4

Esercizi Prima Parte

4.1 Modelli, di calcolo, analisi asintotica e ricorrenze

Esercizio 4.1. Rispondere alle seguenti domande:

1. Come misuriamo l'efficienza di un algoritmo?
2. Quali sono gli algoritmi più efficienti, quelli ricorsivi o quelli iterativi?
3. Perchè è utile l'analisi asintotica?
4. Perchè è importante determinare l'occupazione di memoria di un algoritmo?

Esercizio 4.2. Quale delle seguenti affermazioni vera?

1. $\log n = \Omega(n)$;
2. $n = \Omega(\log n)$;
3. $\log n = O(\log \log n)$;
4. nessuna delle precedenti è vera.

Esercizio 4.3. Quale delle seguenti affermazioni vera?

1. $\log n = O(\log \log n)$;
2. $n = \Theta(\log n)$;
3. $\log \log n = O(\log \log n)$;
4. nessuna delle precedenti è vera.

Esercizio 4.4. È vero che $f(n) + g(n) = \Theta(\min\{f(n), g(n)\})$? È vero che $f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$?

Esercizio 4.5. Dimostrare che $f(n) = \Theta(h(n))$ e $g(n) = \Theta(h(n))$ allora $f(n) = \Theta(g(n))$.

Soluzione: Se $f(n) = \Theta(h(n))$ esistono delle costanti c_{1f}, c_{2f} ed n_{0f} tali che $c_{1f}\, h(n) \leq f(n) \leq c_{2f}\, h(n)$ per ogni $n \geq n_{0f}$. In maniera analoga, se $g(n) = \Theta(h(n))$ esistono delle costanti c_{1g}, c_{2g} ed n_{0g} tali che $c_{1g}\, h(n) \leq g(n) \leq c_{2g}\, h(n)$ per ogni $n \geq n_{0g}$.

Ora, per ogni $n \geq n_0 = \max\{n_{0f}, n_{0g}\}$, abbiamo che $f(n) \geq c_{1f}\, h(n)$ e $g(n) \leq c_{2g}\, h(n)$ e, quindi, $h(n) \geq \frac{1}{c_{2g}}\, g(n)$. Allora $f(n) \geq \frac{c_{1f}}{c_{2g}}\, g(n)$. Inoltre, se $n \geq n_0$ allora $f(n) \leq c_{2f}\, h(n)$ e $h(n) \leq \frac{1}{c_{1g}}\, g(n)$ implica $f(n) \leq \frac{c_{2f}}{c_{1g}}\, g(n)$. Ricapitolando: per ogni $n \geq n_0$ abbiamo che

$$\frac{c_{1f}}{c_{2g}}\, g(n) \leq f(n) \leq \frac{c_{2f}}{c_{1g}}\, g(n)$$

e quindi $f(n) = \Theta(g(n))$.

Esercizio 4.6. Dimostrare che

$$\sum_{i=1}^{n} \log i = \Theta(n \log n)$$

Soluzione: $\sum_{i=1}^{n} \log i = \log 1 + \log 2 + \cdots \log n = \log(1 \cdot 2 \cdot \cdots \cdot n) = \log n!$ Ora $n! \leq n^n$ implica $\log n! \leq \log n^n = n \log n$ e quindi $\log n! = O(n \log n)$. Inoltre, se usiamo l'approssimazione di Stirling ($n! = \sqrt{2\pi n}\,(\frac{n}{e})^n\,(1 + \Theta(\frac{1}{n}))$, abbiamo che $n! \geq (\frac{n}{e})^n$ e quindi $\log n! \geq \log(\frac{n}{e})^n = n \log \frac{n}{e} = \Theta(n \log n)$. Possiamo quindi concludere che $\log n! = \Omega(n \log n)$.

Esercizio 4.7. In ciascuno dei seguenti casi dire se $f(n) = O(g(n))$, se $f(n) = \Omega(g(n))$ oppure entrambi.

1. $f(n) = n - \sqrt{n}$ e $g(n) = 5n + 2\sqrt{n} \log n$;
2. $f(n) = 10n \log n$ e $g(n) = \frac{n}{10} \sqrt[4]{n} = \frac{1}{10} n^{1+1/4}$;
3. $f(n) = 3 \log n$ e $g(n) = \log n^8$;
4. $f(n) = 7^n$ e $g(n) = 8^n$;
5. $f(n) = 3^{\log_2 n}$ e $g(n) = 2^{\log_3 n}$;
6. $f(n) = n4^n$ e $g(n) = n!$;
7. $f(n) = 3^{n+2}$ e $g(n) = 3^{n-3}$;
8. $f(n) = n^2 - n^{3/2}$ e $g(n) = n^2 + n^{2/3}$;
9. $f(n) = \log n^8$ e $g(n) = \log^2 n$;
10. $f(n) = n^2(1 + \cos n)$ e $g(n) = n^2$.

Esercizio 4.8. Dimostrate che $n^3 + 5n^2 = O(n^3)$. Dimostrate inoltre che $n^3 + 5n^2 = \Omega(n^3)$.

Esercizio 4.9. 1. Dimostrate che $cn + d = O(2^n)$ per ogni $c, d > 0$.
2. Dimostrate che $cn^k + d = O(2^n)$ per ogni $c, d, k > 0$.

Esercizio 4.10. Provate che

1. $10n^2 + 5\sqrt{n} \log n^2 = \Theta(n^2)$;
2. $(n \log n)3^n = O(4^n/n)$

Soluzione:

1. $f(n) = 10n^2 + 5\sqrt{n}\log n^2 = 10n^2 + 10\sqrt{n}\log \leq 10n^2 + 10n^2 = 20n^2$, il che dimostra che $f(n) = O(n^2)$. Al contrario $f(n) \geq 10n^2$ e quindi $f(n) = \Omega(n^2)$.
2. Dimostrare che $f(n) = (n+\log n)3^n = O(4^n/n)$ equivale a dimostrare che esistono delle costanti positive c ed n_0 tali che

$$f(n) \leq c\,4^n/n \text{ per ogni } n \geq n_0$$

ossia tali che

$$nf(n) = n(n+\log n)3^n \leq c\,4^n \text{ per ogni } n \geq n_0$$

Dividendo tutto per 3^n otteniamo

$$n(n+\log n) \leq c\,(4/3)^n \text{ per ogni } n \geq n_0$$

In altri termini, possiamo dimostrare che $f(n) = O(4^n/n)$ dimostrando che $n(n+\log n) = O((4/3)^n)$.

A questo punto:

1. $\log n \leq n$ implica $n(n+\log n) \leq n\,(n+n) = 2n^2$ per ogni $n \geq 1$.
2. Inoltre, sappiamo che $n^k = O(a^n)$ per ogni $k \geq 1$ e $a > 1$. In particolare, $n^2 = O((4/3)^n)$ (qui abbiamo scelto $k = 2$ e $a = 4/3 > 1$).
3. Infine, $n(n+\log n) \leq 2n^2$ e $n^2 = O((4/3)^n)$ implicano $n(n+\log n) = O((4/3)^n)$.

Esercizio 4.11. Per la soluzione di un problema, vi e stato proposto di scegliere fra un algoritmo iterativo che ha complessità $\Theta(n^2)$ e un algoritmo ricorsivo la cui complessità è data dalla seguente ricorrenza

$$T(n) = \begin{cases} aT(n/4) + 1 & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

dove $a \in \mathbb{N}$ maggiore di 1. Quale algoritmo preferite? Motivare la risposta.

Soluzione: Dipende dal valore della costante a usata nella definizione di $T(n)$. Infatti, $a > 1$ implica $\log_b a = \log_4 a > 0$, $f(n) = 1 = O(1) = O(n^{\log_4 a - \epsilon})$ (caso 1 del teorema del Master) e quindi $T(n) = \Theta(n^{\log_4 a})$. Ora se $a < 16$ (e quindi $\log_4 a < 4$) la versione ricorsiva è più efficiente di quella ricorsiva, se $a = 16$ il due algoritmi hanno lo stesso comportamento asintotico, se infine $a > 16$ la versione iterativa è da preferire a quella ricorsiva.

Esercizio 4.12. Assumete $T(1) = 1$ e dimostrate (usando il metodo degli alberi di ricorsione) che $T(n) = T(n/4) + T(3/4n) + n$ è $O(n\log n)$.

Esercizio 4.13. Assumete $T(1) = 1$ e dimostrate (usando il metodo della sostituzione) che $T(n) = T(n/3) + T(2/5n) + n$ è $O(n)$.

Soluzione: In base alla definizione della notazione O dobbiamo dimostrare che esistono delle costanti positive c, n_0 tali che $T(n) \leq cn$ per ogni $n \geq n_0$. Procediamo per induzione su n.

Caso base ($n = 1$): $T(1) = 1 \leq c$ a patto di scegliere $c \geq 1$.

Passo induttivo ($n > 1$): Per ipotesi induttiva abbiamo che $T(n/3) \leq c(n/3)$ e $T(2/5n) \leq c(2/5)n$. Allora, $T(n) = T(n/3) + T(2/5n) + n \leq c(n/3) + c(2/5)n + n = (c/3 + (2/5)c + 1)n$. Per terminare la prova non ci resta che capire quale condizioni imporre su c affinchè $c/3+(2/5)c+1 \leq c$.

$$c/3 + (2/5)c + 1 \leq c$$

se e solo se

$$1 \leq c - c/3 - (2/5)c = c\Big(\frac{15-5-3}{15}\Big) = c\frac{7}{15}$$

se e solo se

$$c \geq \frac{15}{7}$$

Ricapitolando, se $c \geq 15/7$, allora $T(n) \leq cn$ per ogni $n \geq n_0 = 1$.

Esercizio 4.14. Considerata la ricorrenza

$$T(n) = 10T(\frac{n}{3}) + 3n^2 + n$$

- la si risolva con il metodo del master;
- si dica se $T(n) = O(n^2)$.

Soluzione: In questo caso $\log_b a = \log_3 10 > \log_3 9 = 2$, e $f(n) = 3n^2 + n = O(n^2) = O(n^{\log_b a - \epsilon})$ per un opportuna $\epsilon > 0$ (caso 1 del teorema del master). Allora $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_3 10})$.

$T(n)$ non è $O(n^2)$. Infatti se così fosse avremmo anche $n^{\log_3 10} = O(n^2)$. Ricordo $\log_3 10 > 2$ e quindi $n^{\log_3 10} = n^{2+\alpha}$ per un opportuna $\alpha > 0$. Se fosse $n^{\log_3 10} = O(n^2)$, allora devono esistere $c, n_0 > 0$ tali che $n^{2+\alpha} \leq cn^2$ e quindi $n^\alpha \leq c$ per ogni $n \geq n_0$. Il che è impossibile.

Esercizio 4.15. Considerata la ricorrenza

$$T(n) = 3T(\frac{n}{2}) + 4n^2\sqrt{n}$$

- la si risolva con il metodo del master;
- si dica se $T(n) = O(n^3)$, giustificando la risposta.

Soluzione:

In questo caso $\log_b a = \log_2 3 < 2$, e $f(n) = 4n^2\sqrt{n} = \Omega(n^2) = \Omega(n^{\log_b a+\epsilon})$ per un opportuna $\epsilon > 0$. Inoltre, $af(n/b) = 3 \cdot 4(\frac{n}{2})^2\sqrt{\frac{n}{2}} = 3 \cdot 4(\frac{n^2}{4})\frac{\sqrt{n}}{\sqrt{2}} = \frac{3}{4\sqrt{2}} \cdot 4n^2\sqrt{n} = \frac{3}{4\sqrt{2}}f(n)$ con $\frac{3}{4\sqrt{2}} < 1$ (caso 3 del teorema del master). Allora $T(n) = \Theta(f(n)) = \Theta(4n^2\sqrt{n})$.

$T(n) = O(n^3)$. Infatti $4n^2\sqrt{n} \leq 4n^3$ per ogni $n \geq 1$ e quindi $4n^2\sqrt{n} = O(n^3)$.

Esercizio 4.16. Considerata la ricorrenza

$$T(n) = 8T(\frac{n}{3}) + 2n^2 \log n$$

- la si risolva con il metodo del master;
- si dica se $T(n) = O(n^3)$, giustificando la risposta.

Esercizio 4.17. Risolvere la ricorrenza

$$T(n) = 3T(\frac{n}{2}) + 4n^2\sqrt{n}$$

applicando il metodo iterativo.

Soluzione:

$$\begin{aligned}
T(n) &= 4n^2\sqrt{n} + 3T(\tfrac{n}{2}) \\
&= 4n^2\sqrt{n} + 3 \cdot 4(\tfrac{n}{2})^2\sqrt{\tfrac{n}{2}} + 9T(\tfrac{n}{4}) \\
&= 4n^2\sqrt{n} + 3 \cdot 4(\tfrac{n}{2})^2\sqrt{\tfrac{n}{2}} + 9 \cdot 4(\tfrac{n}{4})^2\sqrt{\tfrac{n}{4}} + 27T(\tfrac{n}{8}) \\
&= 4n^2\sqrt{n} + 3 \cdot 4(\tfrac{n}{2})^2\sqrt{\tfrac{n}{2}} + 9 \cdot 4(\tfrac{n}{4})^2\sqrt{\tfrac{n}{4}} + 27 \cdot 4(\tfrac{n}{8})^2\sqrt{\tfrac{n}{8}} + 81T(\tfrac{n}{16}) \\
&\vdots \\
&= \sum_{i=0}^{k-1} 3^i \cdot 4 \cdot (\tfrac{n}{2^i})^2\sqrt{\tfrac{n}{2^i}} + 3^kT(\tfrac{n}{2^k})
\end{aligned}$$

Ci fermiamo quando $\frac{n}{2^k} = 1$ e quindi quando $k = \log n$. Allora:

$$T(n) = \sum_{i=0}^{\log n-1} 3^i \cdot 4 \cdot \Big(\frac{n}{2^i}\Big)^2\sqrt{\frac{n}{2^i}} + 3^{\log n}$$

Ora

$$3^i \cdot 4 \cdot \Big(\frac{n}{2^i}\Big)^2\sqrt{\frac{n}{2^i}} = 3^i \cdot 4 \cdot \Big(\frac{n^2}{4^i}\Big)\frac{\sqrt{n}}{\sqrt{2^i}} = 3^i \cdot 4 \cdot \Big(\frac{n^2}{4^i}\Big)\frac{\sqrt{n}}{(\sqrt{2})^i} = 4n^2\sqrt{n}\Big(\frac{3}{4\sqrt{2}}\Big)^i$$

e quindi

$$S(n) = \sum_{i=0}^{\log n-1} 3^i \cdot 4 \cdot \Big(\frac{n}{2^i}\Big)^2 \sqrt{\frac{n}{2^i}} = 4n^2\sqrt{n} \sum_{i=0}^{\log n-1} \Big(\frac{3}{4\sqrt{2}}\Big)^i = 4n^2\sqrt{n}\Big(\frac{1-(\frac{3}{4\sqrt{2}})^{\log n}}{1-\frac{3}{4\sqrt{2}}}\Big)$$

$$1 - \frac{3}{4\sqrt{2}} = \frac{4\sqrt{2}-3}{4\sqrt{2}}$$

mentre,

$$\Big(\frac{3}{4\sqrt{2}}\Big)^{\log n} = \frac{3^{\log n}}{(4\sqrt{2})^{\log n}} = \frac{n^{\log 3}}{n^{\log(4\sqrt{2})}}$$

A questo punto, osserviamo che $4\sqrt{2} = 2^2 \cdot 2^{1/2} = 2^{2+(1/2)}$, quindi $\log(4\sqrt{2}) = 2 + (1/2)$, e $n^{\log(4\sqrt{2})} = n^{2+(1/2)} = n^2\sqrt{n}$.

Ricapitolando

$$S(n) = \frac{16\sqrt{2}}{4\sqrt{2}-3} n^2\sqrt{n}\Big(1 - \frac{n^{\log 3}}{n^2\sqrt{n}}\Big) = \frac{16\sqrt{2}}{4\sqrt{2}-3} n^2\sqrt{n} - \frac{16\sqrt{2}}{4\sqrt{2}-3} n^{\log 3}$$

Infine

$$T(n) = S(n) + 3^{\log n} = \frac{16\sqrt{2}}{4\sqrt{2}-3} n^2\sqrt{n} - \frac{16\sqrt{2}}{4\sqrt{2}-3} n^{\log 3} + n^{\log 3} =$$

$$\frac{16\sqrt{2}}{4\sqrt{2}-3} n^2\sqrt{n} - \Big(\frac{16\sqrt{2}}{4\sqrt{2}-3} - 1\Big) n^{\log 3} = \Theta(n^2\sqrt{n})$$

Esercizio 4.18. Sia $T(1) = 1$. Risolvere le seguenti ricorrenze usando uno dei metodi visti a lezione

1. $T(n) = 2T(n/2) + 5n^3$;
2. $T(9/10n) + 2n$;
3. $16T(n/4) + 8n^2$;
4. $T(n) = 7T(n/3) + 12n^2$;
5. $T(n) = 2T(n/4) + 4\sqrt{n}$;
6. $T(n) = 2T(\sqrt{n}) + \log n$;
7. $T(n) = T(\sqrt{n}) + 1$.

Esercizio 4.19. Rispondere alle seguenti domande:

1. Quale è la complessità dell'algoritmo di ricerca sequenziale in funzione del numero n di elementi?
2. Quale è la complessità nel caso peggiore dell'algoritmo di ricerca binaria in funzione del numero n di elementi?
3. Quanti confronti esegue l'algoritmo di ricerca binaria nel caso migliore?
4. Quanti confronti esegue l'algoritmo di ricerca binaria nel caso medio?

Soluzione:

4. $(n+1)/2$ se l'elemento è presente nell'insieme, n se l'elemento non è presente, dove n è il numero di elementi dell'insieme in input. Assumiamo di cercare l'elemento che si trova in posizione i; per identificare questo elemento l'algoritmo di ricerca binaria esegue esattamente i confronti. Inoltre la probabilità che l'elemento che cerchiamo sia proprio quello in posizione i è pari a $1/n$. La complessità nel caso medio è pari a

$$\sum_{i=1}^{n} \frac{1}{n} \cdot i = \frac{1}{n} \sum_{i=1}^{n} i = \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2}$$

Esercizio 4.20. Per la soluzione di un problema, vi e stato proposto di scegliere fra un algoritmo iterativo che ha complessità $\Theta(n^2)$ e un algoritmo ricorsivo la cui complessità è data dalla seguente ricorrenza

$$T(n) = \begin{cases} aT(n/4) + 1 & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

dove $a \in \mathbb{N}$ maggiore di 1. Quale algoritmo preferite? Motivare la risposta.

Soluzione: Dipende dal valore della costante a usata nella definizione di $T(n)$. Infatti, $a > 1$ implica $\log_b a = \log_4 a > 0$, $f(n) = 1 = O(1) = O(n^{\log_4 a - \epsilon})$ (caso 1 del teorema del Master) e quindi $T(n) = \Theta(n^{\log_4 a})$. Ora se $a < 16$ (e quindi $\log_4 a < 4$) la versione ricorsiva è più efficiente di quella ricorsiva, se $a = 16$ il due algoritmi hanno lo stesso comportamento asintotico, se infine $a > 16$ la versione iterativa è da preferire a quella ricorsiva.

4.2 Strutture Dati Elementari

Esercizio 4.21. Quale delle seguenti affermazioni è vera?

1. Pile e code rappresentano esattamente la stessa struttura dati.
2. In una coda, inserimenti e cancellazioni avvengono allo stesso estremo.
3. In una pila, inserimenti e cancellazioni avvengono allo stesso estremo.
4. Nessuna delle precedenti affermazioni è vera

Esercizio 4.22. Se utilizziamo una rappresentazione collegata (del tipo lista di figli) di un albero, quanto tempo è richiesto per trovare tutti i figli di un dato nodo?

Esercizio 4.23. Quali delle seguenti affermazioni relative alle visite di alberi binari è corretta?

1. Gli algoritmi di visita non utilizzano alcuna struttura dati
2. La visita in ampiezza utilizza una struttura dati coda

3. La visita simmetrica utilizza una struttura dati coda
4. La visita in profondità utilizza una struttura dati coda

Esercizio 4.24. Quale è il tempo richiesto dall'algoritmo di visita in profondità di un albero binario in funzione del numero n di nodi?

Esercizio 4.25. Dato un albero generale rappresentato con una struttura collegata del tipo primo figlio/fratello, scrivere due algoritmi per calcolare il numero totale di nodi e il numero totale di foglie dellalbero.

Soluzione: Forniamo due versioni dell'algoritmo che restituisce il numero di nodi di un albero generale, una ricorsiva e una iterativa.

```
NumNodiREC( nodo r)
1  if r = NULL
2    then return 0
3  s ← son[r]
4  nodi ← 1
5  while s ≠ NULL
6     do nodi ← nodi + NumNodiREC(s)
7        s ← bother[s]
```

```
NumNodiITER( nodo r)
 1   Coda C
 2  C.enqueue(r)
 3  nodi ← 0
 4  while  not C.isempty()
 5     do u ← C.dequeue
 6        nodi ← nodi + 1s ← son[r]
 7        nodi ← 1
 8        while s ≠ NULL
 9           do C.enqueue(s)
10              s ← bother[s]
11  return nodi
```

Esercizio 4.26. Sia dato un array A di n valori numerici. Scrivere un algoritmo che verifica se tale array è un max-heap oppure no.

Esercizio 4.27. È possibile stampare in ordine crescente gli elementi di un albero binario di ricerca in tempo $O(n)$? Se sì descrivere un algoritmo, altrimenti spiegare perchè non è possibile.

Esercizio 4.28. Mostrare un albero binario di ricerca di altezza minima contenente le chiavi x, 13, 11, 22, 19, 33, 44, 5, 7, 9, dove x corrisponde al numero rappresentato dalle ultime due cifre del vostro numero di matricola, o tale numero più uno se già presente. Mostrare la sequenza che corrisponde ad una visita pre-ordine dell'albero.

Esercizio 4.29. Mostrare un albero rosso-nero contenente le chiavi x, 13, 11, 22, 19, 33, 44, 5, 7, 9, dove x corrisponde al numero rappresentato dalle ultime due cifre del vostro numero di matricola, o tale numero più uno se già presente. Mostrare, infine, l'albero ottenuto rimuovendo prima la chiave 13, e poi la chiave 22.

Esercizio 4.30. Si consideri un albero binario quasi completo T con n nodi. Quanti nodi ci sono nel sottoalbero sinistro di T? Motivare la risposta fornita.

Esercizio 4.31. Si consideri una tabella hash di dimensione $m = 12$ inizialmente vuota. Si mostri il contenuto della tabella dopo aver inserito la seguente sequenza di valori 34, 12, 36, 19, 16, 24, 17, 27, 25, 15, 87, 109. Si assuma che le collisioni vengono gestite mediante indirizzamento aperto utilizzando come funzione hash $h(k, i) = (h_1(k) + i) \mod m$ dove la funzione hash ordinaria $h_1(k) = k \mod m$.

k	$h_1(k)$	posizioni scandite
34	10	10
12	0	0
36	0	0, 1
19	7	7
16	4	4
24	0	0, 1, 2
17	5	5
27	3	3
25	1	1, 2, 3, 4, 5, 6
15	3	3, 4, 5, 6, 7 , 8
87	3	3, 4, 5, 6, 7 , 9
109	1	1, 2, 3, 4, 5, 6, 7 , 9, 10, 11

4.3 Algoritmi di ordinamento

Esercizio 4.32. Esibire una sequenza di 10 interi che determinano il caso peggiore per l'algoritmo QuickSort; esibire una sequenza della stessa lunghezza che ne determina il caso migliore.

Esercizio 4.33. Si dimostri che il limite inferiore al numero di confronti necessario per ordinare un insieme di n è $\Omega(n \log)$.

Esercizio 4.34. Eseguire il MergeSort sull'input 7, 6, 5, 3, 2, 0, 8, 9, 4, 1.

Esercizio 4.35. Eseguire il SelectionSort sull'input 7, 6, 5, 3, 2, 0, 8, 9, 4, 1.

4.4 Miscellanea

Esercizio 4.36. Sia A un array di n valori reali e x un valore reale. Scrivere un algoritmo, la cui complessità nel caso peggiore è $O(n \log n)$, che restituisca vero solo se esistono due elementi in A la cui somma è x.

Soluzione:

Innanzitutto ordiniamo gli elementi in A (usando un algoritmo di ordinamento ottimale). Poi, per ogni elemento $A[i]$, con $i = 1, \cdots, n$, cerchiamo in A (usando l'algoritmo di ricerca binaria) un elemento pari a $x - A[i]$.

```
SEARCH(A, x)
1  SORT(A)
2  n ← length[A]
3  for i ← 1 to n
4     do if BINARYSEARCH(A, x − A[i])
5           then return true
6  return false
```

Esercizio 4.37. La distanza tra due numeri interi x e y è definita come il valore assoluto della differenza tra x e y; in altri termini $d(x, y) = |x - y|$. Sia A un array di n numeri interi positivi e distinti. Scrivere un algoritmo per determinare i due elementi distinti di A aventi distanza minima. Si discuta la correttezza e si calcoli la complessità.

Esercizio 4.38. Si considerino un albero binario di ricerca T ed un array ordinato A ciascuno di n elementi. Proporre un algoritmo che produca in output un array ordinato B che contenga gli elementi di T ed A.

Soluzione: È una variante della procedura MERGE usata dall'algoritmo MERGESORT. In questo caso dobbiamo confrontare il più piccolo elemento di A con il più piccolo elemento nell'albero T

```
MERGE(A, T)
 1  n ← length[A]
 2  i ← 1
 3  j ← 1
 4  while T ≠ NIL and i ≤ n
 5     do x ← TREE-MINIMUM(T)
 6        if key[x] < A[i]
 7           then B[j] ← key[x]
 8                j ← j + 1
 9                TREE-DELETE(T, x)
10           else B[j] ← A[i]
11                j ← j + 1
12                i ← i + 1
```

```
13  while i ≤ n
14      do B[j] ← A[i]
15         j ← j + 1
16         i ← i + 1
17  while T ≠ NIL
18      do x ← TREE-MINIMUM(T)
19         B[j] ← key[x]
20         j ← j + 1
21         TREE-DELETE(T, x)
```

Esercizio 4.39. Sia $a_1 < a_2 < \cdots < a_n$ una sequenza ordinata di n numeri distinti. Un indice $i = 1, \cdots, n$ è detto *speciale* quando $a_i = i$. Proporre un algoritmo che risolve il problema della ricerca di un indice speciale in un tempo $O(\log n)$.

Soluzione: L'algoritmo che proponiamo è una variante dell'algoritmo di ricerca binaria basato sulle seguenti osservazioni:

1. Sia $i > 1$; allora $a_i < i$ (e quindi $a_i \leq i - 1$) implica $a_{i-1} < a_i \leq i - 1$. Procedendo a ritroso ($a_{i-1} < i - 1$ implica $a_{i-2} < i - 2$, $a_{i-2} < i - 2$ implica $a_{i-3} < i - 3$, e così via) possiamo dimostrare che $a_i < i$ implica $a_j < j - 1$ per ogni $j = 1, \cdots, i - 1$.
2. Viceversa, $a_i > i$ (e quindi $a_i \geq i+1$) implica $a_{i+1} > a_i \geq i+1$. In questo caso abbiamo che $a_i > i$ implica $a_j > j + 1$ per ogni $j = i + 1, \cdots, n$.

La procedura SPECIALINDEX proposta di seguito cerca l'indice speciale nella posizione mediana m. se $A[m] = m$ allora termina restituendo m. Se $A[m] < m$ (per quanto detto sopra) cerca l'indice speciale nelle posizioni successive ad m, altrimenti lo cerca in quelle precedenti.

```
SPECIAL(A, i, j)
1  if i ≤ j
2     then m ← ⌊(i + j)/2⌋
3          if a_m = m
4             then return m
5          if a_m < m
6             then return SPECIAL(A, m + 1, j)
7             else  return SPECIAL(A, i, m − 1)
```

Esercizio 4.40. Scrivere un algoritmo ricorsivo che, data una sequenza di $n \geq 1$ numeri positivi contenuta in un array $A[1..n]$, utilizzi la tecnica divide-et-impera per determinare se la sequenza contiene un numero pari di 2.

Esercizio 4.41. Sia dato un array A di n valori numerici. Scrivere un algoritmo che verifica se tale array è un max-heap oppure no.

Esercizio 4.42. Date le due seguenti procedure A e B, si determini la complessità della procedura $A(n)$ su input $n \in \mathbb{N}$.

```
A(n)
1  s ← 0
2  for i ← 1 to n
3      do s ← s + B(i)
4  return s
```

```
B(n)
1  s ← 0
2  for i ← 1 to n
3      do s ← s + i
4  return s
```

Esercizio 4.43. Valutare la complessità del seguente algoritmo in funzione del parametro n

```
A(n)
1  s ← 0
2  for j ← 1 to n
3      do i ← n
4          while i ≥ j
5              do s ← s + i
6                 i ← i − 1
7  return s
```

Esercizio 4.44. Calcolare la complessità nel caso peggiore della seguente funzione

```
FUN(n)
1  if n ≤ 1
2     then return 3
3  for i ← 1 to n
4      do x ← x + 5
5  return FUN(⌊n/2⌋) + x
```

Soluzione: Se $n \leq 1$, FUN termina restituendo 3; altrimenti esegue il ciclo for di riga 4 (il cui costo è pari ad n) e poi restituisce FUN$(\lfloor n/2 \rfloor)+x$. La ricorrenza che descrive la complessità di questo algoritmo è quindi $T(n) = T(\lfloor n/2 \rfloor) + n$ se $n > 1$, mentre $n \leq 1$ implica $T(n) = c$. Sicuramente $T(n) \leq T'(n)$ dove

$$T'(n) = \begin{cases} T'(n/2) + n & \text{se } n > 1 \\ c & \text{altrimenti} \end{cases}$$

Risolvete $T'(n)$ applicando il metodo iterativo.