac{\sqrt{2}}{2 - \sqrt{2}} \sqrt{n} \end{aligned}$$

N. B:  $\frac{2}{\sqrt{2}} = \sqrt{2} = 2^{1/2}$ ; quindi  $n^{\log \frac{2}{\sqrt{2}}} = n^{1/2} = \sqrt{n}$ .

Possiamo infine concludere che

$$T(n) = \frac{\sqrt{2}}{2 - \sqrt{2}} n - \frac{\sqrt{2}}{2 - \sqrt{2}} \sqrt{n} + n = \frac{2}{2 - \sqrt{2}} n - \frac{\sqrt{2}}{2 - \sqrt{2}} \sqrt{n} = \Theta(n)$$

6. Si valuti la complessità del seguente algoritmo in funzione di  $n$

ALGORITMO( $n$ )

```

1   $i \leftarrow 1$ 
2   $r \leftarrow 0$ 
3  while  $i \leq n$ 
4  do for  $j \leftarrow 1$  to  $n - i$ 
5      do  $r \leftarrow r + i$ 
6       $i \leftarrow i + 1$ 
7  return  $r$ 
```

il costo dell'algoritmo è dato dai cicli di riga 3 e 4. Il ciclo **for** di riga 4 ha un costo pari a  $c(n - i)$  (qui con  $c$  indichiamo il costo (costante) dei due assegnamenti che costituiscono il corpo del ciclo in questione). La variabile di controllo  $i$  del ciclo **while** di riga varia da 1 ad  $n$ . Il costo dell'algoritmo è  $T(n) = \sum_{i=1}^n c(n - i) =$

$$c \sum_{j=0}^{n-1} j = c \frac{n(n-1)}{2}.$$

7. Un docente utilizza il numero di matricola (6 cifre decimali) come chiave per memorizzare in una tabella i suoi 200 studenti. Quale è il fattore di carico della tabella?

La dimensione  $m$  della tabella è pari a  $10^6$  (il numero totale di numeri di matricola distinti che possiamo formare con 6 cifre decimali), il fattore di carico è quindi  $\frac{200}{10^6} = 0,0002$

8. Quale delle seguenti è una definizione di collisione?

- (a) Si ha una collisione quando alla stessa chiave possono corrispondere valori diversi della funzione hash;
- (b) Si ha una collisione quando a chiavi diverse corrispondono valori diversi della funzione hash;
- (c) Si ha una collisione quando a chiavi diverse corrisponde lo stesso valore della funzione hash;
- (d) Si ha una collisione quando il valore della chiave coincide con quello della funzione hash.

Risposta esatta (c)

9. Dare una definizione del fenomeno conosciuto come *agglomerazione primaria* e discutere il suo impatto sul numero medio di accessi per una ricerca senza successo. Indicare almeno un metodo di scansione che elimina questo problema.

È un fenomeno (legato alle tabelle hash ad indirizzamento aperto) che si verifica quando la funzione hash utilizzata per gestire le collisioni è tale da accumulare posizioni occupate per lunghi tratti della tabella. Tale fenomeno ha un impatto sostanziale sul numero medio di accessi per una ricerca senza successo (vedi le slides del corso al riguardo). Tale problema può essere eliminato se si usa come metodo di scansione la scansione quadratica (che però soffre di un altro problema, l'agglomerazione secondaria) oppure l'hashing doppio.

10. Si consideri la sequenza di numeri

34, 12, 36, 19, 16, 24, 17, 27, 25, 15.

Dire, motivando la risposta, se tale sequenza rappresenta un max-heap oppure no.

Non è un max-heap perchè la proprietà del max-heap non è verificata. Ad esempio,  $36 = A[3] > A[\text{parent}[3]] = A[1] = 34$ .

Se no, si applichi la procedura BUILDMAXHEAP per costruire un max-heap contenente gli stessi numeri. Si mostri, infine il max-heap risultante dall'eliminazione di 36 e 34.

La procedura BUILDMAXHEAP, esegue una serie di chiamate di MAXHEAPIFY( $A, i$ ) per  $i = \lfloor 10/2 \rfloor = 5, \dots, 1$ . MAXHEAPIFY( $A, 5$ ) non effettua alcuno scambio; MAXHEAPIFY( $A, 4$ ) scambia 27 con 19; MAXHEAPIFY( $A, 3$ ) non effettua alcuno scambio; MAXHEAPIFY( $A, 2$ ) scambia 12 prima con 27 e poi con 25; MAXHEAPIFY( $A, 1$ ) scambia 34 con 36. Otteniamo così il max-heap

36	27	34	25	16	24	17	19	12	15
----	----	----	----	----	----	----	----	----	----

Per eliminare 36 dobbiamo rimpiazzare la chiave in posizione 1 con quella in posizione  $\text{heapsize}[A] = 10$  (ossia 15), decrementare di uno la  $\text{heapsize}$ , e poi eseguire una chiamata di MAXHEAPIFY( $A, 1$ ). Questa chiamata scambia 15 prima con 34 e poi con 24. Il risultato è il seguente:

34	27	24	25	16	15	17	19	12	
----	----	----	----	----	----	----	----	----	--

Per eliminare 34 dobbiamo rimpiazzare la chiave in posizione 1 con quella in posizione  $\text{heapsize}[A] = 9$  (ossia 12), decrementare di uno la  $\text{heapsize}$ , e poi eseguire una chiamata di MAXHEAPIFY( $A, 1$ ). Questa chiamata scambia 12 (nell'ordine) con 27, 25 e 19. Il risultato è il seguente:

27	25	24	19	16	15	17	12		
----	----	----	----	----	----	----	----	--	--

11. Dato un intero  $n > 2$ , sia  $T$  l'albero di ricerca binaria (BST) formato da un solo nodo di etichetta  $n$ . Eseguiamo su  $T$  il seguente algoritmo che introduce nuovi nodi nell'albero utilizzando la tradizionale procedura di inserimento di un nodo in un BST.

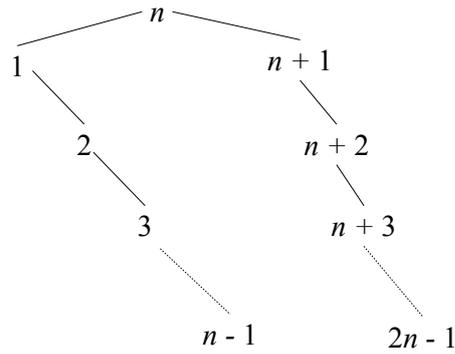
```

ADD()
1  for  $i \leftarrow 1$  to  $n - 1$ 
2  do TREE-INSERT( $T, i$ )
3  TREE-INSERT( $T, n + i$ )

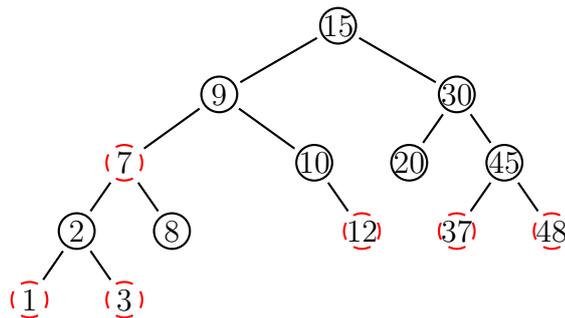
```

Descrivere il BST prodotto da questo algoritmo.

In generale otteniamo un albero della seguente forma:

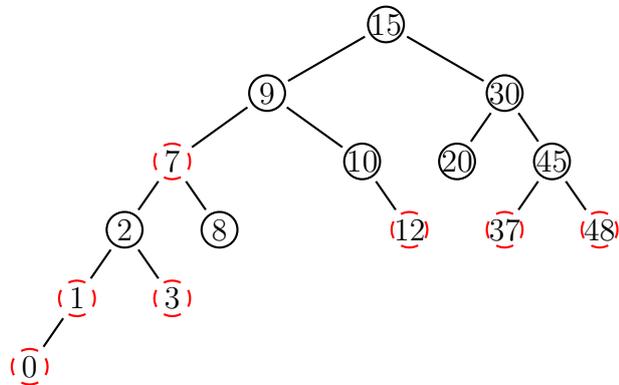


12. Si consideri il seguente albero rosso-nero

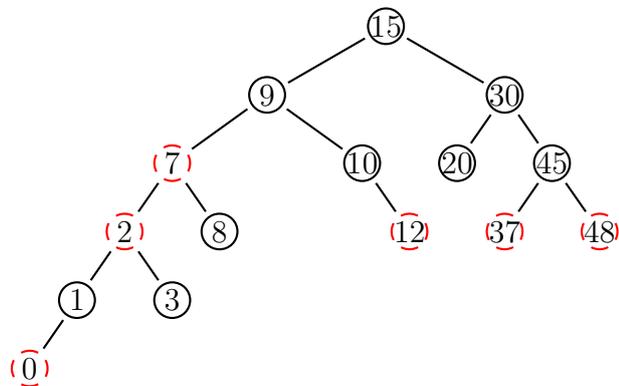


(a) Si mostri l'albero ottenuto inserendo 0 e poi 11 (evidenziare eventuali rotazioni e ricolorazioni di nodi);

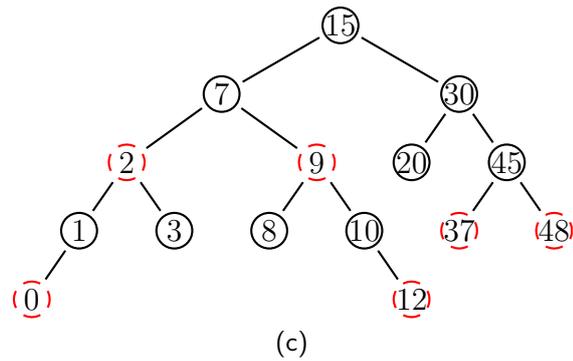
Inserimento di 0: eseguiamo nell'ordine le seguente ricolazioni/rotazioni



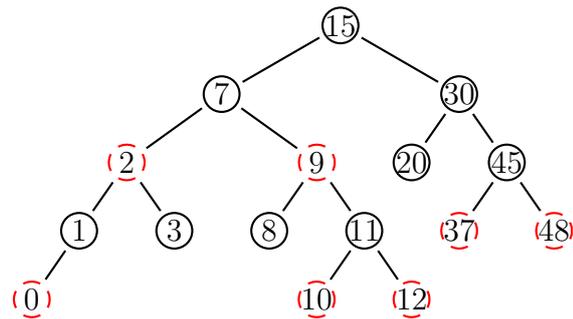
(a) - caso 1: lo zio 3 di 0 è rosso



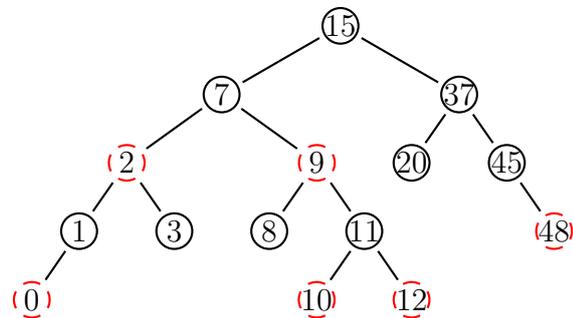
(b) - caso 2: lo zio 10 di 2 è nero e 2 è figlio destro



Inserimento di 11: indico solo il risultato finale (manca una rotazione intermedia infatti 11 inserito come figlio sinistro di 12, causa una rotazione di 12 a destra)



- (b) Si mostri l'evoluzione dell'albero a seguito della cancellazione di 30.  
 Cancellare 30 (nodo con 2 figli) significa rimpiazzare 30 con il minimo del suo sottoalbero destra (ossia 37) e poi eliminare 37 (una foglia). Otteniamo il seguente albero:



13. Quanti scambi vengono effettuati dall'algoritmo SELECTIONSORT?

- (a)  $O(n)$  nel caso peggiore;
- (b)  $O(n^2)$  nel caso peggiore;
- (c)  $O(n \log n)$  nel caso medio;
- (d)  $O(1)$  nel caso migliore.

Risposta esatta: (a). Qui Chiede il numero di scambi e non di confronti. Il SELECTIONSORT nel caso peggiore (ma anche negli altri casi) esegue esattamente  $n$  scambi.

14. Eseguire il COUNTINGSORT sull'input 7, 6, 5, 6, 2, 0, 8, 2, 3, 1, 9.

15. Scrivere un algoritmo che, data in input una sequenza di  $n$  numeri positivi contenuta in un array  $A[1 \dots n]$ , utilizzi la tecnica divide-et-impera per determinare se la sequenza contiene un numero dispari di 3. Fornire inoltre la ricorrenza che descrive la complessità dell'algoritmo proposto.

Forniamo una procedura ricorsiva che restituisce true solo se  $A[i \dots j]$  (la porzione del vettore  $A$  compresa tra gli indici  $i$  e  $j$ ) contiene un numero dispari di 3.

```

ODD3(A, i, j)
1  if i = j
2    then return (A[i] == 3)
3  m ← ⌊(i + j)/2⌋
4  return ODD3(A, i, m) xor ODD3(A, m + 1, j)

```

Se  $i = j$  (la porzione di vettore che stiamo esaminando contiene un solo elemento), restituiamo true solo se tale elemento è 3. Se  $i \neq j$ , la porzione che stiamo esaminando contiene almeno due elementi. In questo caso, identifichiamo l'elemento intermedio (quello in posizione  $m = \lfloor (i + j)/2 \rfloor$ ) e richiamiamo la stessa procedura sulle porzioni di vettore comprese tra gli indici  $i$  e  $m$ , e  $m + 1$  e  $j$ . Ora, non ci rimane che combinare le soluzioni fornite da  $\text{ODD3}(A, i, m)$  e  $\text{ODD3}(A, m + 1, j)$  tenendo conto del fatto che  $A[i \dots j]$  contiene un numero dispari di 3 solo se  $\text{ODD3}(A, i, m)$  oppure  $\text{ODD3}(A, m + 1, j)$ , ma non entrambe, restituiscono true. Abbiamo quindi usato l'operatore xor (o or esclusivo).

A questo punto per verificare se la nostra sequenza di  $n$  numeri contiene un numero dispari di 3 eseguiamo  $\text{ODD3}(A, 1, n)$ .