Data Structures - Simple Sorting 3

Dr. TGI Fernando [1] [2]

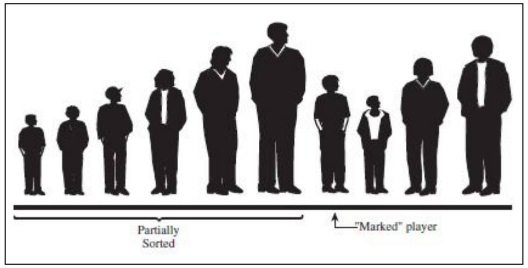
[1]Email: tgi.fernando@gmail.com
[2]URL: http://tgifernando.wordpress.com/

Insertion Sort

- Best of the elementary sorts.
- Still $O(N^2)$ time.
- Twice as fast as bubble sort.
- Somewhat faster than the selection sort in normal situations.
- Slightly complex than bubble and selection sorts.
- Used as the final stage of more sophisticated sorts, such as `quicksort`.

Description

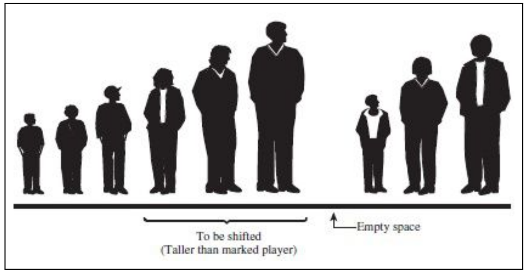
- Let us assume that the baseball team is partially sorted.
- Put an imaginary marker such that the players to the left of this marker are partially sorted.
- This means that they are sorted among themselves; but not necessarily in their final positions.



- Partial sorting did not take place in the `bubble sort` and `selection sort`.

Description (Contd.)

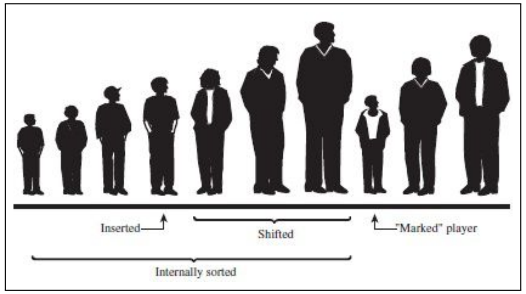
- Next insert the marked player in the appropriate place in the (partially) sorted group.
- To provide a space for this shift, first take the marked player out of line (stored in a temporary variable).



- Then shift the sorted players to make room (tallest $\rightarrow$ marked player's spot, next tallest $\rightarrow$ tallest's spot, and so on).

Description (Contd.)

- This shifting process stops when the first player that is shorter than the marked player is met.
- In this instance, the marked player is inserted into the space the last shift opened up.



- Next move the marked player one position to the right.
- This process is repeated until all the unsorted players have been inserted into the right place in the partially sorted group.

`InsertionSortArray` class

```java
class InsertionSortArray {
  private long[] a; // ref to array a
  private int nElems; // number of data items
  //-----------------------------------------
  public ArrayIns(int max) { // constructor
    a = new long[max];  // create the array
    nElems = 0;      // no items yet
  }
  //-----------------------------------------
  public void insert(long value) { // put element into array
    a[nElems] = value;  // insert it
    nElems++;      // increment size
  }
  //-----------------------------------------
  public void display () {...} // display array contents
  public insertionSort () {...}
}
```

Efficiency of the Insertion Sort

If there N number of elements in the array,

Avg. no of comparisons on the first pass $= 1 = \frac{2}{2}$

Avg. no of comparisons on the second pass $= \frac{1+2}{2} = \frac{3}{2}$

Avg. no of comparisons on the third pass $= \frac{1+2+3}{3} = \frac{6}{3} = 2 = \frac{4}{2}$

Avg. no of comparisons on the fourth pass $= \frac{1+2+3+4}{4} = \frac{10}{4} = \frac{5}{2}$

$\vdots$

Avg. no of comparisons on the last pass $=$

$\frac{1+2+3+\cdots+(N-1)}{N-1} = \frac{N\cancel{(N-1)}}{2\cancel{(N-1)}} = \frac{N}{2}$

Efficiency (Contd.)

$\therefore$ Average no of comparisons

$$
\begin{aligned}
&= \frac{2}{2} + \frac{3}{2} + \frac{4}{2} + \frac{5}{2} + \cdots + \frac{N}{2} \\
&= \left(\frac{1}{2} + \frac{2}{2} + \frac{3}{2} + \frac{4}{2} + \frac{5}{2} + \cdots + \frac{N}{2}\right) - \frac{1}{2} \\
&= \frac{N(N+1)}{4} - \frac{1}{2} \\
&\approx \frac{N(N+1)}{4}
\end{aligned}
$$

Efficiency (Contd.)

- The number of copies is approximately the same as the number of comparisons.
- Copying does not take time as swapping.
- For random data, this algorithm runs twice as fast as the bubble sort and faster than the selection sort.
- The insertion sort runs in $O(N^2)$ time for random data.
- For data that is almost sorted, the algorithm runs in almost $O(N)$ time.
- For data arranged in inverse sorted order, the algorithm runs no faster than the bubble sort.

Memory

- All three algorithms, operations carried out inside the array.
- Very little extra memory is required - an extra variable to store an item temporarily while it's being swapped.

Comparing the simple sorts

- Bubble sort is suitable if the amount of data is small.
- The selection sort minimizes the number of swaps, but the number of comparisons is still high.
- Useful when amount of data is small and swapping data is very time consuming.
- The insertion sort is the most versatile bet in most cases, assuming amount of data is small or the data is almost sorted.