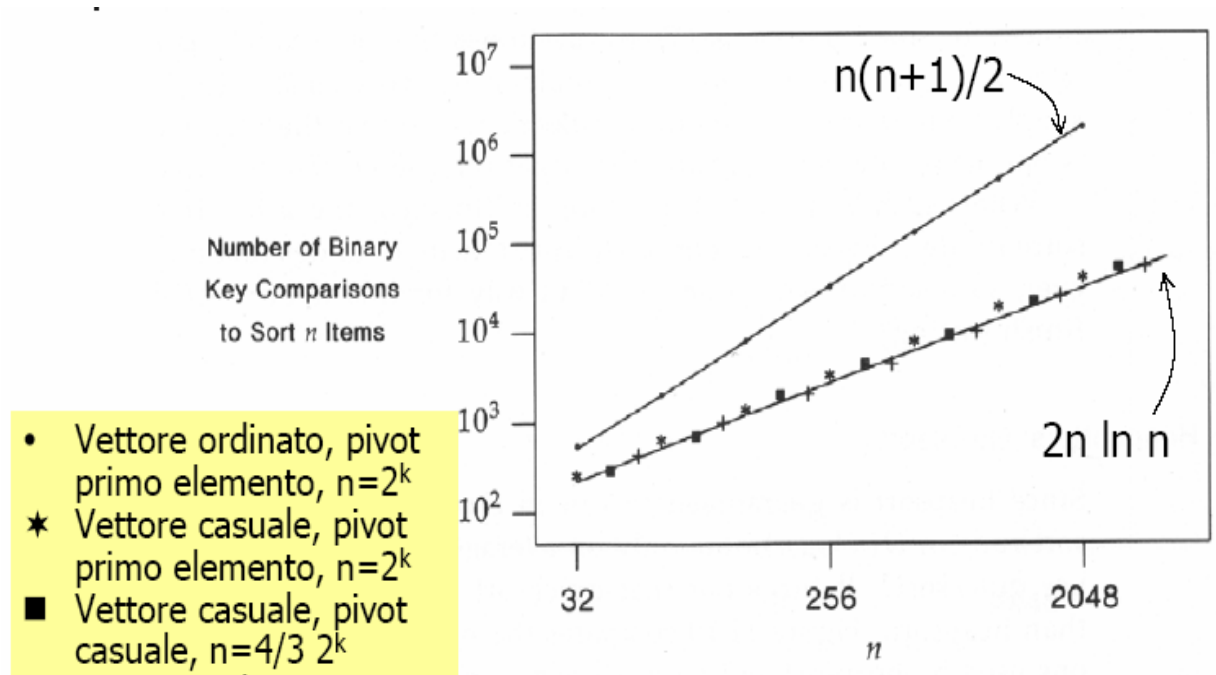
Esercitazione

Quicksort:Quale pivot?



- Vettore ordinato, pivot primo elemento, $n=2^k$
- ✶ Vettore casuale, pivot primo elemento, $n=2^k$
- ■ Vettore casuale, pivot casuale, $n=4/3\ 2^k$
- \+ Vettore ordinato, pivot casuale, $n=\ 5/4\ 2^k$

Visualizzazione di Algoritmi

Esempi in JAVA:

\j2sdk1.4.2_0x\demo\applets\SortDemo

Esempi in Rete:

http://www.di.unipi.it/didadoc/asd1/page6.html
http://gauguin.info.uniroma2.it/~finocchi/ASD2003/animazioni.html
http://www.cs.hope.edu/~alganim/ccaa/
http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/alg_anim.html

Tempo d'esecuzione

Volendo avere una stima effettiva del tempo impiegato dai propri algoritmi è possibile in JAVA utilizzare il metodo **currentTimeMillis()**, della classe **System**:

```
public static long currentTimeMillis()
```

Il valore di ritorno del metodo corrisponde al numero di millisecondi trascorsi dal momento in cui viene chiamato alla mezzanotte del 1 Gennaio 1970.

L'unità di misura del tempo di sistema è solitamente più granulare (1/10, 1/100 di millisecondo).

Alternativamente è possibile fare riferimento alla classe **Date** del package **java.util**

La classe StopWatch

```
public class StopWatch {
    private long elapsedTime;
    private long startTime;
    private boolean isRunning;

    public StopWatch(){ reset(); }

    /** Ferma il cronometro e azzera il tempo totale
        trascorso.*/
        public void reset(){
        elapsedTime = 0;
        isRunning = false;
        }

    /* Fa partire il cronometro */
    public void start(){
    if (isRunning) return;
        isRunning = true;
        startTime = System.currentTimeMillis();   }
```

La classe StopWatch

```
/* Ferma il cronometro. Il tempo non viene più accumulato
   e viene sommato al tempo trascorso.*/

    public void stop(){
        if (!isRunning) return;
        isRunning = false;
        long endTime = System.currentTimeMillis();
        elapsedTime = elapsedTime+endTime-startTime;
    }

    /* Restituisce il tempo totale trascorso.*/

    public long getElapsedTime(){
        if (isRunning)
        {
         long endTime = System.currentTimeMillis();
         elapsedTime = elapsedTime+endTime-startTime;
         startTime = endTime;
        }
        return elapsedTime;    }
```

Esempio

```
…
StopWatch timer = new StopWatch();

            timer.start();
            Sorts.insertionSort( aux );
            timer.stop();

System.out.println("Elapsed time Insertion Sort: "+
timer.getElapsedTime() + " milliseconds");
```

Esame del 18 Luglio 2002

Scrivere un metodo statico in Java che, data in input una lista disordinata di oggetti doppiamente concatenata, ne riordini gli elementi ridefinendo opportunamente i riferimenti tra gli oggetti. Si utilizzi l'algoritmo del **Selection Sort**. Ammettere che gli oggetti possano avere chiavi ripetute. Assumere che la chiave sia un valore intero.

Soluzione

```
public static DblLinkedList selectionSort(DblLinkedList l) {

 DblListNode pi,pj,least;
 for (pi=l.getHead(); pi!=l.getTail(); pi=pi.getNext()) {
   for (pj=pi.getNext(),least=pi; pj!=null; pj=pj.getNext())
     if(((Comparable)
       pj.getInfo()).compareTo(least.getInfo()) < 0)
         least = pj;
     if (least!= pi)
              swapInfo(least,pi);
    }
 return l;
}
private static void swapInfo(DblListNode e1, DblListNode e2)
   {
             Object tmp = e1.getInfo();
             e1.setInfo(e2.getInfo());
             e2.setInfo(tmp);
         }
```

Insertion Sort su liste

IDEA: Si avanza da *sx* verso *dx* a partire dal secondo elemento. Un elemento alla volta viene inserito nel sottogruppo già ordinato a *sx* (all'inizio contenente solo il primo elemento).

Per ogni nuovo elemento, quindi, viene ricercata la posizione all'interno della parte ordinata, slittando gli elementi per creare uno spazio libero.Trattandosi di una implementazione dell'algoritmo su una lista linkata possiamo evitare lo shift degli elementi modificando direttamente la struttura dei puntatori dei nodi interessati. In pratica effettuiamo prima una cancellazione e poi un inserimento.

ListSort

```
public static LinkedList listSort(LinkedList l) {
  if(l.isEmpty( )) return l;

  ListNode i, j, j_1, i_1;

  for (i_1=l.getHead(), i=l.getHead().getNext(); i!= null;
        i_1 = i,i = i.getNext()){

 //cerco la corretta posizione di i tra head ed i-1
 //partendo (tramite il puntatore j) dall'inizio della
 //lista fino a quando non trovo un elemento j maggiore di i

   for (j = l.getHead(), j_1 = null; j!= i &&
     (((Comparable)j.getInfo()).compareTo(i.getInfo())<= 0);
           j_1 = j, j = j.getNext());
```

ListSort

```
if (i!=j){ //Inserisco i tra j_1 e j

       if (j == l.getHead())
          l.setHead(i);  //inseriamo i in testa
         else
             j_1.setNext(i);
             i_1.setNext(i.getNext());
             i.setNext(j);
     }

   }
   return l;
 }
```

Esame del 1 Ottobre 2003

Scrivere un metodo java statico che, ricevuta in input una lista doppiamente linkata, ne riordini gli elementi col metodo del **MergeSort**.

```
public static LIST Sort( LIST L )
```

I nodi della lista contengono, nel campo ***info***, un riferimento ad una classe ***object***, sulla quale è definito un ordinamento totale.

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari.

2° Prova in Itinere 2001/02

Scrivere un metodo Java che, dato un albero binario di ricerca, calcoli la lunghezza del cammino caratteristico del nodo contenente il successore della chiave della radice.

Soluzione

```
public static int getLenghtRootSucc(BSTree t){
     int l= 0;
     BTNode succ;

     if ((!t.isEmpty())&&(t.getRoot().getRight()!=null) )
     {      //Il sottoalbero dx della radice non e' vuoto

       succ = t.getRoot().getRight();
       l++;
       while (succ.getLeft() != null)
        {
           succ = succ.getLeft();
           l++;
         }
     }
     return l;
   }
```

Esame del 18 Giugno 2002

Scrivere un metodo Java che legga da uno stream di Input una sequenza di interi e costruisca un albero binario bilanciato (utilizzare eventualmente una struttura dinamica di supporto).

Definire e implementare tutte le classi accessorie utilizzate.

Soluzione

```
public static BTNode createBalancedBinaryTree(int s[]) {
   QueueListByComposition Q= new QueueListByComposition();
   BTNode root = new BTNode (new Integer(s[0]));
   Q.enqueue(root);
   for (int i=1; i<s.length; i++){
     BTNode n = (BTNode) Q.firstEl();
     BTNode n1 = new BTNode(new Integer(s[i]));
     if ( n.getLeft() == null )
              n.setLeft(n1);
     else { n.setRight(n1);
            Q.dequeue(); //Eliminiamo il nodo n completo
                         //di figlo sx e dx, e carichiamo
                         //quest'ultimi su Q
                         //come nuovi nodi da riempire.
            Q.enqueue(n.getLeft());
            Q.enqueue(n1);
     }
   }
   Q.clear();   return root;
  }
```

Esame del 23 Settembre 2002

Scrivere un metodo statico Java che, date due chiavi e un albero binario di ricerca, restituisca il nodo antenato più vicino ai nodi con le chiavi fornite.

Nota: le chiavi possono essere non presenti nell'albero.

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari.

Soluzione

```
public static BTNode LCA(BSTree A,Comparable n,Comparable m)
{
     if (n.compareTo(m)>0) return LCAaux(A.getRoot(),n,m);
      else  return LCAaux(A.getRoot(),m,n);
    } //LCAaux assume che n1 < n2

 private    static    BTNode    LCAaux(BTNode    r,Comparable
  n1,Comparable n2){
    if ( (((Comparable)r.getInfo()).compareTo(n1) < 0) &&
           (((Comparable)r.getInfo()).compareTo(n2) < 0 ))
        return LCAaux(r.getRight(), n1, n2);
    else if ( (((Comparable)r.getInfo()).compareTo(n1)>0) &&
           (((Comparable)r.getInfo()).compareTo(n2) > 0 ))
            return LCAaux(r.getLeft(), n1, n2);
         else
         if ((((Comparable)r.getInfo()).compareTo(n1)<=0) &&
             (((Comparable)r.getInfo()).compareTo(n2)>= 0 &&
                 search(n1,r)!=null && search(n2,r)!=null ))
             return r;
         else return null; }
```

Esame del 28 Gennaio 2003

Scrivere un metodo statico Java ***TreeList*** che, dati in input un albero binario ***A*** ed un numero intero ***K***, restituisca in una lista i campi *info* dei nodi dell'albero ***A*** di livello ***K***.

`LISTA Livello(TREE A, int K)`

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari.

Soluzione

```
public static LinkedList level(BSTree T,int k) {
 LinkedList l = new LinkedList();
 QueueListByComposition queue = new QueueListByComposition();
 int level = -1;  Pair pair;
    BTNode p = T.getRoot();
    if (p != null) {
      queue.enqueue(new Pair(p, 0));
      while (!queue.isEmpty()) {
        pair = (Pair) queue.dequeue();
        p = pair.node;
        level = pair.level;
       if (level == k+1) return l;
        if (level == k) //Stiamo sul livello k
          l.insertTail(p.getInfo());
        if (p.getLeft() != null)
          queue.enqueue(new Pair(p.getLeft(), level + 1));
        if (p.getRight() != null)
          queue.enqueue(new Pair(p.getRight(), level + 1));
      }    }
     return l;}
```

Esame del 18 Febbraio 2003

Scrivere un metodo statico JAVA

`void InsList(Tree T, int n, int k, Object l)`

che, preso in input un albero binario ***T,*** rimpiazzi l'etichetta del nodo ***n***-esimo (contando da sinistra verso destra) del livello ***K***-esimo, qualora esista, con l'etichetta ***l.***

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari.

Esame del 9 Luglio 2003

Scrivere un metodo statico JAVA che, preso in input un albero binario non necessariamente completo, restituisca, tramite una lista semplicemente linkata, le altezze dei livelli (zero o più) aventi un numero di nodi pari all'altezza dell'albero.

La firma del metodo richiesto è la seguente

```
public static List search (TREE A)
```

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari.

Esame del 10 Dicembre 2003

Si consideri una classe ***MyObject***, contenente un campo intero, sulla quale è definito un ordinamento totale.
Si consideri un ***BST*** di nodi di classe ***ObjNodo***, contenenti un campo ***info*** di classe ***Object***.

Scrivere un metodo Java della classe ***BST*** che, dato un BST di ***ObjNodo*** e un oggetto di classe ***MyObject***, inserisca l'oggetto dato nel ***BST***.

```
public void objInsert(Object elem)
```

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari (non dimenticare i metodi per definire l'ordinamento totale su MyObject).

Esame del 27 Gennaio 2004

Scrivere un metodo statico Java che, dato in input un albero binario di ricerca, restituisca il numero di nodi completi utilizzando un algoritmo ricorsivo.

La firma del metodo richiesto è la seguente

```
public static int conta(BST Tree)
```

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari.

Esame del 18 Febbraio 2004

Si consideri una classe ***Tree*** di alberi binari non di ricerca, i cui nodi abbiano un campo ***info*** di tipo intero.

Scrivere un metodo statico JAVA che, dato in input un oggetto della classe ***Tree*** e un valore di ***soglia***, restituisca **vero** se esiste un cammino caratteristico di una foglia tale che la somma dei valori ***info*** dei nodi presenti nel cammino sia maggiore del parametro ***soglia***.

```
public static boolean find(TREE A, int soglia)
```

Implementare il metodo richiesto e tutte le classi e i metodi accessori necessari.