PRIMA PARTE

1. **[5 punti]** Dare la definizione di failure function per un pattern P
2. **[6 punti]** Dimostrare per induzione che per ogni albero binario proprio T non vuoto con m foglie e altezza h vale $m \leq 2^h$ (induzione su h).
3. **[5 punti]** L'array seguente rappresenta uno *heap* (mappato tramite level numbering):

(4,*)	(9,*)	(10,*)	(12,*)	(15,*)	(12,*)

   Mostrare l'array che rappresenta lo heap risultante dopo l'esecuzione delle operazioni `insert(7,*)`, `insert(6,*)`, `removeMin()`, `removeMin()`, mostrando anche lo heap (come albero e riportando solo le chiavi) dopo ciascuna operazione.

SECONDA PARTE

1. **[8 punti]** Si consideri un testo T di lunghezza n e un pattern P di lunghezza $m \leq n$. La seguente variante dell'algoritmo `findBrute(T,P)` usa un solo ciclo e confronta i caratteri di P e T da destra verso sinistra. Come `findBrute(T,P)`, restituisce un indice i tale che $P = T[i...i+m-1]$, se esiste, e restituisce -1 altrimenti.

```
i ← n-m; j ← m-1;
while (i ≥ 0) do {
  if (T[i+j]=P[j]) then {
    j ← j-1;
    if (j=-1) then return i;
  }
  else {
    i ← i-1;
    j ← m-1;
  }
}
return -1;
```

   Trovare un opportuno invariante per il ciclo, che serva per provare la correttezza dell'algoritmo.

2. **[8 punti]** Si vuole progettare un algoritmo *ricorsivo* `deepLeaf` per trovare la foglia più profonda in un albero binario proprio T. Detta `deepLeaf(T,v)` la generica invocazione su un nodo $v \in T$, per risolvere il problema si eseguirà `deepLeaf(T,T.root())`. Si tenga presente che per ogni sottoalbero T_v la profondità (in T_v) della sua foglia più profonda è pari all'altezza di T_v.

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

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

TEMPO COMPLESSIVO A DISPOSIZIONE: 1.5 ore