Algoritmi e Basi di dati – Modulo di Algoritmi e Strutture dati - a.a. 2011/2012

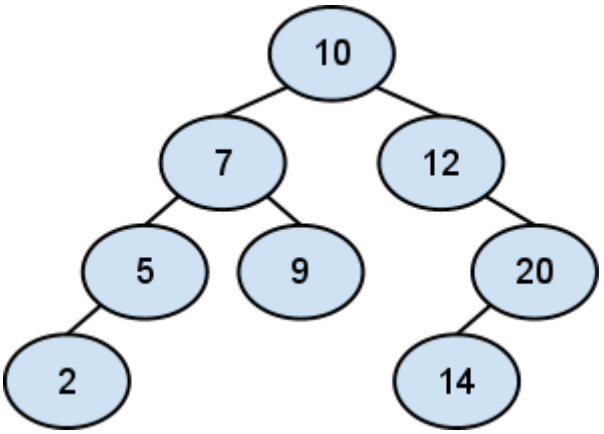
3 febbraio 2012

Cognome	Nome	Matricola

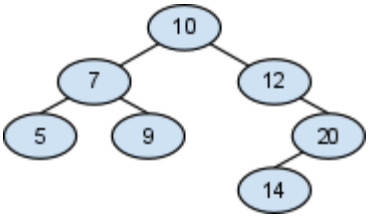
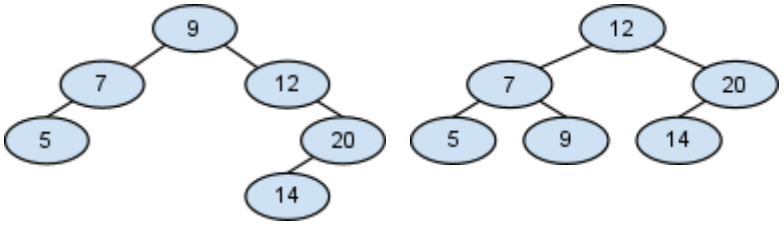
1	2	3	4	5

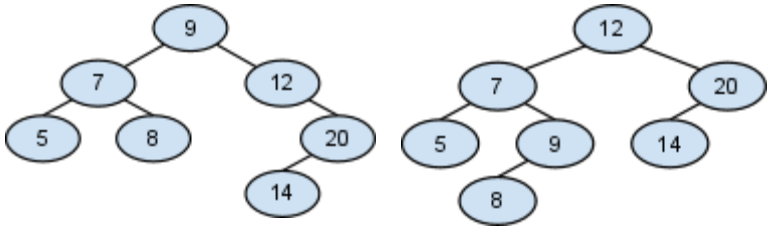
Esercizio 1

Dato l'albero binario di ricerca in figura:



Eseguire un'estrazione dell'elemento 2, un'estrazione della radice ed un successivo inserimento di un elemento di valore 8. Indicare l'albero ottenuto al termine di ogni operazione.

Dopo l'estrazione di 2:	
Dopo l'estrazione della radice:	Due possibilità, entrambe giuste: 

Dopo l'inserimento di 8:	

Esercizio 2

Calcolare la complessità in termini di occupazione di memoria della funzione **es2** in funzione di **n**. Indicare per esteso le relazioni di ricorrenza.

```
int es2(int n) {
  if (n <= 1) return 1;
  int a = 0;
  for (int i = 1; i <= f(n); i++)
    a += 3;
  cout << es2(n/2);
  return a + es2(n/2) + es2(n/2);
}
```

```
int f(int n) {
  if (n <= 1) return 1;
  return 10 + f(n-1);
}
```

Funzione f:

$M_f(1) = d$
$M_f(n) = c + M_f(n-1)$ $M_f(n)$ è $O(n)$

Funzione es2:

Si occupa una memoria O(n) per la chiamata a f() nella prima ricorsione + M(n/2) per le successive chiamate ricorsive. Anche se le ricorsioni sono 3, ogni volta la memoria viene liberata. La complessità dominante è quella della prima chiamata a f(). Quindi O(n).

Esercizio 3

Calcolare la complessità del **`for`** in funzione di **`n`>*0***.

```
for (int i=0; i <= g(f(100))*f(n); i++) cout << i;
```

con le funzioni **`f`** e **`g`** definite come segue. Indicare per esteso le relazioni di ricorrenza per il tempo e per il risultato delle funzioni e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

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

```
int f(int x) {
  if (x <= 0) return 1;
  int b = 3 + 2*f(x/2);
  cout << b + f(x/2);
  return b + b; }
```

Funzione g

Primo for :

Complessità della singola iterazione: O(1)

Numero di iterazioni : O(n)

Complessità del for: O(n)

$T_g(0)= d$

$T_g(n)= cn+ 2T_g(n-1)$ $T_g(n)$ è $O(2^n)$

$R_g(0)= d$

$R_g(n)= cn^2+ R_f(n-1)$ $R_g(n)$ è $O(n^3)$

Funzione f

$T_f(0)= d$

$T_f(n)= c+ 2T_f(n/2)$ $T_f(n)$ è $O(n)$

$R_f(0)= d$

$R_f(n)= c+ 4R_f(n/2)$ $R_f(n)$ è $O(n^2)$

Calcolo for del blocco:

Numero iterazioni: $f(g(100)) * f(n)= O(1) * O(n^2)= O(n^2)$

Complessità della singola iterazione: $O(1) + T_f(n) = O(1) + O(n) = O(n)$

Complessità del for: $O(n^3)$

Esercizio 4

Scrivere una funzione ricorsiva **`Node* heap(const int a[], int i, int last)`** che, dato uno heap memorizzato in array, restituisce l'albero binario corrispondente memorizzato con strutture di lista dinamiche (strutture con i campi **`label`**, **`left`**, **`right`**).

```
Node* heap(const int a[], int i, int last){
   if (i > last) return NULL;
   Node* t = new Node;
   t->label = a[i];
   t->left = heap(a, 2*i + 1, last);
   t->right = heap(a, 2*i + 2, last);
   return t;
}
```

Esercizio 5

Scrivere una funzione ricorsiva **`int piu_figli(const Node* t)`** che, dato un albero generico memorizzato con la tecnica figlio-fratello, restituisce il numero dei nodi che hanno più figli che fratelli successivi.

```
int piu_figli(const Node* t) {
 if (!t) return 0;
 return (conta(t->left) > conta(t->right)) + piu_figli(t->left) +
    piu_figli(t->right);
}

int conta(const Node* t) {
 if (!t) return 0;
 return 1 + conta(t->right);
}
```