PRIMA PARTE

1. **[5 punti]** Si definisca come problema computazionale il problema di verificare se il minimo di un array A di interi è minore di un dato intero x.

2. **[6 punti]** Sia T un testo di lunghezza n e P un pattern di lunghezza $m < n$.

   (a) Descrivere tramite pseudocodice l'algoritmo di pattern matching di Knuth, Morris e Pratt (KMPMatch).

   (b) Dimostrare che tale algoritmo ha complessità $O(n)$, assumendo, senza dimostrarlo, che la failure function si calcola in tempo $O(m)$.

3. **[5 punti]** Si consideri uno heap rappresentato da un array $P[1 \div 10]$ contenente 10 entry con chiavi distinte. Quante entry sappiamo con certezza che avranno chiave maggiore di quella di $P[2]$? Motivare la risposta.

SECONDA PARTE

1. **[7 punti]** Sia $S = S[1], S[2], \ldots, S[n]$ una sequenza di n bit dove vengono prima gli 0 e poi gli 1. Il seguente algoritmo restituisce l'indice massimo i per cui $S[i] = 0$ (tale indice è -1 se S contiene tutti 1).

```
if (S[1]=1) then return -1
if (S[n]=0) then return n
i ⟵ 1; j ⟵ n;
while  i < j-1 do {
  k ⟵ ⌊(i+j)/2⌋
  if (S[k]=0) then i ⟵ k else j ⟵ k
}
return i
```

   Si osservi che quando $i < j - 1$ si ha che $i < \lfloor (i + j)/2 \rfloor < j$.

   (a) Trovare un opportuno invariante per il ciclo.

   (b) Supponendo che l'invariante trovato valga alla fine dell'ultima iterazione, dimostrare che l'algoritmo è corretto.

2. **[8 punti]** Si consideri un albero T in cui ogni nodo v contiene un valore binario (restitutito da `v.getElement()`) e si dice *alive* se tale valore è 1, e *dead* se esso è 0. Progettare un algoritmo ricorsivo `allLiveAncestors` che per ogni nodo $v \in T$ memorizzi in un campo v`.liveAncestors` il numero di antenati *alive* di v. Detta `allLiveAncestors(T,v)` la generica invocazione su un nodo $v \in T$, per risolvere il problema si eseguirà `allLiveAncestors(T,T.root())`.

   (a) Descrivere tramite pseudocodice `allLiveAncestors(T,v)`, specificandone con attenzione l'input e l'output.

   (b) Analizzare la complessità di `allLiveAncestors(T,T.root())` in funzione del numero n di nodi in T.

TEMPO COMPLESSIVO A DISPOSIZIONE: 1.5 ore