Algoritmi e strutture dati: esercizi per la preparazione all'esame

1) Qual è un modo possibile di definire il caso medio di *InsertionSort*?
   A) Ipotizzare che il ciclo interno non si esegua mai.
   B) Ipotizzare che il ciclo interno si esegua sempre.
   C) Ipotizzare che il ciclo interno si esegua sempre, e per la metá dei passi possibili.
   D) Nessuna delle precedenti.

Ragionamento.

2) Diciamo che A è un array di interi è **quasi ordinato** se ogni elemento si trova, al massimo, a k posizioni di distanza dalla sua posizione corretta (quella che assumerebbe nell'array ordinato). Qual è la complessitá di *InsertionSort* quando A è quasi ordinato e k è una costante?
   A) $\Theta(n)$.
   B) $\Theta(k)$.
   C) $\Theta(n^2)$.
   D) $\Theta(k \cdot log(k))$.

Ragionamento.

3) Qual è la complessitá, nel caso peggiore, di *InsertionSort* quando A è quasi ordinato (come definito nell'esercizio 2) e k è $\frac{n}{2}$?

```
proc Parity (A, Key)
{return RecParity(A, Key, 1, A.length)

proc RecParity (A, Key, i, j)
if (i = j)
  then
  if (A[i] = Key)
    then return false
  return true
k = (i+j)/2
if (RecParity(A, Key, i, k) = RecParity(A, Key, k + 1, j))
  then return true
return false
```

Figure 1. Codice per gli esercizi 5 e 6.

   A) $\Theta(n)$.
   B) $\Theta(k)$.
   C) $\Theta(n^2)$.
   D) $\Theta(k \cdot log(k))$.

4) Diciamo che un algoritmo di ordinamento è **stabile** se e solo se, a fine esecuzione, gli elementi con lo stesso valore non hanno cambiato posizione relativa, ed che è **in place** se e solo se la quantità di memoria utilizzata esternamente all'array di partenza è costante. *InsertionSort* è:
   A) Stabile, ma non in place.
   B) In place, ma non stabile.
   C) Stabile ed in place.
   D) Nè stabile, nè in place.

5) Si consideri un array A che contiene chiavi qualsiasi. Si consideri il problema di stabilire se il numero di occorrenze di una certa chiave Key è o no pari (0 si considera pari), ed il codice in Fig. 1. É vero che:
   A) La procedura è sbagliata ed è possibile fornire un controesempio.
   B) La procedura è corretta; un'invariante induttiva di *RecParity* è: dopo ogni chiamata ricorsiva su A, k ha il valore del numero di occorrenze di Key in A.
   C) La procedura è corretta; un'invariante induttiva di *RecParity* è: al termine di ogni chiamata ricorsiva su A viene restituita la paritá del numero di occorrenze di Key in A.
   D) La procedura è sbagliata, ma si puó correggere cambiando l'ultima istruzione.

6) Si consideri un array A che contiene chiavi qualsiasi. Si consideri il problema di stabilire se il numero di occorrenze di una certa chiave Key è o no pari (0 si considera pari). É vero che, se n è la cardinalitá dell'array, la ricorrenza che determina la complessità dell'algoritmo

il cui pseudo codice si trova in Fig. 1 è:

A) $T(n) = T(\frac{n}{2}) + 1$.
B) $T(n) = 2 \cdot T(\frac{n}{2}) + 1$.
C) $T(n) = T(\frac{n}{2}) + \Theta(n)$.
D) $T(n) = 2 \cdot T(\frac{n}{2}) + \Theta(n)$.

7) Considerare l'algoritmo *InsertionSort* e costruirne una versione ricorsiva. Quale sarebbe la ricorrenza che ne determina la complessitá?

A) $T(n) = 2 \cdot T(n-1) + 1$.
B) $T(n) = 2 \cdot T(n-1) + n$
C) $T(n) = T(n-1) + 1$.
D) $T(n) = T(n-1) + n$.

Pseudo codice.

```
def RecInsSort(array, x):
        min = array[x]
        min_i = x
        for i in range(x,len(array)):
                if array[i] < min:
                        min = array[i]
                        min_i = i
        temp = array[x]
        array[x] = min
        array[min_i] = temp
        if (x != len(array)-1):
                RecInsSort(array,x+1)
```

8) Quali sono, tra quelle elencate, proprietá che la relazione $O()$ soddisfa?

A) É simmetrica.
B) É transitiva.
C) É transitiva e riflessiva.
D) É transitiva, riflessiva, e simmetrica.

Dimostrazione.

9) Quali sono, tra quelle elencate, proprietá che la relazione $\Theta()$ soddisfa?

A) É simmetrica.
B) É transitiva.
C) É transitiva e riflessiva.
D) É transitiva, riflessiva, e simmetrica.

10) Quali sono, tra quelle elencate, proprietá che la relazione $o()$ soddisfa?

A) É simmetrica.
B) É transitiva.
C) É transitiva e riflessiva.
D) É transitiva, riflessiva, e simmetrica.

11) Quale delle seguenti è una affermazione vera?

A) Le notazioni $\Theta(), O(), \Omega()$ sono transitive.
B) La notazione $O()$ è simmetrica.
C) Se $f(n) = O(g(n))$ allora $g(n) = \Omega(f(n))$, ma non è vero il contrario.
D) Sono tutte vere.

12) Si consideri la funzione $f(n) = 3 \cdot n^2 + 5 \cdot n + 1$, ed il problema di mostrare che $f(n) = \Theta(n^2)$. Quale tra le seguenti affermazioni istanzia correttamente la definizione?

A) Se $n > 1$, allora $\frac{5}{4} \cdot n^2 \leq f(n) \leq \frac{21}{7} \cdot f(n)$.
B) Se $n > 2$, allora $\frac{3}{5} \cdot n^2 \leq f(n) \leq \frac{21}{5} \cdot f(n)$.
C) Se $n > 4$, allora $\frac{3}{4} \cdot n^2 \leq f(n) \leq \frac{21}{4} \cdot f(n)$.
D) Se $n > 3$, allora $\frac{3}{5} \cdot n^2 \leq f(n) \leq \frac{21}{5} \cdot f(n)$.

Dimostrazione.
risolvo per tentativi

parto dalla disequazione
per poi risolvere il sistema.

c1°n^2 <= f(n) <= c2n^2

ora risolvo il sistema e controllo che
le soluzioni siano comprese nell' intervallo
n > 4 ovvero [5, +inf]

13) È vero che $6 \cdot n^3 = \Theta(n^2)$?

A) No. Infatti, per contraddizione, si assuma $6 \cdot n^3 = \Theta(n^2)$. Tra le altre cose questo implica che per qualche n_0, c_2 si ha che per ogni $n \geq n_0$ è il caso che $6 \cdot n^3 \leq c_2 \cdot n^2$. Questo significa che $6 \cdot n \leq c_2$, il che è contraddice il fatto che c_2 è una costante.
B) No. Infatti, per contraddizione, si assuma $6 \cdot n^3 = \Theta(n^2)$. Tra le altre cose questo implica che per qualche n_0, c_1 si ha che per ogni $n \geq n_0$ è il caso che $6 \cdot n^3 \geq c_1 \cdot n^2$. Questo significa che $6 \cdot n \geq c_1$, il che è contraddice il fatto che c_1 è una costante.
C) Si. Infatti, $6 \cdot n^3$ si comporta asintoticamente come n^2.
D) Si.

14) Sappiamo che $3 \cdot n^2 + 2 = O(n^2)$. Per quali costanti n_0, c vale che $3 \cdot n^2 + 2 \leq c \cdot n^2$ per ogni $n > n_0$?

A) $3 \cdot n^2 + 2 = O(n^2)$ è falso.
B) $n_0 = 10, c = 4$.
C) $n_0 = 1, c = 1$.
D) É vero per tutti gli $n \geq 0$ e tutte le costanti c.

15) É vero che $5 \cdot n + 6 = O(n^2)$?

A) No.

B) Si. Questa affermazione è vera per $c = 1$ e per tutti gli $n > 0$.
C) Si. Questa affermazione è vera per $c = 1$ e per tutti gli $n > 2$.
D) Si. Questa affermazione è vera per $c = 1$ e per tutti gli $n \geq 6$.

16) Nella notazione $O(), \Omega(), \Theta()$:
A) La base dei logaritmi è importante in tutti i casi, ad esempio $\Theta(log_2(n)) \neq \Theta(log_3(n))$.
B) La base dei logaritmi è importante ma solo nella notazione $O()$. Si ha infatti, ad esempio, che $O(log_2(n)) \neq O(log_3(n))$.
C) La base dei logaritmi non è importante. Si ha infatti, ad esempuio, che $\Theta(log_a(n)) = \Theta(log_b(n))$ per qualsiasi $a \neq b$, con $a, b > 1$, e questo vale anche per le altre due notazioni.
D) La base dei logaritmi è importante ma solo nella notazione $\Omega()$. Si ha infatti, ad esempio, che $\Omega(log_2(n)) \neq \Omega(log_3(n))$.

17) Si trovi l'andamento asintotico della funzione $log(n!)$. Succede che:
A) $log(n!) = O(n)$.
B) $log(n!) = O(\sqrt{n})$.
C) $log(n!) = \Theta(n \cdot log(n))$.
D) Nessuna è corretta.

Dimostrazione.

18) Dato un array A di interi non ordinato, vogliamo trovare il massimo ed il minimo di A. Il problema ha costo asintotico lineare cioè $O(n)$, perchè l'algoritmo naïve che lo risolve (trova il minimo con una passata su A, poi trova il massimo con una seconda passata su A) ha costo $O(2 \cdot n) = O(n)$. Tralasciando però in questo esercizio la notazione asintotica, è possibile trovare un algoritmo che risolva questo problema con un numero di **confronti** (non operazioni qualsiasi) strettamente inferiore a $2 \cdot n$?
A) Si. É possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{4}{5} \cdot n$ confronti.
B) Si. É possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{1}{2} \cdot n$ confronti.
C) Si. É possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{3}{4} \cdot n$ confronti.
D) Si. É possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{3}{2} \cdot n$ confronti.

Pseudo codice.

19) Considerata una matrice quadrata di interi A, di lato n, tale che ogni riga è ordinata da sinistra a destra, ed ogni colonna è ordinata dall'alto verso il basso. É possibile mostrare che il problema della ricerca di un certo intero k in A ha complessità, in tempo:
A) $\Omega(n \cdot log(n))$.
B) $\Omega(n^2)$.
C) $\Theta(n \cdot log(n))$.
D) $O(n)$.

Idea.

20) Si consideri la ricorrenza:

$$T(n) = 4 \cdot T(\frac{n}{2}) + 3 \cdot n^2 - 17.$$

Quale delle seguenti è una soluzione?
A) $\Theta(n^2 \cdot lg(n))$.
B) $\Theta(n)$.
C) $\Theta(n^3)$.
D) Questa ricorrenza è irrisolvibile.

21) Si consideri la ricorrenza $T(n) = 2 \cdot T(\frac{n}{2}) + 1$. Usando lo sviluppo, si ottiene:
A) $T(n) = \Theta(n^2)$.
B) $T(n) = \Theta(log(n))$.
C) $T(n) = \Theta(n)$.
D) Nessuna è corretta.

22) Si consideri la ricorrenza $T(n) = 2 \cdot T(\frac{n}{2}) + n^3$. Usando lo sviluppo, si ottiene:
A) $T(n) = \Theta(n^{\frac{3}{4}})$.
B) $T(n) = \Theta(n^2)$.
C) $T(n) = \Theta(n^3)$.
D) $T(n) = \Theta(n^2 \cdot log(n))$.

Dimostrazione.

23) Si consideri la ricorrenza $T(n) = T(\frac{2 \cdot n}{3}) + 1$. È vero che:
A) Possiamo usare il caso 2 del Master Theorem.
B) Possiamo usare il caso 1 del Master Theorem.
C) Possiamo usare il caso 3 del Master Theorem.
D) Potremmo usare il caso 3 del Master Theorem, ma la seconda condizione non è rispettata.

24) Si consideri la ricorrenza $T(n) = 9 \cdot T(\frac{n}{3}) + n$. Usando lo sviluppo, si ottiene:
A) $T(n) = \Theta(n)$.
B) $T(n) = \Theta(n^2 \cdot log(n))$.
C) $T(n) = \Theta(n)$.
D) $T(n) = \Theta(n^2)$.

Dimostrazione.

25) Si consideri la seguente dimostrazione per sostituzione della ricorrenza $T(n) = 2 \cdot T(\frac{n}{2}) + 1$. Vogliamo mostrare che $T(n) \leq c \cdot log(n)$ per qualche c, e lo facciamo con i seguenti passaggi:

$$\begin{aligned} T(n) &\leq 2 \cdot c \cdot log(\tfrac{n}{2}) + 1 \\ &\leq 2 \cdot c \cdot log(n) - 2 \cdot c + 1 \\ &\leq c \cdot log(n) \end{aligned}$$

Possiamo dire che:
A) La dimostrazione è corretta.
B) La dimostrazione è sbagliata.
C) La dimostrazione è corretta, e $T(n) = O(log(n))$.
D) La dimostrazione è sbagliata, ma $T(n) = O(log(n))$.

26) Nel testo usato a lezione (il 'Cormen') appare un errore nell'esercizio 4.3-8. Si chiede di considerare la ricorrenza $T(n) = 4 \cdot T(\frac{n}{2}) + n^2$, e di mostrare che il metodo della sostituzione fallisce cercando di dimostrare che $T(n) \leq c \cdot n^2$. Qual è l'errore nell'esercizio?
A) Non c'è errore: la sostituzione non funziona con $T(n) \leq c \cdot n^2$ ma funziona con $T(n) \leq c \cdot n^2 - d$.
B) Con il Master Theorem vediamo che, diversamente da quanto indicato, $T(n) = \Theta(n^2 \cdot log(n))$; possiamo mostrare che, mentre assumere $T(n) \leq c \cdot n^2 \cdot log(n)$ non funziona, la sostituzione funziona con $T(n) \leq c \cdot n^2 \cdot log(n) - n$.
C) Con il Master Theorem vediamo che, diversamente da quanto indicato, $T(n) = \Theta(n \cdot log(n))$; possiamo mostrare che, mentre assumere $T(n) \leq c \cdot n \cdot log(n)$ non funziona, la sostituzione funziona con $T(n) \leq c \cdot n \cdot log(n) - n$.
D) Nessuna delle altre è corretta.

27) Si consideri la ricorrenza $T(n) = 5 \cdot T(\frac{n}{3}) + n^2$. É vero che:
A) $T(n) = \Theta(n \cdot log^2(n))$.
B) $T(n) = \Theta(n \cdot log(n))$.
C) $T(n) = O(n^2)$.
D) $T(n) = O(n)$.

Dimostrazione.

28) Si consideri la seguente dimostrazione per sostituzione della ricorrenza $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1$. Vogliamo mostrare che $T(n) \leq c \cdot n - d$ per qualche c, d:

$$\begin{aligned} T(n) &\leq (c \cdot \lfloor \tfrac{n}{2} \rfloor - d) + (c \cdot \lceil \tfrac{n}{2} \rceil - d) + 1 \\ &\leq c \cdot n - 2 \cdot d + 1 \\ &\leq c \cdot n - d \qquad \text{se } d \geq 1 \end{aligned}$$

Possiamo dire che:
A) La dimostrazione è errata perchè $T(n) = o(n)$.
B) La dimostrazione è corretta e $T(n) = O(n^2)$
C) La dimostrazione è errata, ma è vero che $T(n) = O(n)$.
D) La dimostrazione è corretta e $T(n) = O(n)$.

29) Consideriamo le ricorrenze $T(n) = 7 \cdot T(\frac{n}{2}) + n^2$ e $T'(n) = a \cdot T'(\frac{n}{4}) + n^2$. Ci domandiamo: qual è il più alto valore intero di a per il quale l'algoritmo che ha complessità $T'(n)$ è asintoticamente più veloce di quello con complessità $T(n)$?
   A) 46.
   B) 47.
   C) 48.
   D) 149.

Dimostrazione.

30) Qual è la complessitá di *SelectionSort*?
   A) $\Theta(n^2)$ in tutti i casi.
   B) $\Theta(n^2)$ nel caso peggiore, $\Theta(n)$ nel caso migliore.
   C) $\Theta(n \cdot log(n))$ in tutti i casi.
   D) $\Theta(n^2)$ nel caso peggiore, $\Theta(n)$ nel caso medio.

31) Si supponga che A sia un array di interi. Un **massimo relativo** di A è una posizione t tra 1 ed n, dove n è la lunghezza di A, tale che $A[t-1] \leq A[t]$ (se $t > 1$) e che $A[t] \geq A[t+1]$ (se $t < n$). Ad esempio, se $A = [2, 3, 8, 5, 4, 9, 10, 7]$, sia la posizione 3 che la posizione 7 sono massimi relativi. Qual è la complessitá del problema di trovare un massimo relativo di A?
   A) $\Theta(n \cdot lg(n))$, perchè possiamo ordinare A e poi, con una ricerca lineare, trovare un massimo relativo.
   B) $\Theta(n)$, perchè possiamo utilizzare direttamente una semplice ricerca lineare.
   C) $O(lg(n))$, perchè possiamo utilizzare una variante della ricerca binaria.
   D) $\Theta(n^2)$.

32) Il cosiddetto h-indice per autori di articoli scientifici è definito come segue: se un autore ha contribuito ad almeno n articoli diversi, ognuno citato almeno n volte, allora il suo h-indice è n. Supponiamo che un array A, contenga, in ogni posizione i, il numero di citazioni ricevute dall'articolo i-esimo di un certo autore; per esempio, se $A = [4, 5, 1, 2, 34, 8, 2, 6, 7]$, allora $h = 5$. Qual è la complessità di stabilire l'indice h dell'autore?
   A) $\Omega(n^2)$.
   B) $\Omega(n^2 \cdot lg(n))$.
   C) $O(n \cdot lg(n))$.
   D) $O(lg(n))$.

Pseudo codice.

```
sort (ARRAY)
n_max = ARRAY.length
while ( ARRAY[i] <= n_max)
      i++
for ( k = i to 0 )
      if ( (ARRAY.length-k) >= A[k] )
            break
return k
```

33) Qual è una invariante del ciclo più esterno di *Selection-Sort*?
   A) Dopo la j-esima esecuzione del ciclo più esterno, l'array $A[j+1, \ldots, n]$ contiene gli $n-j$ elementi più piccoli di A ed è ordinato in maniera non decrescente.
   B) Dopo la j-esima esecuzione del ciclo più esterno, l'array $A[1, \ldots, j]$ contiene gli j elementi più piccoli di A ed è ordinato in maniera non crescente.
   C) Dopo la j-esima esecuzione del ciclo più esterno, l'array $A[1, \ldots, j]$ contiene gli j elementi più piccoli di A ed è ordinato in maniera non decrescente.
   D) Dopo la j-esima esecuzione del ciclo più esterno, l'array $A[j, \ldots, n-1]$ contiene gli $n-j$ elementi più piccoli di A ed è ordinato in maniera non decrescente.

34) Nella nostra implementazione, *SelectionSort* è:
   A) Stabile.
   B) In place, ma non necessariamente stabile.
   C) Stabile, ma non necessariamente in place.
   D) Stabile e in place.

35) Nella nostra implementazione, *MergeSort* è un algoritmo di ordinamento stabile?
   A) No, in quanto non è in place.
   B) Non è possibile stabilirlo.
   C) Sí.
   D) No.

36) Chi, tra *MergeSort* e *InsertionSort*, si comporterebbe meglio sul problema di ordinare un array di n posizioni quasi ordinato (come definito nell'esercizio 2), con k costante?
   A) *MergeSort*: chiaramente $\Theta(n \cdot log(n))$ è sempre meglio di $\Theta(n^2)$.
   B) *MergeSort*: chiaramente $\Theta(n \cdot log(n))$ è sempre meglio di $\Theta(k \cdot n)$.
   C) *InsertionSort*: il suo tempo di esecuzione nel caso peggiore è $\Theta(k \cdot n)$, e quando k è costante questo diventa $\Theta(n)$.
   D) È impossibile stabilirlo, se non sperimentalmente.

37) É possibile utilizzare una strategia simile a quella di *MergeSort* per trovare il massimo elemento di un array A non ordinato?

A) No, il problema è risolvibile unicamente utilizzando un approccio iterativo.
B) Si. L'algoritmo risultante ha complessità $\Theta(log(n))$, risultando migliore di quella dell'approccio standard.
C) Si. L'algoritmo risultante ha complessità $\Theta(n)$, risultando uguale a quella dell'approccio standard.
D) Si. L'algoritmo risultante ha complessità $\Theta(n^2)$, risultando peggiore di quella dell'approccio standard.

Pseudo codice.

```
int findMax(int a[ ], int l, int r)
{
    if (r – l == 1)
        return a[ l ];

    int m = ( l + r ) / 2;
    int u = findMax(a, l, m);
    int v = findMax(a, m, r);

    if (u > v)
        return u;
    else
        return v;
}
```

38) Sia A un array di interi, e chiamiamo **inversione** una coppia di indici $i < j$ tale che $A[i] > A[j]$. Il problema di calcolare il numero di inversioni di A:
A) É irrisolvibile.
B) Ha complessità $\Omega(n^2)$.
C) Ha complessità $\Theta(n^2)$
D) Ha complessità $O(n \cdot log(n))$.

Idea.

risolvibile anche tramite RBT

39) In una versione alternativa di *MergeSort* facciamo tre chiamate ricorsive a tre sotto-array, e poi utilizziamo una versione di *Merge* che è capace di riordinare in un solo array tre, invece di due, sotto-array già ordinati. Come si comporta, asintoticamente, questa nuova versione rispetto all'orginale?
A) In maniera identica.
B) La nuova versione, nel caso peggiore, ha complessità $T(n) = \Theta(n \cdot log^3(n))$, pertanto si comporta in modo peggiore.
C) La nuova versione, nel caso peggiore, ha complessità $T(n) = \Theta(n \cdot log_3(n))$, pertanto si comporta in modo migliore.
D) È impossibile stabilirlo, se non sperimentalmente.

40) Applicare *Partition*$(A, 1, 5)$ all'array $A = [7, 1, 4, 5, 3]$. Qual è il valore restituito (cioè l'indice del pivot)?
A) 2.
B) 1.
C) 3.
D) Dipende dall'esecuzione.

41) La scelta di $A[r]$ come pivot in *Partition* è completamente arbitraria. Cambiarla con qualsiasi altro elemento di A genera un'altra versione di *QuickSort* che ha la stessa complessitá e lo stesso difetto: esiste una istanza di input che certamente provoca il peggior comportamento computazionale.
A) Vero.
B) Falso.
C) Dipende dal fatto che A non è già ordinato in partenza.
D) Non è possibile stabilirlo.

42) Sotto quali condizioni vale l'ultimo passaggio nella dimostrazione relativa al calcolo della complessità del caso medio di *RandomizedQuickSort*?
A) Sempre.
B) Quando a è abbastanza grande, cioè quando $\frac{a \cdot n}{4}$ è asintoticamente più grande di $b + \Theta(n)$
C) Quando a è abbastanza piccolo, cioè quando $\frac{a \cdot n}{4}$ è asintoticamente più piccolo di $b + \Theta(n)$.
D) Esiste una condizione, ma non è nessuna di quelle dette precedentemente.

43) Si consideri la nostra implementazione di *Partition*, ricordando che i valori p ed r sono, rispettivamente, gli indici del primo e dell'ultimo elemento della parte dell'array A in esame. Nel caso in cui A contenga tutti elementi uguali, che valore restituisce *Partition*?
A) x.
B) $r + 1$.
C) r.
D) p.

44) Si consideri *RandomizedQuickSort*, e si consideri una ricorrenza che esprima il numero di chiamate alla funzione *Random* che vengono effettuate nel peggior caso. Quale è, tra queste?
A) $T(n) = T(n-1) + 1$, cioè $\Theta(n^2)$.
B) $T(n) = T(n-1) + 1$, cioè $O(n)$.
C) É impossibile stabilirlo perchè si tratta di una chiamata casuale.
D) $O(log(n))$.

45) Si consideri il problema di trovare il k-esimo elemento piú piccolo di un array A di n posizioni, senza ordinarlo. Usando *Partition*, possiamo trovare un algoritmo di complessità:
A) $\Theta(n^2)$ nel caso migliore.
B) $\Theta(n \cdot log(n))$.
C) $O(n \cdot log(n))$ nel caso peggiore.
D) $O(n)$ nel caso medio.

Pseudo codice.

```
proc MinPartition(int array[], int p, int r,int index)
{

    int q = Partition(array, p, r);

    if(q == index)
       return index;
    else if ( index < q )
       return MinPartition(array, p, q-1,index);
    else
       return MinPartition(array, q+1, r,index);
}
```

Dimostrazione.

Counting sort si basa sulla conoscenza a priori dell' intervallo di valori

46) Si consideri il problema di ordinare istanze di numeri interi positivi diversi da zero. Si danno le seguenti ipotesi: ogni istanza è lunga al massimo 100, le chiavi sono totalmente casuali, ed ogni elemento di ogni istanza è un intero inferiore o uguale a 10^6. Tra le seguenti scelte, qual è la piú efficiente, in media?
   A) Le ipotesi permettono di utilizzare efficientemente *CountingSort*: in media ogni ordinamento prenderá tempo dell'ordine di 100 unitá.
   B) É conveniente usare *QuickSort*: l'ipotesi di istanze casuali ci permette di dire che in media ogni ordinamento prenderá tempo dell'ordine di 700 unitá.
   C) Le ipotesi permettono di dedurre che ogni istanza sará quasi ordinata (come definito nell'esercizio 2): utilizzando *InsertionSort*, impiegheremo, per ogni istanza, un tempo dell'ordine di 100 piú il numero di coppie nell'ordine inverso.
   D) La B) e la A) sono entrambe corrette, ma la scelta A) è piú efficiente.

47) Perchè la complessitá di *QuickSort* è $\Theta(n^2)$ nel caso peggiore?
   A) Nel caso peggiore la partizione è sempre sbilanciata, producendo 2 sotto-problemi di dimensione, rispettivamente, 0 e $n-1$. Questo porta alla ricorrenza:

$$T(n) = T(n-1) + \Theta(n)$$

   che, con il metodo dell induzione, porta a concludere che $T(n) = \Theta(n^2)$.
   B) Nel caso peggiore la partizione è sempre bilanciata, producendo 2 sotto-problemi di dimensione $\frac{n}{2}$. Questo porta alla ricorrenza:

$$T(n) \leq 2 \cdot T(\frac{n}{2}) + \Theta(n)$$

   che, con il metodo della sostituzione, porta a concludere che $T(n) = \Theta(n^2)$.
   C) É sbagliato: *QuickSort* ha complessitá, nel caso peggiore, $O(n \cdot lg(n))$ (perciò si chiama *Quick*sort!).
   D) La complessitá di *QuickSort* dipende direttamente da quella di *Partition*, la quale è $O(n^2)$.

48) Sia A un array di n interi positivi tale che ogni elemento di A è minore o uguale a $5 \cdot n$, e k un intero positivo dispari. Possiamo mostrare che il problema di stabilire se esistono $i, j \geq 1$, $i \neq j$, tali che $A[i] + A[j] = k$ ha complessitá:
   A) $\Theta(n^2)$.
   B) $\Theta(n \cdot log(n))$.
   C) $O(n)$.
   D) $O(log(n))$.

Pseudo codice.

```
// prime 3 righe COUNTING_SORT
let C[0,...,k] new array;
for ( j = 0 to k ) C[ i ] = 0;
for ( j = 1 to A.length ) C[ A[i] ] = C[ A[i] ]++;

for ( i = 1 to k ){
      if ( C[i] != 0  && C[k-i] != 0 )
            trovato ++;
}

essendo k è dispari non può mai essere che
tale che i=( k - j ) sia uguale a j
```

49) É possibile, in linea di principio, ordinare 6 elementi distinti usando non più di 10 confronti, in assenza di ipotesi aggiuntive?
   A) Sí. Il valore $lg(6!) = 9.49$, approssimato a 10, è un limite minimo al numero di confronti necessari per ordinare 6 elementi distinti in assenza di ipotesi aggiuntive.
   B) Sí. Infatti, il numero di confronti eseguiti da *MergeSort* su 10 elementi è, al massimo, 9.
   C) No. Il valore $6! = 720$ è un limite minimo al numero di confronti necessari per ordinare 6 elementi distinti in assenza di ipotesi aggiuntive.
   D) No. Il valore $6 \cdot lg(6) = 15.48$, approssimato a 16, è un limite minimo al numero di confronti necessari per ordinare 6 elementi distinti in assenza di ipotesi aggiuntive.

50) Si consideri la sequenza 4-2-3, 6-2-1, 7-0-3, 1-4-4, 9-4-2, dove le cifre a sinistra sono più significative. Dopo la

seconda esecuzione del ciclo **for** più esterno di *RadixSort*, qual è la terza tripla nell'ordinamento?

A) 7-0-3.
B) 9-4-2.
C) 6-2-1.
D) 4-2-3.

51) Si supponga di voler risolvere il problema dell'ordinamento di un array A si n elementi interi, positivi e negativi, qualsiasi, sotto la seguente ipotesi: le chiavi positive hanno valore $O(n)$, e ci sono al massimo un numero costante k di chiavi negative. Questo problema ha complessità, in tempo:

A) $\Omega(k^{lg(n)})$.
B) $\Omega(n \cdot lg(n))$.
C) $\Theta(n)$.
D) $\Theta(lg(n))$.

Pseudo codice.

52) Se nel codice di *CountingSort* invertiamo l'ordine dell'ultimo ciclo:

A) L'algoritmo diventa scorretto.
B) L'algoritmo non è piú stabile.
C) L'algoritmo rimane invariato.
D) Nessuna delle precedenti.

Dimostrazione.

Durante l' ultimo ciclo si guardano gli elem. di A[]
dall' ultimo al primo.

Scopro un elemento x ipotizzato di pari valore
ad un altro elemento y, già scoperto e posizionato in B.

Dato che stiamo scorrendo A da Dx verso Sx, la posizione
relativa di x rispetto a y in A è alla sua sx.
[| |x| | | y| |]

La correttezza della posizione relativa di x rispetto a y in B è
garantita dall' ultima istruzione (C[A[j]] = C[A[j]] - 1)
la quale indica che x verrà posizionato esattamente nella
casella precedente a y.
Tale ragionamento non regge se si inverte l' ordine del ciclo for,
infatti x verrà posizionato alla destra di y invertendo la loro
posizione relativa.

53) Si supponga che $S = [3, 12, 5, 1, ?, ?, ?, ?]$ sia uno stack con $S.max = 8$. Allora è vero che:

A) $S.top = 3$; dopo *Push*$(S, 6)$ il risultato sará $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.top = 4$, e dopo *Pop*(S)=6, e *Pop*(S)=1 avremo $S = [3, 12, 5, ?, ?, ?, ?, ?]$ con $S.top = 2$.
B) $S.top = 4$; dopo *Push*$(S, 6)$ il risultato sará $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.top = 5$, e dopo *Pop*(S)=6, e *Pop*(S)=1 avremo $S = [3, 12, 5, ?, ?, ?, ?, ?]$ con $S.top = 3$.
C) $S.top = 4$; dopo *Push*$(S, 6)$ il risultato sará $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.top = 5$, e dopo *Pop*(S)=1, e *Pop*(S)=6 avremo $S = [3, 12, 5, ?, ?, ?, ?, ?]$ con $S.top = 3$.
D) $S.top = 4$; dopo *Push*$(S, 6)$ il risultato sará $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.top = 5$, e dopo *Pop*(S)=3, e *Pop*(S)=12 avremo $S = [?, ?, 5, 1, 6, ?, ?, ?]$ con $S.top = 3$.

54) L'idea di implementare una coda su un array in maniera circolare è utile perchè?

A) In realtá implementando una coda su un array in maniera lineare si otterrebbero le stesse prestazioni.
B) Perchè quest'idea consente di risparmiare spazio in memoria, considerato che occupano solo le posizioni che hanno correntemente un valore.
C) Perchè quest'idea consente di risparmiare tempo, considerato che l'implementazione lineare richiederebbe $\Omega(n)$ per ogni operazione di *Dequeue*.
D) Perchè quest'idea consente di assicurare la correttezza di *Eniqueue* e *Dequeue*: l'implementazione lineare potrebbe essere scorretta.

55) Si consideri una lista singolarmente collegata, della quale conosciamo l'indirizzo della testa $L.head$. Qual è la complessitá del problema di stabilire se la lista ha un loop (cioè se l'ultimo elemento punta ad un elemento della lista giá visto)?

A) $O(n)$.
B) $\Theta(n \cdot log(n))$.
C) $\Omega(n \cdot log(n))$
D) $O(1)$.

Dimostrazione.

56) Qual è il costo di invertire una lista singolarmente collegata?

A) $O(n)$.
B) $O(1)$.
C) $\Theta(n^2)$.

D) Non è possibile invertire una lista singolarmente collegata.

57) Si consideri un'implementazione qualsiasi di una pila, che mette a disposizione le operazioni *Push*,*Pop*, ed *Empty*. Utilizzando esclusivamente queste operazioni e variabili di tipo pila, è possibile implementare una struttura dati di tipo coda?
A) Si, è possibile. L'operazione di *Enqueue* avrebbe ancora complessità $O(1)$, ma l'operazione di *Dequeue* passerebbe ad avere complessità $O(n)$.
B) Si, è possibile. Le operazioni di *Enqueue* e *Dequeue* così ottenute passerebbero ad avere complessità $\Theta(n)$.
C) Si, è possibile. L'operazione di *Empty* avrebbe però complessità $\Theta(n)$.
D) Non è possibile.

58) È l'array $A = [23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$ una max-heap?
A) Sí.
B) No.
C) No, perchè la proprietá delle max-heap non è rispettata per nessuna terna padre-destro-sinistro.
D) No, perchè la proprietá delle max-heap non è rispettata per almeno una terna padre-destro-sinistro.

59) Considera le chiavi 1,2,3,4. Quante max-heap diverse si possono costruire con queste chiavi?
A) 3.
B) 4.
C) 2.
D) 1.

60) Si consideri l'array $A = [16, 4, 10, 14, 7, 9, 3, 2, 8, 1]$. Quale delle seguenti affermazioni è vera?
A) A puó essere una max-heap. Inoltre, applicando *MaxHeapify*$(A, 2)$, l'algoritmo termina alla terza chiamata ricorsiva, e restituisce una permutazione degli elementi di A che la rende una min-heap.
B) A non puó essere una max-heap. Inoltre, applicando *MaxHeapify*$(A, 2)$, l'algoritmo termina alla prima chiamata ricorsiva, e restituisce una permutazione degli elementi di A che la rende una max-heap.
C) A non è una max-heap. Inoltre, applicando *MaxHeapify*$(A, 2)$, l'algoritmo termina alla terza chiamata ricorsiva, e restituisce una permutazione degli elementi di A che la rende una max-heap.
D) A non è una max-heap. Inoltre, applicando *MaxHeapify*$(A, 2)$, l'algoritmo fallisce perchè viene chiamato su una struttura che non rispetta le precondizioni richieste.

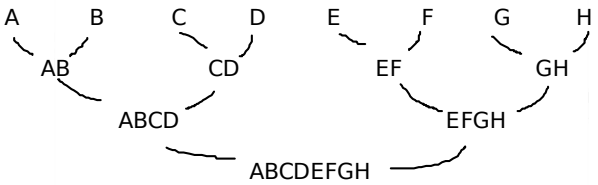
61) É possibile che la min-heap H sia di altezza 3 e contenga, nell'ordine, esattamente i seguenti elementi: $1, 5, 7, 9, 12, 10, 8$?
A) No, perchè se H è di altezza 3 deve contenere almeno 8 elementi.
B) No, perchè l'ordine non rispetta la proprietá.
C) No, perchè l'albero corrispondente non sarebbe completo.
D) Si.

62) Si consideri l'array $A = [7, 1, 14, 2, 10, 20]$, ed si applichi *BuildMaxHeap* su A. Quanti elementi di A sono rimasti nella stessa posizione prima e dopo l'esecuzione?
A) 1.
B) 2.
C) Nessuno.
D) 3.

Dimostrazione.

63) Si considerino k array di numeri interi $A_1, \ldots, A_k$, tutti ordinati, tali che il numero di totale di elementi degli array è n. Qual è la complessità del problema di produrre un array B che contiene tutti gli n numeri ordinati?
A) $O(k \cdot log(k))$.
B) $O(n \cdot log(k))$.
C) $O(k \cdot log(n))$.
D) $O(n \cdot log(n))$.

Dimostrazione.



K = 8

Riprendo da Merge. Ad ogni livello dell' albero effettuo esattamente n confronti. I livelli sono log(k)
-> =O(n log(k))

64) Si consideri una min-heap. É noto che l'estrazione del minimo costa $O(log(n))$, dove n è il numero di elementi della heap. Quanto costa l'estrazione del massimo da una min-heap?
A) $\Theta(n^2)$.
B) $O(log(n))$, applicando lo stesso algoritmo per l'estrazione del minimo.
C) $O(log(n))$.
D) $O(n)$.

65) Supponiamo che $H = [9, 7, 2, 1, 4, 1, 1]$ sia una max-heap, e supponiamo di chiamare *IncreaseKey*$(H, 6, 6)$. Qual è il risultato?
A) $H = [9, 7, 1, 6, 2, 4, 1]$.
B) $H = [9, 1, 6, 1, 4, 7, 2]$.
C) $H = [9, 6, 7, 1, 4, 2, 1]$.
D) $H = [9, 7, 6, 1, 4, 2, 1]$.

66) In una max-heap A:
A) L'ultimo livello è sempre completo.
B) Gli elementi sono ordinati in modo non crescente.
C) L'elemento piú piccolo è $A[1]$.
D) Nessuna è corretta.

67) Riguardo alla complessità di *BuildMaxHeap*:
A) Utilizzando il fatto che in una heap di n elementi al massimo $\lceil \frac{n}{2^{h+1}} \rceil$ di questi si trovano ad altezza h, si ottiene una complessita di $O(n)$.
B) Utilizzando l'analisi aggregata, possiamo mostrare che la complessitá è $O(n)$.
C) Utilizzando il fatto che in una heap di n elementi al massimo $\lceil \frac{n}{2^{h+1}} \rceil$ di questi si trovano ad altezza h, si ottiene una complessita di $O(lg(n))$.
D) La procedura esegue $O(n)$ volte *MaxHeapify*, pertanto la sua complessità è $O(n \cdot lg(n))$.

68) Si consideri il problema di trovare il k-esimo elemento piú piccolo di un array A di n posizioni, senza ordinarlo. Usando una max-heap possiamo trovare un algoritmo di complessità:
A) $O(log(n)^2)$.
B) $O(k + (n - k) \cdot log(k))$ nel caso peggiore.
C) $O(log(n))$ nel caso peggiore.
D) Non si può usare una max-heap per risolvere questo problema.

Dimostrazione.

69) Si consideri il problema di trovare il k-esimo elemento piú piccolo di un array A di n posizioni, senza ordinarlo. Usando una min-heap, possiamo trovare un algoritmo di complessità:
A) $\Theta(n)$.
B) $O(n + k \cdot log(n))$ nel caso peggiore.
C) $O(k \cdot log(n))$ nel caso peggiore.
D) Non si può usare una min-heap per risolvere questo problema.

Dimostrazione.

70) Supponiamo che T sia una tabella hash con conflitti risolti via chaining attraverso liste, e che la dimensione di T sia 9 (dalla posizione 1 alla posizione 9 entrambe comprese). Supponiamo che $h(k) = k \bmod 9 + 1$. In T inizialmente vuota si inseriscono le chiavi $5, 28, 19, 15, 20, 33, 12, 17, 10$. Qual é, alla fine di tutti gli inserimenti, lo stato della lista puntata dalla posizione 2, letta da sinistra a destra?
A) 28,10,19.
B) 10,28.
C) 28,10,15.
D) 10,19,28.

-- Facile --

71) Supponiamo di dover memorizzare 10000 chiavi intere diverse, e che abbiamo a disposizione fino a 500 slot $(1 \leq m \leq 500)$ in una tabella hash con conflitti risolti via chaining. Volendo usare il metodo del modulo, quale, tra le seguenti, è la migliore scelta per m?
A) 317.
B) 255.
C) 257.
D) 500.

72) Supponiamo di avere una tabella hash con conflitti risolti via chaining, $m = 7$ (posizioni dalla 1 alla 7 comprese), e di usare il metodo della moltiplicazione con $A = 0.5$ per inserire nella tabella, inizialmente vuota, i seguenti valori nell'ordine: $0.37, 12, 124, 3.47, 2.56, 41$. Quale posizione della tabella avrà, alla fine degli inserimenti, la lista piú lunga?
A) La 1.
B) La 4.
C) La 7.
D) La 1 e la 2.

73) In una tabella hash con indirizzamento aperto, l'hashing è detto uniforme se:
A) La funzione h è sempre di tipo modulo.
B) La sequenza di probing garantisce che tutte le posizioni sono considerate, prima o poi.
C) La funzione h inserisce una chiave k nella posizione k.

D) Il probing è quadratico.

74) Supponiamo di avere una tabella hash con indirizzamento aperto, $m = 7$ (posizioni dalla 1 alla 7 comprese), e di usare il metodo del probing lineare per inserire nella tabella, inizialmente vuota, i seguenti valori nell'ordine: $12, 5, 8, 19$. Quale posizione della tabella occuperà l'elemento a chiave 19, se $h'(k) = (k \ mod \ m) + 1$?
   A) La 1.
   B) La 4.
   C) La 7.
   D) Nessuna delle precedenti.

75) Sia A un array di n interi positivi tale che ogni elemento di A è minore o uguale a n^3, e k un intero positivo dispari. Possiamo trovare un algoritmo che permette di stabilire se esistono $i, j \geq 1$, $i \neq j$, tali che $A[i] + A[j] = k$ che ha complessitá, nel caso medio:
   A) $\Theta(n^2)$.
   B) $\Theta(n \cdot log(n))$.
   C) $O(n)$.
   D) $O(log(n))$.

Dimostrazione.

alloco un array (hashTable -> H) di n3 posizioni in modo da poter contenere tutti i possibili elementi.
Scorro A con i ed effettuo H[A[i]] = true

```
for (i=1;i<n;i++)
        if ( H[ k - H[A[i]] ] == TRUE )
                // trovato
                H[ k - H[A[i]] ] = FALSE
                // passo a false così la coppia
                // non viene contata 2 volte
```

76) Supponiamo di avere una tabella hash con conflitti risolti via chaining, $m = 100$ (posizioni dalla 1 alla 100 comprese), e di poter usufruire di una funzione di hashing uniforme semplice. Dopo 4 inserimenti, qual è la probabilità che le prime 4 posizioni della tabella siano rimaste vuote?
   A) Superiori a 0.96.
   B) Tra 0.88 e 0.92.
   C) Tra 0.86 e 0.88.
   D) Meno di 0.85.

77) Sia $h()$ una funzione di hash. Il tempo necessario per effettuare la ricerca di una chiave k in una tabella hash con conflitti risolti via chaining utilizzando liste concatenate, assumendo che sia presente, dipende da:
   A) Il numero di elementi k' tali che $h(k) = h(k')$ che sono stati inseriti dopo k nella tabella.
   B) Il numero di elementi k' tali che $h(k) = h(k')$ che sono stati inseriti prima di k nella tabella.
   C) Il numero di elementi k' tali che $h(k) \neq h(k')$ che sono stati inseriti dopo k nella tabella.
   D) Il numero di elementi k' tali che $h(k) \neq h(k')$ che sono stati inseriti prima di k nella tabella.

78) In una foresta $\mathcal{S}$ di alberi k-ari che rappresentano insiemi disgiunti, supponiamo che $S_1 = \{2, 3, 7\}$, dove $2.p = 2, 3.p = 2$, e $7.p = 3$, e che $S_2 = \{4, 8\}$, dove $4.p = 4$ e $8.p = 4$. Quali sono i minimi ranghi possibli di 2 e di 4, e cosa accade, in questo caso, eseguendo *Union*$(3, 8)$, assumendo di utilizzare l'unione per rango e la compressione del percorso?
   A) $2.rank = 2$, $4.rank = 3$, e il risultato è un solo insieme, radicato in 4, tale che i padri di S_1 sono invariati, $4.p = 8$, e $2.rank = 1$.
   B) $2.rank = 3$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 3, tale che i padri di S_1 sono invariati, $4.p = 3$, e $2.rank = 2$.
   C) $2.rank = 2$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 2, tale che i padri di S_1 sono invariati, $4.p = 2$, e $2.rank = 2$.
   D) Nessuna delle altre risposte è corretta.

79) In una foresta $\mathcal{S}$ di alberi k-ari che rappresentano insiemi disgiunti, supponiamo che $S_1 = \{a, b, f\}$, dove $a.p = f, b.p = f$, e $f.p = f$, e che $S_2 = \{c, d\}$, dove $c.p = d$ e $d.p = d$, e supponiamo che $f.rank = d.rank = 1$. Cosa accade dopo aver eseguito *Union*(c, b), assumendo di utilizzare l'unione per rango, la compressione del percorso, e l'implementazione vista in classe?
   A) Si ottiene un solo insieme con tutti gli elementi elencati, tale che $f.rank = 2$, $d.p = b$.
   B) Si ottiene un solo insieme con tutti gli elementi elencati, tale che $d.rank = 2$, $f.p = d$.
   C) Si ottiene un solo insieme con tutti gli elementi elencati, tale che $f.rank = 2$, $d.p = f$.
   D) Nessuna delle altre risposte è corretta.

80) Nella rappresentazione degli insiemi disgiunti con foreste di alberi k-ari, con unione per rango e compressione del percorso, quale delle seguenti è vera?
   A) La complessità di *MakeSet* é $\Theta(n+m)$, nel caso in cui abbiamo fatto m operazioni di cui n *MakeSet* prima di quella analizzata.
   B) La complessità di *Union* nel caso peggiore è $\Theta(n)$, dove n è il numero di operazioni di *MakeSet* che sono state fatte precedentemente.
   C) La complessità di *FindSet* nel caso peggiore è $O(1)$.
   D) La complessità di *FindSet* nel caso peggiore è $O(h)$, dove h è l'altezza dell'albero su cui l'operazione è chiamata.

81) In una foresta $\mathcal{S}$ di alberi k-ari che rappresentano insiemi disgiunti, supponiamo che $S_1 = \{2, 3, 7\}$, dove $2.p = 2, 3.p = 2$, e $7.p = 3$, e che $S_2 = \{4, 8\}$, dove $4.p = 4$ e $8.p = 4$. Quali sono i minimi ranghi possibli di 2 e di 4, e cosa accade, in questo caso, eseguendo *Union*$(3, 8)$, assumendo di utilizzare l'unione per rango e la compressione del percorso?
   A) $2.rank = 2$, $4.rank = 3$, e il risultato è un solo insieme, radicato in 4, tale che i padri di S_1 sono

!!! findset comprime il percorso solo a partire dal nodo 3 !!

invariati, $4.p = 8$, e $2.rank = 1$.

B) $2.rank = 3$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 3, tale che i padri di S_1 sono invariati, $4.p = 3$, e $2.rank = 2$.

C) $2.rank = 2$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 2, tale che i padri di S_1 sono invariati, $4.p = 2$, e $2.rank = 2$.

D) Nessuna delle altre risposte è corretta.

82) Si consideri la struttura per insiemi disgiunti basata su foreste di alberi che offre le operazioni di *MakeSet*, *Union*, *FindSet*, e *Link*, con unione per rango e compressione del percorso. Si immagini di eseguire le seguenti operazioni, nell'ordine: *MakeSet*(4), *MakeSet*(6), *MakeSet*(2), *MakeSet*(5), *Union*(4,6), *Union*(2,5), *MakeSet*(8), *Union*(2,8), *Union*(8,6). Alla fine delle operazioni, qual è l'effetto di chiamare *FindSet*(2)?

A) Ritornerà il nodo che contiene la chiave 4 senza effettuare nessuna modifica ai puntatori.

B) Ritornerà il nodo che contiene la chiave 5 senza effettuare nessuna modifica ai puntatori.

C) Ritornerà il nodo che contiene la chiave 4, e lo metterà come nuovo padre del nodo che contiene 2, modificando il suo vecchio padre che era il nodo che contiene 8.

D) Ritornerà il nodo che contiene la chiave 6, e lo metterà come nuovo padre del nodo che contiene 2, modificando il suo vecchio padre che era il nodo che contiene 5.

83) Si consideri l'implementazione basata su liste con unione pesata per insiemi disgiunti, e si consideri il risultato dell'analisi ammortizzata per m operazioni di cui n *MakeSet*. Qual è, nel caso peggiore, il costo medio di ogni operazione?

A) $\Theta(1)$.

B) $\Theta(lg(n))$.

C) $\Theta(lg^2(n))$.

D) $\Theta(n)$.

84) In ogni albero binario di ricerca, ogni nodo con due figli è tale che:

A) Il suo successore ha sempre un figlio sinistro.

B) Il suo predecessore ha sempre un figlio destro.

C) Il suo successore non ha mai un figlio sinistro.

D) Tutte le altre risposte sono incorrette.

85) Si consideri un albero binario qualsiasi tale che la sequenza $BAFILGSCHD$ è il risultato della sua visita pre-order, e la sequenza $IFLAGSBHCD$ quello della sua visita in-order. Quanto è alto l'albero?

A) 3.

B) 4.

C) 2.

D) 5.

Ragionamento.

86) Sappiamo che è possibile ricostruire la struttura di un albero binario qualsiasi a partire dal risultato della sua visita in-order e della sua visita pre-order; è possibile farlo anche a partire dal risultato della sua visita in-order e della sua visita post-order. Ma è possibile fare lo stesso a partire dal risultato della sua visita pre-order e della sua visita post-order?

A) Si, e la complessità del problema è la stessa che negli altri due casi.

B) Si, ma la complessità del problema è minore che negli altri due casi.

C) Si, ma la complessità del problema è maggiore che negli altri due casi.

D) No.

Dimostrazione.

87) Quali sono le differenze più rilevanti tra gli alberi utilizzati nelle strutture per insiemi disgiunti (implementazione con foreste di alberi k-ari) e quelli utilizzati come struttura dati per inserimento, cancellazione, visite, e operazioni simili?

A) Gli alberi utilizzati come base per insiemi disgiunti sono sempre k-ari, mentre quelli utilizzati come struttura dati classica sono sempre binari.

B) Non c'è alcuna differenza.

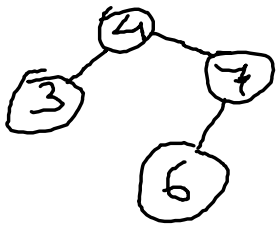
C) Gli alberi utilizzati come base per insiemi disgiunti non hanno puntatori ai figli; quindi, non possono essere visitati, nè la loro struttura può essere mantenuta da operazioni di inserimento o cancellazione, che non hanno senso.

D) Gli alberi utilizzati come base per insiemi disgiunti in realtà sono array visiti come alberi.

88) È possibile avere un albero binario di ricerca con le chiavi $\{1, 4, 5, 10, 16, 17, 21\}$ e di altezza 6? E di altezza 2? In questo ultimo caso, quale sarebbe la chiave della radice?
A) Si puó avere di altezza 2, ma non di altezza 6.
B) Si puó avere di altezza 6, ma non di altezza 2.
C) Sí in entrambi i casi; per avere un albero binario di ricerca di altezza 2 esiste una sola soluzione e la chiave della radice è 5.
D) Sí in entrambi i casi; per avere un albero binario di ricerca di altezza 2 esiste una sola soluzione e la chiave della radice è 10.

89) Considerando la nostra implementazione di *BST-TreeDelete*, è vero che l'operazione di eliminazione di una chiave da un albero binario di ricerca è commutativa? È vero, cioé, che se si eliminano dallo stesso albero 2 nodi diversi in un particolare ordine, si ottiene esattamente lo stesso risultato che eliminandoli nell'ordine inverso?
A) No, in generale non è vero.
B) No. Un controesempio è dato dall'eliminazione di due nodi foglia.
C) È impossibile stabilirlo.
D) Sì.

Dimostrazione.



eliminare i nodi 4 e 3 in diverso ordine porta ad un diverso albero risultante.
Tale

90) Considerando l'albero binario di ricerca in Fig. 2, e immaginando una versione di *BSTTreeDelete* nella quale si utilizzi il predecessore immediato, invece del successore immediato, dopo aver eliminato la chiave 14, l'oggetto risultante:
A) Ha altezza 2, la chiave della radice è 9, e il figlio sinistro del nodo che contiene 7 contiene la chiave 3.
B) Ha altezza 4, la chiave della radice è 7, e il figlio sinistro del nodo che contiene 9 contiene la chiave 3.
C) Ha altezza 3, la chiave della radice è 5, e il figlio sinistro del nodo che contiene 7 contiene la chiave 9.
D) Ha altezza 3, la chiave della radice è 5, e il figlio sinistro del nodo che contiene 9 contiene la chiave 7.

91) Si supponga che T sia un albero binario di ricerca con chiavi intere, e che A sia un array di interi ordinato, e di voler stampare la successione di elementi di $T \cup A$ in ordine non decrescente. Si consideri il codice in Fig. 3. Qual è una istruzione corretta da inserire al posto di *?
A) $(x \neq \textbf{nil})$ **and** $(A[i] \leq A[A.length])$.
B) $(x = \textbf{nil})$ **and** $(i \leq A.length)$.
C) $(x \neq \textbf{nil})$ **or** $(i \leq A.length)$.
D) $(x \neq \textbf{nil})$ **and** $(i \leq A.length)$.

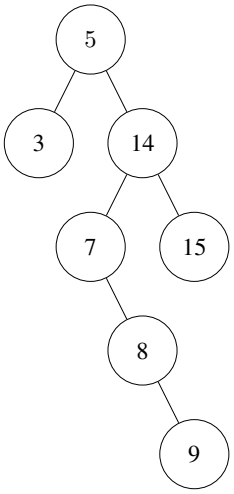


Figure 2. Codice per l'esercizio 90.

```
proc MergeTreeArray (T, A)
  x = TreeMinimum(T.root)
  i = 1
  while (*)
    if (x.key ≤ A[i])
      then
        Print(x.key)
        x = TreeSuccessor(x)
      else
        Print(A[i])
        i = i + 1
  while (x ≠ nil)
    Print(x.key)
    x = TreeSuccessor(x)
  while (i ≤ A.length)
    Print(A[i])
    i = i + 1
```

Figure 3. Codice per l'esercizio 91.

92) Si consideri due alberi binari di ricerca T, T', tali che ogni chiave di T è minore di ogni chiave di T'. Si consideri il problema di costruire un albero binario di ricerca T'' che contiene tutte e sole le chiavi di $T \cup T'$, ed il codice in Fig. 4. É vero che:
A) La procedura è corretta.
B) La procedura è incorretta.
C) La procedura è corretta, ma sotto l'ipotesi che T e T' siano bilanciati.
D) La procedura è corretta, ma sotto l'ipotesi che T e T' siano completi.

93) Si consideri un albero binario di ricerca T, ed un nodo $x \in T$. Supponiamo di cambiare la chiave $x.key$ con una chiave piú grande, ed il problema di stabilire se T' risultante è ancora un albero binario di ricerca. Si

```
proc BSTUnion (T, T')
  T'' = ∅
  x = TreeMinimum(T')
  TreeDelete(T', x)
  T''.root = x
  x.left = T.root
  x.right = T'.root
  x.p = nil
  return T''
```

Figure 4. Codice per l'esercizio 92.

qui x è gia cambiata

```
proc StillBST (T, x)
  y = TreeSuccessor(T, x)
  if (y = nil)
    then return true
  if (*)
    then return true
  return false
```

Figure 5. Codice per gli esercizio 93.

consideri il codice in Fig. 5. Al posto di * possiamo inserire:

A) $x.key \geq y.key$.
B) $x.key = y.key$.
C) $x.key \leq y.key$.
D) $x.key \neq y.key$.

94) Si consideri un albero binario di ricerca T, ed un nodo x in T. Qual è la complessita del problema di stabilire se, dopo aver cambiato la chiave $x.key$ di un certo nodo x con una nuova chiave piú grande, T è ancora un albero binario di ricerca valido?

A) $\Theta(n^2)$.
B) $O(1)$.
C) $O(n)$.
D) $\Omega(n^2)$.

la nuova chiave di x deve essere maggiore del predecessore di x e minore del suo successore. Tree_Successor ha complessità O(h) che nel caso pessimo (albero altamente sbilanciato -> (lista)) diventa è pari a O(n)

95) Si supponga che T sia un albero qualsiasi con chiavi intere, che Cl sia una variabile globale condivisa da tutte le chiamate di *PLS* e inizializzata a zero, il cui codice è mostrato in Fig. 6, e si supponga di chiamare *PLS(x.root,1)*. Qual è l'effetto?

A) Quello di stampare tutte e sole le chiavi che sono presenti sulla "frontiera" di T, cioè tutte e sole quelle che si trovano su nodi foglia.
B) Quello di stampare tutte e sole le chiavi che sono presenti sul "ramo di mezzo" di T, cioè tutte e sole quelle che si trovano su nodi che hanno altri nodi sia a destra che a sinistra sullo stesso livello.
C) Quello di stampare tutte e sole le chiavi che sono presenti sul "costato sinistro" di T, cioè tutte e sole quelle che si trovano su nodi non nascosti da nodi allo stesso livello e piú a sinistra.
D) Quello di stampare tutte e sole le chiavi che sono presenti sui "rami dispari' di T, cioè tutte e sole quelle che si trovano su nodi che appartengono a rami con un indice dispari contando da uno, da sinistra verso destra, sulla foglia del ramo stesso.

```
proc PLS (x, Ml)
  if (Cl < Ml)
    then
      Print(x.key)
      Cl = Ml
  if (x.left ≠ nil)
    then PLS(x.left, Ml + 1)
  if (x.right ≠ nil)
    then PLS(x.right, Ml + 1)
```

Figure 6. Codice per l'esercizio 95.

96) Si assuma che T sia un albero binario qualsiasi con chiavi intere. Si consideri il problema di stabilire se T è un BST. Quale tra le seguenti tecniche funziona correttamente?

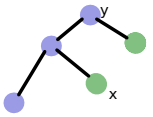
A) Consideriamo la chiave della radice. Ad ogni passo, deve essere vero che l'elemento considerato non contiene una chiave maggiore di quella contenuta in uno dei suoi figli. Se questa propietá è vera per tutti gli elementi, restituiamo TRUE. Si noti che la tecnica funziona perchè è basata sulle chiavi. stupida
B) Consideriamo il nodo che dovrebbe contenere il minimo di T. Ad ogni passo, deve essere vero che l'elemento considerato non contiene una chiave maggiore dell'elemento che viene restituito da *BSTTreeSuccessor*. Se questa proprietá è vera per tutti gli elementi, restituiamo TRUE. Si noti che la tecnica funziona perchè sia *BSTTreeMinimum* che *BSTTreeSuccessor* sono procedure basate sulla struttura dell'albero, non sulle chiavi. non basta
C) Consideriamo il nodo che dovrebbe contenere il massimo di T. Ad ogni passo, deve essere vero che l'elemento considerato contiene una chiave maggiore dell'elemento che viene restituito da *BSTTreeSuccessor*. Se questa proprietá è vera per tutti gli elementi, restituiamo TRUE. Si noti che la tecnica funziona perchè *BSTTreeSuccessor* è una procedura basata sulla struttura dell'albero, non sulle chiavi. non basta
D) Nessuna delle tecniche presentate funziona correttamente.

97) Qual è la complessità del problema di trasformare un albero binario qualsiasi quasi completo in una min-heap?

A) $\Theta(n \cdot log(n))$.
B) $O(1)$.
C) $O(n)$.
D) Tutti gli alberi binari quasi completi sono min-heap.

98) Si consideri un BST T ed una chiave k tale che *BSTTreeSearch*(T, k) ritorna x dove x è una foglia. Si consideri adesso tre insiemi A, B, C tali che A contiene tutte e sole le chiavi che sono lasciate a sinistra del percorso di ricerca, B contiene tutte e sole le chiavi sul percorso di ricerca, e C contiene tutte e sole le chiavi che sono lasciate a destra del percorso di ricerca. É vero che:

A) $\forall k_1, k_2, k_3((k_1 \in A \wedge k_2 \in B \wedge k_3 \in C) \rightarrow k_1 \leq k_2 \leq k_3)$;
B) $\forall k_1, k_2, k_3((k_1 \in A \wedge k_2 \in B \wedge k_3 \in C) \rightarrow k_1 < k_2 < k_3)$;



CONTROESEMPIO

x.key < y.key anche se x in C e y in B

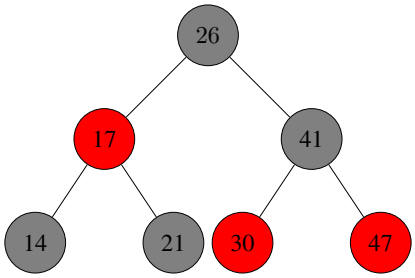


Figure 7. Albero per gli esercizi 100 e 102.

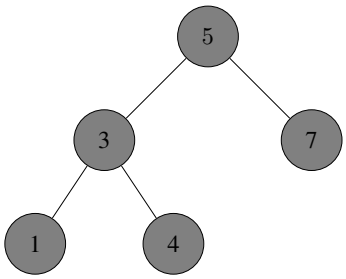


Figure 8. Albero binario di ricerca per l'esercizio 101.

C) $\forall k_1, k_2, k_3((k_1 \in A \wedge k_2 \in B \wedge k_3 \in C) \rightarrow k_1 \geq k_2 \geq k_3)$;
D) Nessuna è corretta.

99) Consideriamo un albero binario con nodi senza chiavi (cioè, solo la struttura dell'albero), con n nodi. Fissate n chiavi, in quanti modi diversi possiamo etichettare i nodi con le chiavi in maniera che l'albero risultante sia un BST regolare?
A) 1.
B) 2.
C) $n!$.
D) $O(n)$.

100) Si consideri il codice di *BSTTreeLeftRotate* visto in classe si applichi sul nodo x che contiene la chiave 26 nel albero RBT in Fig. 7. Qual è l'altezza dell'albero risultante? Qual è la chiave del figlio sinistro della radice?
A) L'altezza è 2, e il figlio sinistro della radice è **nil**.
B) L'altezza è 3, e il figlio sinistro della radice ha chiave 47.
C) L'altezza è 2, e il figlio sinistro della radice ha chiave 41.
D) L'altezza è 3, e il figlio sinistro della radice ha chiave 26.

101) Supponiamo di applicare *BSTTreeRightRotate* sull'albero binario di ricerca in Fig. 8, centrato nel nodo che contiene la chiave 5. Quali sono, dopo la rotazione, i figli del nodo che contiene la chiave 5?
A) Figlio sinistro: 1; figlio destro: 4.
B) Figlio sinistro: 3; figlio destro: 7.
C) Figlio sinistro: 1; figlio destro: 7.
D) Figlio sinistro: 4; figlio destro: 7.

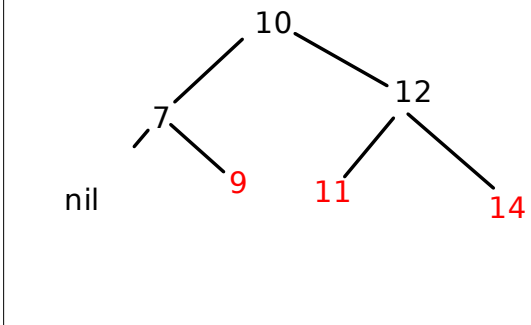
102) Consideriamo l'albero RBT di Fig. 7, e le regole viste nella teoria:
1. Ogni nodo é rosso o nero;
2. La radice è nera;
3. Ogni foglia (esterna, **nil**) è nera;
4. Se un nodo è rosso, entrambi i suoi figli sono neri;
5. Per ogni nodo, tutti i percorsi semplici da lui alle sue foglie, contengono lo stesso numero di nodi neri.

Inserendo quale delle seguenti chiavi, in un nodo rosso, violiamo sicuramente la proprietá 4?
A) 7.
B) 31.
C) 18.
D) 23.

103) Si consideri il codice visto in classe per l'inserimento di una nuova chiave in un albero RBT T. Supponiamo adesso che T sia un albero RBT inizialmente vuoto, e di inserire, nell'ordine: 10,7,14,9,11,12. Dopo tutti gli inserimenti, che chiave ha il figlio destro della radice e di che colore è?
A) 12, rosso.
B) 11, rosso.
C) 10, nero.
D) Nessuna è corretta.

12 Nero

Albero risultante.



104) Si consideri il codice visto in classe per l'inserimento di una nuova chiave in un albero RBT T. Supponiamo adesso che T sia un albero RBT inizialmente vuoto, e di inserire, nell'ordine: 15,10,11,8,9. Dopo tutti gli inserimenti, qual è la posizione e qual è il colore del nodo che contiene la chiave 9?
A) Figlio sinistro della radice, nero.
B) Figlio destro della radice, rosso.
C) Figlio sinistro del nodo che contiene la chiave 15, nero.
D) Figlio destro del nodo che contiene la chiave 8, rosso.

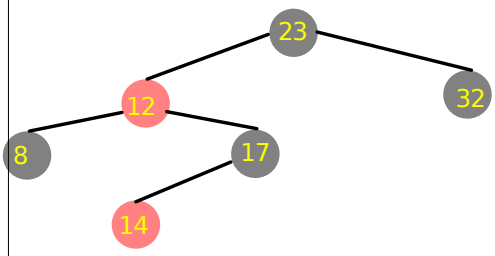
105) Ricordiamo le proprietá di un RBT:
1. Ogni nodo é rosso o nero;
2. La radice è nera;
3. Ogni foglia (esterna, Nil) è nera;
4. Se un nodo è rosso, entrambi i suoi figli sono neri;
5. Per ogni nodo, tutti i percorsi semplici da lui alle sue foglie, contengono lo stesso numero di nodi neri.

Qual è una invariante di albero RBTreeinsert?

A) Se l'albero víola qualche proprietà, allora ne víola esattamente una, che è la 2 o la 4. Se é la 2, è perché z (il nodo inserito) è la radice ed è rossa, se é la 4, è perché z e $z.p$ sono entrambi rossi.
B) Se l'albero víola qualche proprietà, allora ne víola esattamente una, che è la 1 o la 4. Se é la 1, è perché z (il nodo inserito) non è né rosso né nero, se é la 4, è perché z e $z.p$ sono entrambi rossi.
C) Se l'albero víola qualche proprietà, allora ne víola esattamente una, che è la 2 o la 5. Se é la 2, è perché z (il nodo inserito) è la radice ed è rossa, se é la 5, è perché l'altezza nera di qualche ramo è diversa da quella di tutti gli altri.
D) Se l'albero víola qualche proprietà, allora ne víola esattamente una, che è la 2 o la 3. Se é la 2, è perché z (il nodo inserito) è la radice ed è rossa, se é la 3, è perché z non ha foglie esterne.

106) Dopo aver inserito in un RBT inizialmente vuoto le chiavi 32,23,12,8,17,14, nell'ordine, quanti sono i nodi rossi?
A) 1.
B) 2.
C) 3.
D) 4.

Albero finale.



107) Si consideri un albero B con tutti i nodi pieni, $t = 50$, e $h = 2$ Quante chiavi contiene?
A) 100000.
B) 500000.
C) 990000.
D) 999999.

108) Quanti sono i possibili alberi B che si possono costruire con tutte e sole le chiavi $\{1, 2, 3, 4, 5\}$, con grado minimo 3?
A) 1.
B) 2.
C) 3.
D) 4.

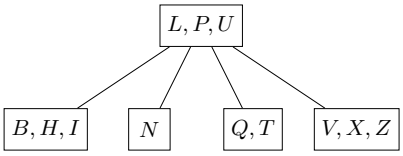


Figure 9. Albero per l'esercizio 110.

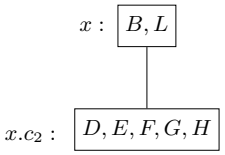


Figure 10. Albero per l'esercizio 111.

Dimostrazione.

109) Qual è la complessità in termini di CPU dell'operazione di ricerca di una chiave in un albero B se, invece della ricerca lineare sul nodo, usiamo la ricerca binaria?
A) la complessitá non cambia.
B) $O(t \cdot log(log(n)))$.
C) $O(log_t(n))$.
D) $O(log(n))$.

110) Sia T l'albero in Fig. 9, con $t = 2$, e nel quale si assume l'ordinamento alfabetico. Si consideri, adesso, le procedure *BTreeInsert* e *BTreeSplitChild*, e *BTreeInsertNonFull*, come viste in classe. Dopo l'inserimento della chiave G, quanti nodi ha in tutto l'albero risultante? Quante chiavi ha la radice nell'albero risultante?
A) L'albero risultante ha 8 nodi, e la radice ha la sola chiave P.
B) L'albero risultante ha 7 nodi, e la radice ha la sola chiave U.
C) L'albero risultante ha 9 nodi, e la radice ha le chiavi P, U.
D) Nessuna delle altre risposte è corretta. Inoltre, se la chiave inserita fosse O, invece di G, l'albero risultante avrebbe ancora 5 nodi.

111) Sia T l'albero parzialmente mostrato in Fig. 10. Dopo aver eseguito *BTreeSplitChild*$(x, 2)$, e assumendo di utilizzare l'ordinamento alfabetico, succede che:
A) x contiene B, E, L, $x.c_2$ contiene D, F, e $x.c_3$ con-

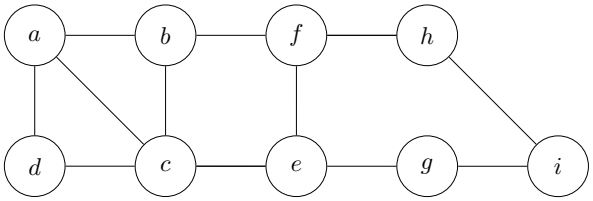


Figure 11. Grafo per gli esercizi 112 e 113.

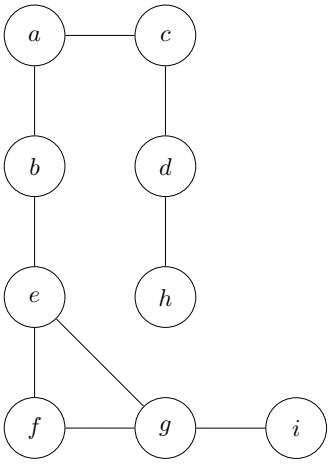


Figure 12. Grafo per gli esercizi 114 e 120.

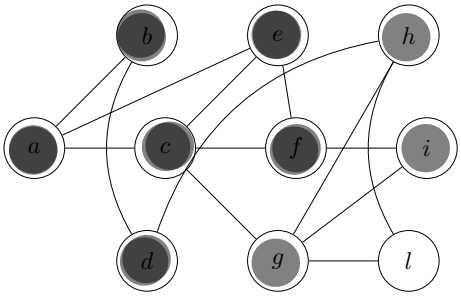


Figure 13. Grafo per l'esercizio 115.

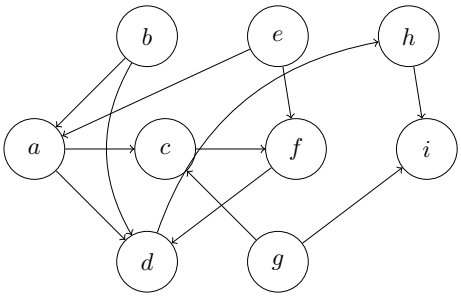


Figure 14. Grafo per gli esercizi 118 e 119.

tiene G, H.

B) x contiene B, F, L, $x.c_2$ contiene D, E, e $x.c_3$ contiene G, H.

C) x contiene F, L, $x.c_2$ contiene B, D, E, e $x.c_3$ contiene G, H.

D) x contiene B, F, H, $x.c_2$ contiene D, E, e $x.c_3$ contiene G, L.

112) Si esegua *BreadthFirstSearch* sul grafo di Fig. 11 assumendo che la sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica, dopo la visita, a cosa punta $e.\pi$:

A) c.
B) f.
C) e.
D) g.

113) Si esegua *BreadthFirstSearch* sul grafo di Fig. 11, assumendo che la sorgente sia c e che i nodi vengano sempre estratti in ordine alfabetico e si dica quale delle seguenti è una configurazione della coda che si verifica durante la visita:

A) $Q = [a, b, f]$.
B) $Q = [c, b, d]$.
C) $Q = [d, e, f]$.
D) $Q = [a, b, e]$.

114) Si esegua *BreadthFirstSearch* sul grafo di Fig. 12, assumendo che la sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica quanti nodi vengono visitati, compresa la sorgente, prima di visitare il nodo h, escluso.

A) 7.
B) 8.
C) 5.
D) 4.

115) Sia G il grafo in Fig. 13. Assumendo che $s = a$, e che i vertici in una lista di adiacenza siano visitati in ordine alfabetico, qual è lo stato della coda Q durante l'esecuzione di *BreadthFirstSearch*(G, s) nel momento in cui f diventa nero?

A) $Q = [g, h, i]$.
B) $Q = \emptyset$.
C) $Q = [b, c, d]$.
D) $Q = [h, i, f, e]$.

116) In un grafo indiretto, diciamo che una **componente connessa** è un sottoinsieme massimale di vertici tutti raggiungibili tra loro ma dai quali non si raggiungono altri vertici del grafo fuori dal sottoinsieme. Si consideri un grafo indiretto, ed il problema di stabilire quali sono le sue componenti connesse. Qual è la complessità del problema di stabilire il suo numero di componenti connesse?

A) $\Theta(|E|^2)$.
B) $\Theta(|V|^2)$.
C) $\Theta(|V|^2 + |E|)$.
D) $\Theta(|V| + |E|)$.

Idea.

117) Si consideri la seguente strategia per risolvere il problema di trovare le minime distanze da una sorgente s di un grafo indiretto *pesato* dove i pesi appartengono all'insieme $\{1, 2\}$. L'algoritmo *BreadthFirstSearch* non puó essere usato direttamente perchè il grafo è pesato; modifichiamo il grafo nel seguente modo: per ogni arco di peso 2 che unisce (u, v) insieriamo un nuovo vertice z, due nuovi archi di peso 1 $(u, z), (z, v)$, e eseguiamo *BreadthFirstSearch* sul nuovo grafo. Gli attributi $u.d$ che vengono calcolati sono la risposta cercata. Chiamando $G = (V, E)$ il grafo originale e $G' = (V', E')$ quello modificato, si ha che:
   A) Questa strategia è corretta. Nel caso peggiore tutti gli archi di G pesano 2, e pertanto ci saranno $|E|$ nuovi vertici; in questo caso però, anche gli archi saranno aumentati, e ci saranno, in tutto $2 \cdot |E|$ archi. Pertanto il costo totale sará $O(|V'| + |E'|) = O((|V| + |E|) + 2 \cdot |E|)$.
   B) Questa strategia è corretta. Nel caso peggiore tutti gli archi di G pesano 2, e pertanto ci saranno $|E|^2$ nuovi vertici; in questo caso però, anche gli archi saranno aumentati, e ci saranno, in tutto $|E|$ archi. Pertanto il costo totale sará $O(|V'| + |E'|) = O((|E|^2) + |E|)$.
   C) Questa strategia è incorretta, ma puó essere corretta inserendo tre nuovi vertici per ogni arco di peso 2.
   D) Tutte le altre risposte sono sbagliate.

118) Si esegua *DepthFirstSearch* sul grafo di Fig. 14, assumendo che la prima sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica, dopo la visita, qual è l'intervallo inizio scoperta - fine scoperta del vertice d.
   A) $[17, 18]$.
   B) $[3, 4]$.
   C) $[4, 12]$
   D) $[4, 9]$.

119) Si esegua *DepthFirstSearch* sul grafo di Fig. 14, assumendo che la prima sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica, dopo la visita, qual è il vertice scoperto per ultimo.
   A) g.
   B) h.
   C) e
   D) b.

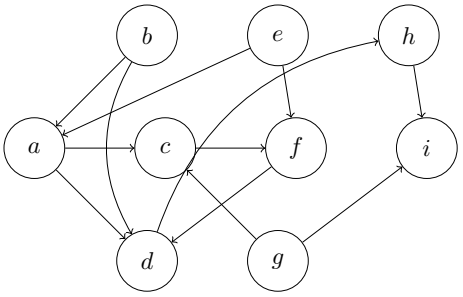


Figure 15. Grafo per l'esercizio 122.

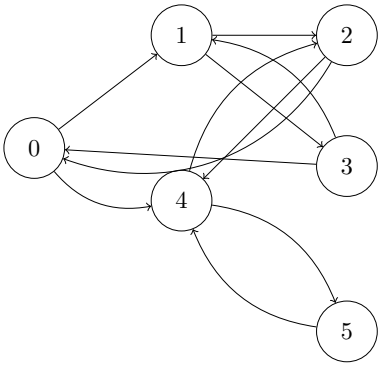


Figure 16. Grafo per l'esercizio 123.

120) Si esegua *DepthFirstSearch* sul grafo di Fig. 12, assumendo che la sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica quanti nodi vengono visitati, compresa la sorgente, prima di visitare il nodo h, escluso.
   A) 7.
   B) 8.
   C) 5.
   D) 4.

121) Dopo l'esecuzione di *DepthFirstSearch*, dati due vertici u, v, supponiamo che l'intervallo $[v.d, v.f]$ sia completamente contenuto nell'intervallo $[u.d, u.f]$. Allora:
   A) v è discendente di u in uno degli alberi della foresta di visita in profonditá.
   B) u è discendente di v in uno degli alberi della foresta di visita in profonditá.
   C) v, u sono incomparabili negli alberi della foresta di visita in profonditá.
   D) u è raggiungibile da v e viceversa.

122) Qual è un ordinamento topologico del grafo in Fig. 15?
   A) $c, f, a, b, d, h, i, g, e$.
   B) $c, e, b, a, d, h, i, g, f$.
   C) $g, e, b, a, c, f, d, h, i$.
   D) $g, e, b, f, d, h, i, c, a$.

123) Qual è un ordinamento topologico del grafo di Fig. 16?

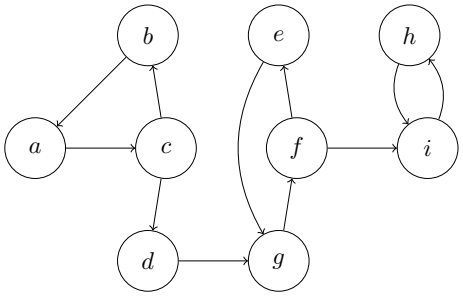


Figure 17. Grafo diretto per l'esercizio 125.

A) $0, 1, 4, 5, 2, 3$.
B) $5, 2, 4, 3, 1, 0$.
C) $0, 1, 2, 5, 4, 3$.
D) Il grafo nella figura non ha un ordinamento topologico.

124) Si consideri l'algoritmo *TopologicalSort* come visto in classe. Questo:
A) Lavora su grafi diretti tanto pesati e non pesati, in assenza di cicli.
B) Lavora su grafi diretti aciclici non pesati. Se lanciato su grafi diretti aciclici pesati, la sua complessitá peggiora.
C) Lavora su grafi diretti aciclici pesati. Se lanciato su grafi diretti aciclici non pesati, la sua complessitá peggiora.
D) Lavora su grafi indirietti aciclici.

Idea.

125) Quante sono le componenti fortemente connesse del grafo in Fig. 17?
A) 1.
B) 2.
C) 3.
D) 4.

126) Sia G un grafo diretto in cui ogni vertice rappresenta un punto di interesse in una cittá, e ogni arco rappresenta una strada, a senso unico, che connette due punti di interesse. Per poter ottimizzare i percorsi turistici della cittá, un sistema basato su intelligenza artificiale ha bisogno di sapere se esistono, e quante sono, zone della cittá dalle quali, una volta usciti, non è possibile rientrare senza violare un senso unico. Quale algoritmo su grafi, tra quelli che abbiamo studiato, puó essere utile a questo scopo?
A) *BreadthFirstSearch*.
B) *CycleDet*.
C) *StronglyConnectedComponents*.
D) Questo problema non può essere risolto.

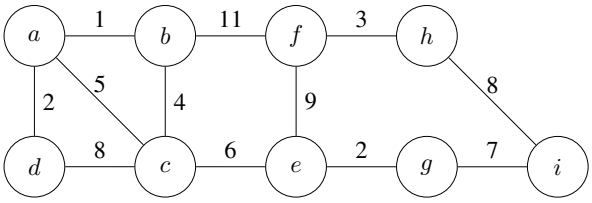


Figure 18. Grafo per gli esercizi 128 e 138.

127) Quale delle seguenti affermazioni è falsa, rispetto a grafi diretti?
A) Eliminando archi, il numero delle componenti fortemente connesse aumenta o rimane uguale.
B) Aggiungendo archi, il numero delle componenti fortemente connesse diminuisce sempre.
C) Il numero delle componenti fortemente connesse è compreso tra 1 e il numero dei vertici.
D) Il grafo delle componenti fortemente connesse puó essere topologicamente ordinato.

Dimostrazione.

128) Utilizzare *MST-Prim* per calcolare il peso di un albero di copertura minimo del grafo in Fig.18?
A) 17.
B) 24.
C) 31.
D) 33.

129) In *MST-Prim*, l'istruzione $v.key = W(u, v)$ è, in realtà, una istruzione di?
A) *IncreaseKey*.
B) *Enqueue*.
C) *Dequeue*.
D) *DecreaseKey*.

130) Sia G il grafo in Fig 19. Quale arco non appare nel MST prodotto da *MST-Prim* se $r = a$ e se, in caso di chiavi

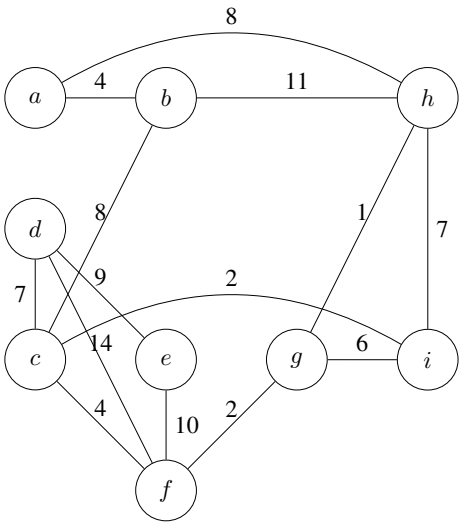


Figure 19. Grafo per l'esercizio 130.

uguali, la coda di priorità utilizza l'ordine alfabetico?

A) (a, h).
B) (c, i).
C) (a, b).
D) (c, d).

131) Supponiamo che G sia un grafo indiretto pesato in cui gli archi possono avere peso negativo. E' possibile utilizzare *MST-Prim* per calcolarne un albero di copertura minimo?

A) No. La dimostrazione di correttezza fallisce in questo caso, e il risultato potrebbe non essere corretto.
B) No. La definizione di albero di copertura minimo perde di senso in caso di archi di peso negativo.
C) Si.
D) Si, ma la complessità risulterebbe asintoticamente peggiore.

132) Qual è la complessità di *MST-Prim* eseguito su grafi sparsi (quindi con $|E| << |V|^2$) e implementando la coda di priorità con un array senza struttura?

A) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la costruzione della coda), $\Theta(|V|^2)$ (per le estrazioni del minimo), e $\Theta(|E|)$ (per i decrementi, analisi ammortizzata, non si può sostituire con $\Theta(|E|)$ sotto queste ipotesi). Il costo totale diventa dunque $\Theta(|V| + |E|)$.
B) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la costruzione della coda), $\Theta(|E|)$ (per le estrazioni del minimo, sotto l'ipotesi di grafi sparsi), e $\Theta(|E|)$ (per i decrementi, analisi ammortizzata). Il costo totale è $\Theta(|E|)$.
C) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|^2)$ (per la costruzione della coda, sotto l'ipotesi di grafo denso), $\Theta(|V|^2)$ (per le estrazioni del minimo), e $\Theta(|E|)$ (per i decrementi, analisi ammortizzata). Il costo totale è $\Theta(|V|^2 + |E|)$.
D) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la costruzione della coda), $\Theta(|V|^2)$ (per le estrazioni del minimo), e $\Theta(|E|)$ (per i decrementi, analisi ammortizzata, non si può sostituire con $\Theta(|E|)$ sotto queste ipotesi). Il costo totale è ancora $\Theta(|V|^2)$, ma con costanti nascoste inferiori al caso di grafi densi.

133) Supponiamo che G sia un grafo indiretto pesato. Possiamo utilizzare *MST-Prim* oppure *MST-Kruskal* per calcolare il peso di un albero di copertura massimo?

A) No. Il problema è ben definito, ma è necessario utilizzare un altro tipo di algoritmo.
B) No. La definizione di albero di copertura massimo è priva di senso.
C) Si. E' sufficiente invertire il senso di tutte le disuguaglianze.
D) Si. *MST-Prim* calcola l'albero di copertura massimo senza che sia necessario alcun cambio; *MST-Kruskal*, invece, no.

134) Si consideri il grafo in Fig.18, ed applicare *MST-Kruskal* per calcolare l'albero di copertura minimo. Quale, tra questi archi, non è certamente parte del risultato, se gli archi di peso uguale sono ordinati lessicograficamente?

A) (e, c).
B) (e, g).
C) (c, d).
D) (h, i).

135) Qual è la complessità di *MST-Kruskal* eseguito su grafi densi (quindi con $|E| = O(|V|^2)$), quando gli insiemi disgiunti sono gestiti con liste e utilizzando l'unione pesata?

A) *MST-Kruskal* non può essere implementato utilizzando insiemi disgiunti su liste.
B) Abbiamo: inizializzazione $(O(1))$, ordinamento: $(\Theta(|E| \cdot log(|E|))$, e $O((|V| + |E|)$ diverse operazioni su insiemi, di cui $O(|V|)$ sono *MakeSet* (per un totale di $\Theta(|V|^2)$. Totale, $\Theta(|E|^2)$. Quindi, su grafi densi, l'implementazione con foreste di alberi ha chiaramente un impatto sulla complessità asintotica.
C) Abbiamo: inizializzazione $(O(1))$, ordinamento: $(\Theta(|E| \cdot log(|E|)) = \Theta(|V|^2 \cdot log(|E|)) = \Theta(|V|^2 \cdot log(|V|))$, e $O((|V| + |E|)$ diverse operazioni su insiemi, di cui $O(|V|)$ sono *MakeSet* (per un totale di $\Theta(|V| \cdot log(|V|))$. Totale, $\Theta(|V|^2 \cdot log(|V|))$. Quindi, su grafi densi, l'implementazione con foreste di alberi non ha un impatto sulla complessità asintotica.
D) Tutte le altre risposte sono sbagliate.

136) Si supponga che G sia un grafo indiretto, pesato, e che tutti i pesi siano interi tra 1 e un numero $O(|E|)$. Possiamo implementare *MST-Kruskal* in modo che, sotto queste ipotesi, la sua complessitá per grafi sparsi sia asintoticamente migliore di quella calcolata senza ipotesi sul grafo?

A) Si. Utilizzando foreste di alberi per implementare insiemi disgiunti, la complessitá puó abbassarsi a $O(log(|E|))$.

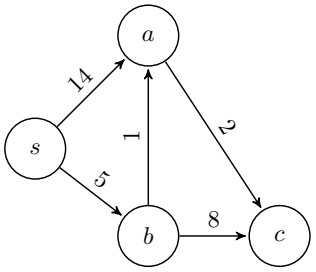


Figure 20. Grafo per gli esercizi 138 e 139.

B) Si. Utilizzando foreste di alberi per implementare insiemi disgiunti, la complessitá puó abbassarsi a $O((|V| + |E|) \cdot \alpha(|V|))$.
C) No. La complessitá non risente di queste ipotesi.
D) Tutte le altre risposte sono sbagliate.

Idea.

137) Si supponga che G sia un grafo indiretto, pesato, e denso. Supponiamo inoltre che G venga fornito di una struttura aggiuntiva (un array di coppie) in cui è inserito ogni arco ed il suo peso, in maniera tale gli archi risultino quasi ordinati (come definito nell'esercizio 2). Sotto queste ipotesi, si vuole realizzare una versione di *MST-Kruskal* con complessità asintoticamente inferiore a quella originale. Possiamo ottenere una complessità di:
A) $\Theta(|V|)$, cambiando l'implementazione dell'operazione di *MakeSet*.
B) $\Theta(|V|)$, cambiando l'implementazione dell'operazione di *Union*.
C) $\Theta(|V|^2 \cdot lg(|V|))$, cambiando l'algoritmo di ordinamento soggiacente.
D) $\Theta(|V|^2)$, cambiando l'algoritmo di ordinamento soggiacente.

138) Si consideri il grafo diretto in Fig. 20. Qual é il valore di $c.\pi$ dopo l'esecuzione di *Bellman-Ford*? E qual é il valore del suo (del padre di c) attributo $.d$, assumendo s come sorgente?
A) $b, 6$.
B) $a, 6$.
C) $c, 0$.
D) s, ∞.

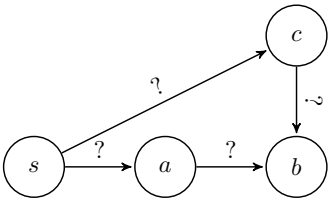


Figure 21. Grafo per l'esercizio 140.

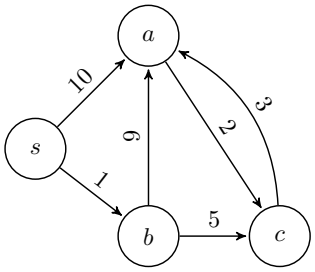


Figure 22. Grafo per l'esercizio 141.

139) Si consideri il grafo diretto in Fig. 19. Nel caso peggiore, quante chiamate a *Relax* devono essere effettuate da *Bellman-Ford* prima che $c.d$ assuma il valore corretto, assumendo s come sorgente? E da *DAGShortestPath*?
A) $8, 4$.
B) $10, 6$.
C) $12, 5$.
D) $21, 7$.

Dimostrazione.

140) Si consideri il grafo diretto in Fig. 21. Come si possono distribuire i pesi $10, 10, 25, -50$ sugli archi in maniera da garantire che *Dijkstra*, eseguito con sorgente s, fallisca?
A) $W_{sc} = -50, W_{cb} = 25, W_{sa} = 10, W_{ab} = 10$.
B) $W_{sc} = 25, W_{cb} = -50, W_{sa} = 10, W_{ab} = 10$.
C) $W_{sc} = 25, W_{cb} = 10, W_{sa} = -50, W_{ab} = 10$.
D) *Dijkstra* fallisce sempre in presenza di un arco negativo.

141) Si consideri il grafo diretto in Fig. 22. Qual é il valore di $a.d$ dopo l'esecuzione di *Dijkstra*?
A) 10.
B) 9.

```
proc GreedyShortestPath (G, w, u, v)
  for (t ∈ G.V)
    { u.π = nil
      u.color = WHITE
  Current = u
  Current.color = BLACK
  while (Current ≠ nil)
    { z = MinWhite(Current)
      if (z ≠ nil)
        then
          { z.π = Current
            z.color = BLACK
      Current = z
```

Figure 23. Codice per l'esercizio 142.

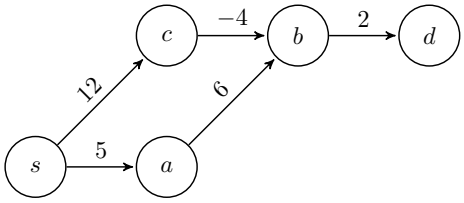


Figure 24. Grafo indiretto per l'esercizio 143.

C) 7.
D) ∞.

142) Si consideri il codice in Fig. 23, eseguito su un grafo diretto pesato, dove la funzione *MinWhite(t)* restituisce, se esiste, il vertice bianco raggiungibile da t con arco di peso minimo. Se G è un grafo diretto pesato con funzione di peso w, e u, v sono due vertici di G, *GreedyShortestPath(G,w,u,v)* dovrebbe trovare un percorso minimo tra u e v, utilizzando il puntatore π per memorizzarlo. Succede che:
   A) L'algoritmo è corretto.
   B) L'algoritmo è scorretto, e fallisce sempre.
   C) L'algoritmo è scorretto, e fallisce in qualche grafo.
   D) L'algoritmo è corretto, ma non è ottimo.

143) Si consideri il grafo in Fig. 24. Succede che:
   A) Applicando *Dijkstra* con sorgente s, il peso $d.\pi$ si calcola correttamente, nonostante la presenza di un ciclo con peso negativo.
   B) Applicando *Dijkstra* con sorgente s, il peso $d.\pi$ si calcola in maniera incorretta, dovuto alla presenza di un arco con peso negativo.
   C) Applicando *Dijkstra* con sorgente s, il peso $d.\pi$ si calcola correttamente, nonostante la presenza di un arco con peso negativo.
   D) Applicando *Dijkstra* con sorgente s, il peso $d.\pi$ si calcola in maniera incorretta, dovuto alla presenza di un ciclo con peso negativo.

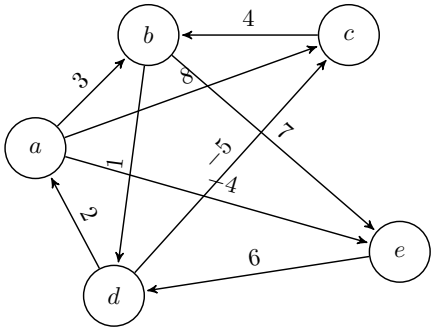


Figure 25. Grafo per gli esercizi 145 e 146.

Dimostrazione.

144) Nell'ambito del problema dei cammini minimi tra tutte le coppie, e dell'algoritmo della moltiplicazione di matrici per la sua soluzione, che cosa rappresenta il valore L_{ij}^m, per una certa coppia di vertici i, j?
   A) Il peso del cammino minimo tra i e j.
   B) Il peso del cammino minimo tra i e j usando almeno m archi.
   C) Il peso del cammino minimo tra i e j usando esattamente m archi.
   D) Il peso del cammino minimo tra i e j usando al più m archi.

145) Si consideri il grafo diretto in Fig. 25. Durante l'esecuzione di *SlowAllPairsMatrix*, qual é il valore di L_{ad}^2?
   A) 2.
   B) ∞.
   C) 13.
   D) Non è definito.

146) Si consideri il grafo diretto in Fig. 25. Durante l'esecuzione di *Floyd-Warshall*, qual é il valore di D_{ec}^3?
   A) 3.
   B) ∞.
   C) 10.
   D) -2.