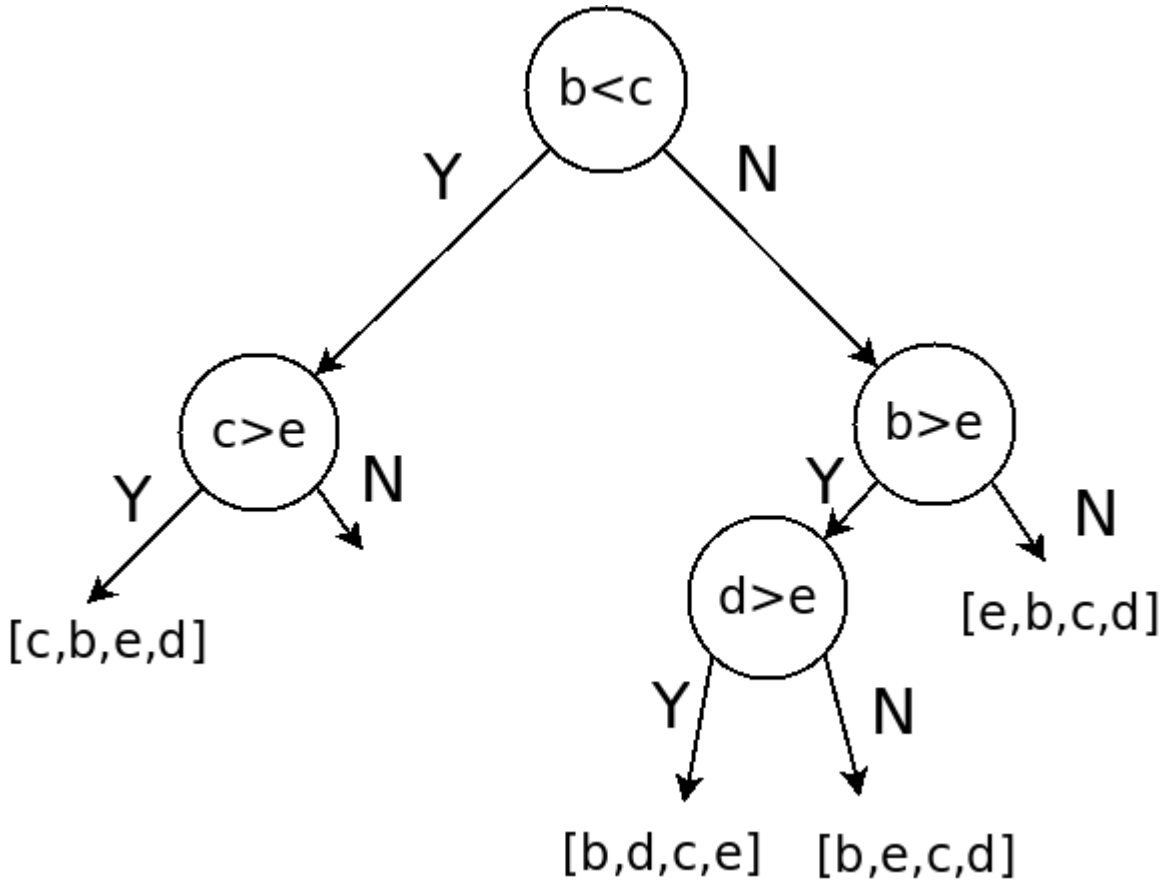
Algoritmi e Basi di dati – Modulo di Algoritmi e Strutture dati

23 luglio 2010 - ANNO ACCADEMICO 2009/10

NOME: **COGNOME:** **MATRICOLA:**

Esercizio 1

Mostrare l'albero di decisione per l'estrazione di un elemento da un max-heap contenente [a, b, c, d, e]. I nodi dell'albero dovranno riferirsi ai confrontri tra elementi dello heap.



1	*2*	*3*	*4*	*5*

Esercizio 2
Data la lista di interi con puntatore alla testa p

11 → 56 → 9 → 23 → 4 → 13

scrivere tutte le chiamate a mergeSort generate da mergeSort(p), specificando, per ogni chiamata, la lista a cui la funzione viene applicata. Non scrivere le chiamate a liste vuote.

Invocazione
mergeSort([11 56 9 23 4 13])
mergeSort([11 9 4])
mergeSort([11 4])
mergeSort([11])
mergeSort([4])
mergeSort([9])
mergeSort([13 23 56])
mergeSort([13 56])
mergeSort([13])
mergeSort([56])
mergeSort([23])

Esercizio 3
Calcolare la complessità della chiamata di funzione g(n) e del suo risultato 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:
Le funzioni **f** e **g** sono definite come segue:

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

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

$T_f(1)= k_1$
$T_f(n)= k_2 + 2T_f(n/2)$ $T_f(n)$ è $O(n)$
$R_f(1)$ è $O(n^2)$

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

$T_g(1)= k_1$
$T_g(n)= k_2\ n^3 + T_g(n-2)$ $T_g(n)$ è $O(n^4)$

$R_g(0,1)= 5$
$R_g(n)= 10+ R_g(n-2)$ $R_g(n)$ è $O(n)$

Esercizio 4
Sia dato un albero generico ad etichette intere memorizzato figlio fratello. Si scriva una funzione conta che restituisce il numero dei nodi dell'albero che hanno un etichetta minore di quella di tutti i loro figli. I nodi che sono foglie non vanno contati.

```
unsigned int conta(Node* t) {
  if (!t) return 0;
  if (!t->left) return conta(t->right);
  int min = t->left->label;
  Node* p = t->left->right;
  while (p) {
    if (p->label < min)
      min = p->label;
    p = p->right;
  }
  return conta(t->left) + conta(t->right) + (t->label<min);
}
```

Esercizio 5
Sia dato un albero binario ad etichette intere con valori compresi nell'intervallo (a,b) (estremi esclusi). Si scriva una funzione isBST che restituisce true se l'albero è un albero binario di ricerca, false altrimenti. Si calcoli la complessità della soluzione proposta in funzione del numero di nodi dell'albero.

```
bool isBST(Node* root, int m=a, int M=b) {
  if (!root) return true;
  int label = root->label;
  if ((label <= m) || (label >= M))
    return false;
  return isBST(root->left, m, label) &&
         isBST(root->right, label, M);
}
```

Complessità O(n)