FONDAMENTI DI INFORMATICA II

Esercizio 1
Sia dato il problema di cercare l'elemento minimo di un'insieme di numeri naturali. Indicare la complessità del migliore algoritmo che risolve il problema, al variare della struttura dati utilizzata per memorizzare l'insieme.
Scrivere l'algoritmo per il caso d.

	Struttura dati	Complessità
a	Lista non ordinata	O(n)
b	Array non ordinato	O(n)
c	Array ordinato	O(1)
d	Max-heap	O(n)
e	Min-heap	O(1)
f	Albero binario	O(n)
g	Albero binario di ricerca	O(log n)

```
int min(int* max_heap, int n) {
  int minimo = max_heap[n/2];
  for (int i=n/2+1;i<n;i++) {
    if (max_heap[i] < minimo)
      minimo = max_heap[i];
  }
  return minimo;
}
```

Esercizio 2
Sia dato lo heap di caratteri:

[Z S H G M A B]

mostrare lo stato dello stesso e le chiamate ad **`up`** e **`down`**
a) Dopo l'inserimento del carattere 'P'

b) Dopo l'estrazione di un elemento (a partire dallo heap ottenuto al passo a)

a) [Z S H P M A B G] up(7) up(3)
B) [S P H G M A B] down(0) down(1) down(3)

Esercizio 3
Calcolare la complessità del blocco (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione):

```
{
  int a = 0;
  for (int i=0; i <= f(n)*g(n); i++)
     a += f(n);
}
```

con le funzioni `f` e `g` definite come segue:

```
int f(int x) {
  if (x<=1) return 1;
  int a = 1;
  for (int i=0; i <= x; i++)
     a++;
 a+= 2*f(x/2); cout << a;
 return 1 + 2*f(x/2) + a;
}
```

```
int g(int x) {
  if (x<=1) return 1;
  int a=0;
  for (int i=0; i <= x*x; i++)
     a+=1;
  cout << a + g(x-1);
  return a;
}
```

Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità dell'iterazione singola.

F:

for: numero iterazioni : O(n), complessità della singola iterazione: O(1), complessità del for: O(n)

TF(0,1)=a
TF(n)= bn + 2 TF(x/2) O(nlogn)

RF(n)= 1
RF(n)= bn + 4 RF(x/2) O(n^2)

G:

for: numero iterazioni : O(n^2), complessità della singola iterazione: O(1), complessità del for: O(n^2)

TG(0,1)=a
TG(n)=bn^2+T(n-1) O(n^3)

RG(n)= bn^2   O(n^2)

---
for
numero di iterazioni: O(n^4)
tempo di una iterazione: C[f(n)] +C[g(n)] = O(nlogn) + O(n^3)= O(n^3)

tempo del for: O(n^7)

Esercizio 4
Si scriva una funzione che, dato un albero binario ad etichette di tipo **`string`**, con puntatore alla radice **`t`**, restituisca come risultato il numero di nodi che hanno un numero dispari di discendenti. Si calcoli la complessità della soluzione proposta in funzione del numero di nodi dell'albero.

```
int dispari(Node* t, int& nodi) {
  if (!t) {
    nodi = 0;
    return 0;
  }
  int cs,cd,nodis,nodid;
  cs = dispari(t->left,nodis);
  cd = dispari(t->right,nodid);
  nodi = nodis+nodid+1;
  return cs+cd+((nodis+nodid)%2);
}
```

$T_{dispari}$ è O(N)

Esercizio 5
Si scriva una funzione che, dato un albero generico ad etichette di tipo **`string`**, memorizzato figlio-fratello, con due parametri **`father`** e **`son`** di tipo **`string`** ed un parametro intero **`n`**, inserisce nell'albero un nuovo nodo con etichetta **`son`** come **`n`**-esimo figlio del nodo con etichetta **`father`**. Si supponga che l'inserzione abbia sempre successo (nell'albero esiste un nodo con etichetta **`father`** e ha almeno **`n`**-1 figli).

```
void insertList(Node*& t, T& son) {
  Node* bro = t;
  t = new Node(son);
  t -> right = bro;
}

void insertN(Node* t, string& father, string& son, int n) {
  Node *a = findNode(father,t);
  if (n == 1) {
    insertList(a->left,son);
    return;
  }
  a = a->left;
  for(int i = 2; i<n;i++)
    a = a->right;
  insertList(a->right,son);
}
```