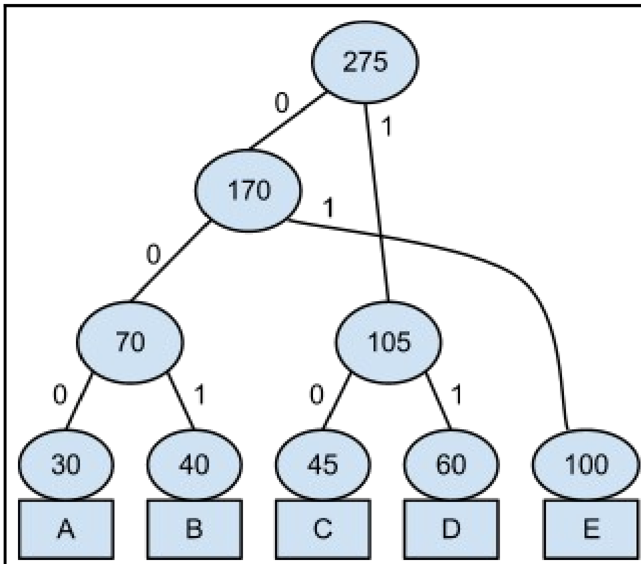
Esercizio 1

Dati i seguenti simboli con la relativa frequenza di apparizione in un testo:

Simbolo:	**Frequenza:**	**Codifica di Huffman:**
A	30	`000`
B	40	`001`
C	45	`10`
D	60	`11`
E	100	`01`

Eseguire l'algoritmo di Huffman per trovare una codifica che comprima il testo. Scrivere poi le codifiche di Huffman.



Esercizio 2

Sia data la seguente funzione:

```
void f(int a[], int last, int i=0) {
  if (i > last) return;
  if (2*i+1 > last) a[i]= a[i]*a[i];
  f(a, last, i+1);
}
```

Deve a è un array che rappresenta uno heap con valori interi.

a) Cosa fa la funzione?

b) L'array risultante è comunque uno heap? Se sì, mostrare che le proprietà dello heap sono preservate, altrimenti mostrare un contro-esempio.

a) Eleva al quadrato le etichette delle foglie dello heap.
b) L'array risultante non è necessariamente uno heap. Ad esempio, se applicata allo heap con due nodi formato da 3 (radice) e 2 (primo figlio), restituisce lo heap con 3 come radice e 4 come figlio sinistro, violando la proprietà dello heap.

Esercizio 3

Calcolare la complessità dell'espressione:

```
g(f(n)) + f(g(n))
```

in funzione di **n** (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni f e g definite come segue:

```
int f(int n) {
  if (n<=1) return 2;
  int a = f(n/2) + f(n/2);
  cout << a;
  return 1 + n + 2*a;
}
```

```
int g(int n) {
  if (n<=1) return 5;
  for (int i=1; i<=f(n); i++)
    b++;
  return 10 + g(n-2);
}
```

$T_f(1)= k_1$
$T_f(n)= k_2 + 2Tf(n/2)$ $\quad$ $T_f(n)$ è $O(n)$

$R_f(1)= k_1$
$R_f(n)= k_2\,n + 4R_f(n/2)$ $\quad$ $R_f(n)$ è $O(n^2)$

Calcolo $T_g(n)$
for:
Numero iterazioni: $R_f(n) = O(n^2)$
Complessità singola iterazione: $T_f(n) = O(n)$
Complessità del for: $O(n^3)$

$T_g(1)= k_1$
$T_g(n)= k_2n^3+ T_g(n-2)$ $\quad$ $T_g(n)$ è $O(n^4)$

$R_g(1)= 5$
$R_g(n)= 10+ R_g(n-2)$ $\quad$ $T_g(n)$ è $O(n)$

Tempo di g(f(n))= Tempo per il calcolo di f(n) + tempo per il calcolo di $g(n^2)$ =

$O(n)+O(n^8)= O(n^8)$

Tempo di f(g(n))= Tempo per il calcolo di g(n) + tempo per il calcolo di f(n) =

$O(n^4)+O(n)= O(n^4)$

Tempo per l'esecuzione di g(f(n)) + f(g(n)) = $O(n^4) + O(n^8) = O(n^8)$

Esercizio 4

Scrivere una funzione ricorsiva che, dato un albero binario a etichette intere, conta il numero di nodi che hanno più foglie maggiori o uguali a zero che minori di zero tra i propri discendenti.

```
int conta(const Node* t, int& pos, int& neg){
  if (!t) { pos = 0; neg = 0; return 0; }
  if (!t->left && !t->right) {
    pos = (t->info>=0)?1:0;
    neg = (t->info<0)?1:0;
    return 0;
  }
  int pos_left, pos_right;
  int neg_left, neg_right;
  int conta_left = conta(t->left, pos_left, neg_left);
  int conta_right = conta(t->right, pos_right, neg_right);
  pos = pos_left + pos_right;
  neg = neg_left + neg_right;
  return (pos>neg)?1:0 + conta_left + conta_right;
}
```

Esercizio 5

Scrivere una funzione che, dato un albero generico a etichette intere e memorizzazione figlio-fratello, conta il numero di nodi che hanno più figli maggiori o uguali a zero che minori di zero.

```
int conta(const Node* t){
  if (!t) return 0;
  const Node* n;
  int pos = 0, neg = 0;
  for(n=t->left; n != NULL; n = n->right){
    if(n->info >= 0) pos++;
    else neg++;
  return (pos>neg)?1:0 + conta(t->left) + conta(t->right);
}
```