THE SORTING ALGORITHMS

By
VIKRAM
SHARAD
KIRANKUMAR
PRABHAKAR

4/2009

Contents

1 BUBBLE SORT:

Bubble sort is a simple sorting algorithm use to sort the elements. It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements *bubble* to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort.

1.1 The Algorithm:

```
BUBBLESORT(A ,n)
  for i = 1 to n
      for j = 0 to ( n – i )
        if A[ j ] > A[j + 1]
              A[ j ] ⇔ A[j + 1]
```

Bubble sort has worst-case and average complexity both (n^2), where n is the number of items being sorted. Even other (n^2) sorting algorithms, such as insertion sort, tend to have better performance than bubble sort. Therefore bubble sort is not a practical sorting algorithm when n is large.

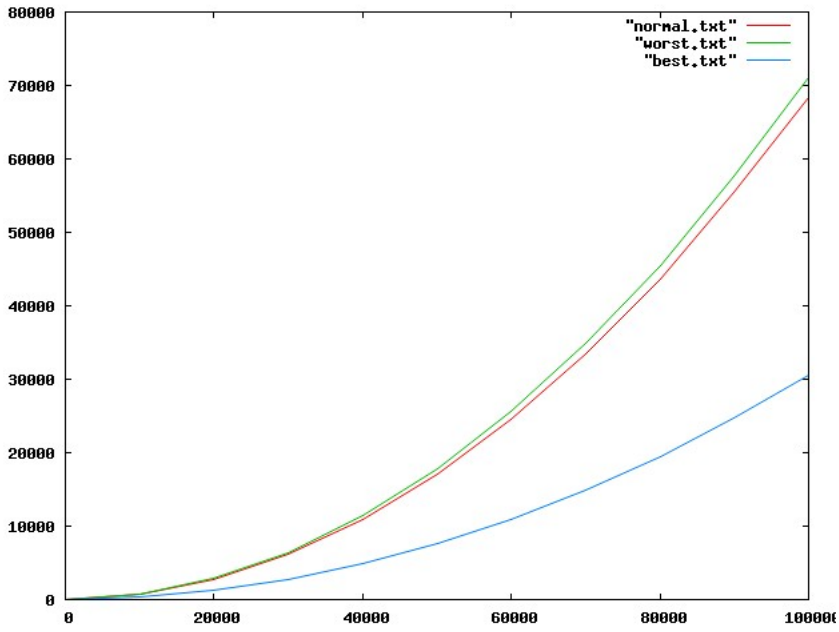

Figure 1: Gnuplot for bubble sort

Consider test case with numbers – 4 2 5 1 2 and on each iteration of inner loop, (2 4 5 1 2)→ (2 4 5 1 2)→ (2 4 1 5 2)→ (2 4 1 2 5)→ (2 4 1 2 5)→ (2 1 4 2 5)→ (2 1 2 4 5)→ (1 2 2 4 5)→ (1 2 2 4 5)→ (1 2 2 4 5)

1.2 Worst Case

The worst case occurs when the elements are sorted in non-ascending order, because the largest element need to bubbled till last and hence the worst case is $O(n^2)$ time. The graph for worst case is shown in the figure.

1.3 Best Case

The best case occurs when the elements are already sorted, because largest element is already in right place and needs no swapping. This case takes O(n) time. The graph for best case is shown in the figure.

2 SELECTION SORT

Selection sort is one of simplest sorting technique in which we select smallest item in the list and exchange it with the first item.Obtain the second smallest element in the list and exchange it with the second element and so on.Finally all the items will be arranged in ascending order.In otherwords selection sort performs sorting by repeatly putting the largest element in the unprocessed portion of the array to the end of the this unprocessed portion until the whole array is sorted.

2.1 The Algorithm:

```
for i←1 to n – 1
    do pos ← i
    for j ← i + 1 to n
        if A[j] < A[pos]
            then pos←j
    if pos ≠ i
        then A[i] ⇔ A[pos]
```

Selection sort has two loops each of which execute n times. Hence the selection sort is $O(n^2)$. The current element is assumed to be the smallest and is compared with remaining elements. The position of the smallest element is found. If the smallest element is not the current position, then the two elements are swapped.

Consider a test case with elements – 45 20 40 5 15
(45 20 40 5 15)→ (5 20 40 45 15) → (5 15 40 45 20)→
(5 15 20 45 40) → (5 15 20 40 45)

2.2 Worst Case

Worst case occurs if the elements are already sorted in the non-ascending order, because for each iteration the smallest element needs to be swapped and thus the Worst Case is $O(n^2)$.

2.3 Best Case

Best case occurs if the elemsnts are already sorted in non-descending order, because current element is smallest and no element needs to be swapped and thus the Best Case is O(n).

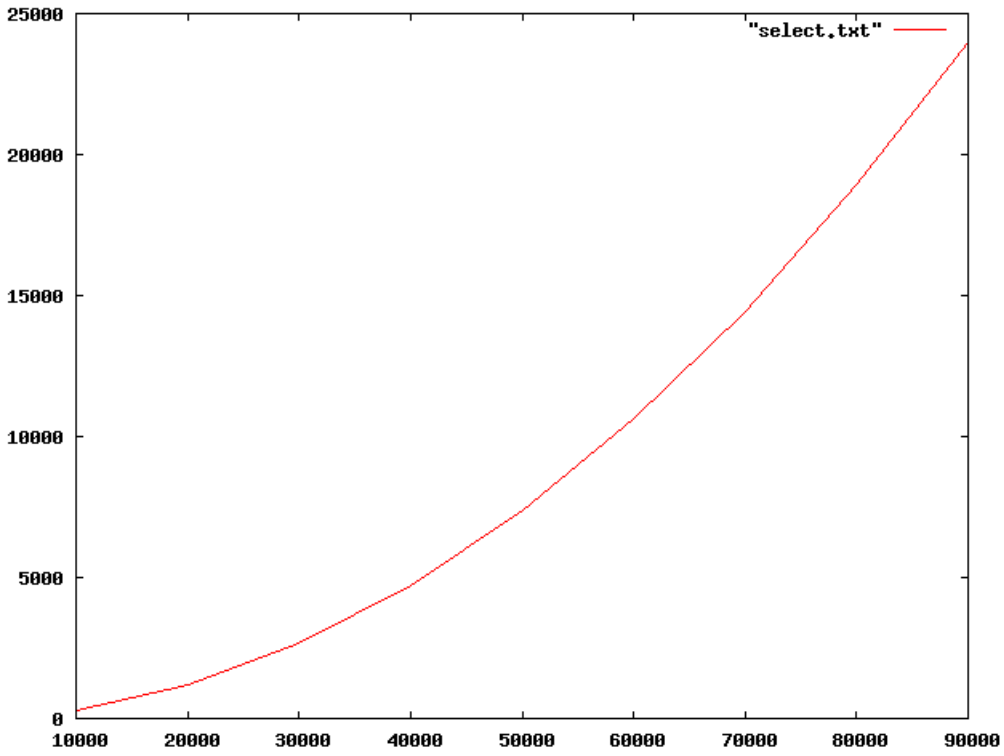

Figure 2: Gnuplot for selection sort

3 THE HEAP SORT

This sorting technique builds a max heap tree, in which *none* of the parent nodes are smaller than their children, and then sorts the elements by extracting the topmost element. A sequence of events takes place in this sorting. First the main function calls 'heapSort', which in turn calls buildMaxHeap. This is the basic tool for building the max heap. It builds the max heap tree from the lower nodes to the root by calling the maxHeap function. maxHeap function checks always that parent is larger than their children, and if the property is violated the largest child is exchanged with the parent.

3.1 The Algorithm:

```
LEFT(i)
        return 2i + 1

RIGHT(i)
        return 2i + 2

MAX-HEAPIFY(A, i)
        l ← LEFT(i)
        r ← RIGHT(i)
        if l ≤ size[A] and A[l] > A[i]
            then largest ← l
            else largest ← i
        if r ≤ size[A] and A[r] > A[largest]
            thenlargest ← r
        if largest ≠ i
            then exchange A[i] ⇔ A[largest]
                 MAX-HEAPIFY(A,largest)
```

The recurence relation for MAX-HEAPIFY is $T(n) \leq T(2n/3) + \Theta(1)$, since each childern's subtrees have a size at most $2n/3$ – the worst case being when the last row of the tree is exactly half full. Thus MAX-HEAPIFY takes $T(n) = O(\lg n)$.

```
BUILD-MAX-HEAP(A)
        for i ← ⌊n/2⌋ downto 1
            do MAX-HEAPIFY(A,i)
```

BUILD-MAX-HEAP takes O(n) time.

```
HEAP-SORT(A)
        BUILD-MAX-HEAP(A)
        for i ← n downto 2
            do exchange A[1] ⇔ A[i]
                n ← n – 1
                MAX-HEAPIFY(A,1)
```

HEAP-SORT takes O(n lg n) time since call to BUILD-MAX-HEAP takes O(n) time and each of the n–1 calls to MAX-HEAPIFY takes O(lg n) time.

For the elements - 10, 5, 16, 8, 68, 21, 43; the heapSort procedure results in a non-descendingly sorted elements. The state of the elements at each instance within heapSort iterative with i from number of elements to 2 is:
(10 5 16 8 68 21 43)→ (16 10 43 8 5 21 68)→ (16 10 21 8 5 43 68)→
(5 10 16 8 21 43 68)→ (8 10 5 16 21 43 68)→ (5 8 10 16 21 43 68)→
(5 8 10 16 21 43 68)

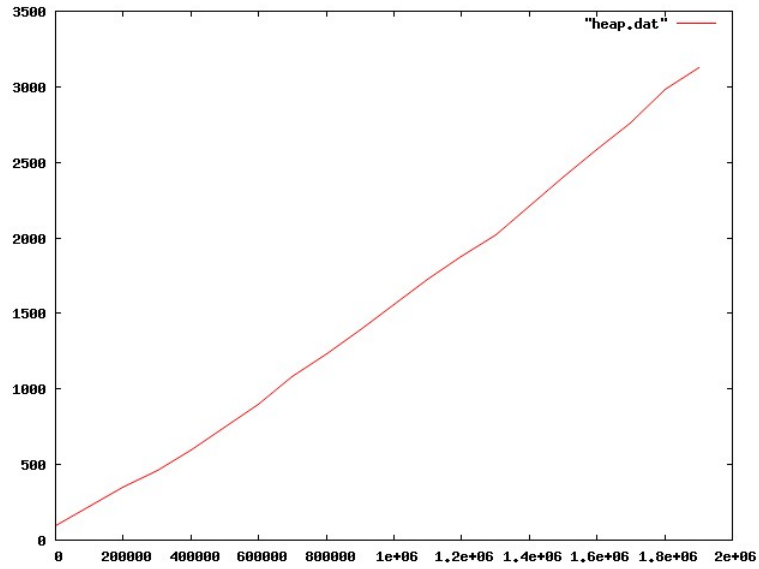

Figure 3: GnuPlot for Heap Sort

3.2 Analysis of Run Times:

Any of General, Worst or Best Cases of inputs need O(n) time to BUILD-MAX-HEAP. Further, any of "add" or "remove" operations takes O(lg n) time. Hence, runtimes for heap sorting in all the three cases is *always* same with O(n lg n) time.

4 QUICK SORT

Quick Sort is based on the technique *'Divide and Conquer'*. It divides the elements into two parts and then sorts them. This sorting is continued recursively till all the elements are sorted. Quick Sort has been implemented using two functions - Partition and QuickSort.

Paritition routine takes the middle element as *pivot*, rearranging elements such that elements smaller than the pivot is moved onto left of pivot and elements greater than pivot are moved onto its right. This sorting technique swaps the elements even if they are equal, hence is considered as an *unstable algorithm*.

```
PARTITION (A, p, r)
        x ← A[(p + r) / 2]
        i ← p – 1
        j ← r + 1
        while TRUE
            do repeat j ← j – 1
                until A[j] ≤ x
            repeat i ← i + 1
                until A[i] ≥ x
            if i < j
                then exchange A[i] ⇔ A[j]
                else return j
```

Quick Sort calls the partition subroutine to obtain the pivot element and recursively call itself to sort elements from p^{th} element to pivot and then from pivot to r^{th} element. This recursive call is done until p < r.

```
QUICKSORT (A, p, r)
        if p < r
            then q ← PARTITION(A, p, r)
                QUICKSORT(A, p, q)
                QUICKSORT(A, q+1, r)
```

The steps for sorting of an input is given below :-
(4 2 3 5 1)→ (1 2 3 5 4)→ (1 2 3 5 4)→ (1 2 3 5 4)→ (1 2 3 5 4)→ (1 2 3 5 4)→ (1 2 3 5 4)→ (1 2 3 4 5)→ (1 2 3 4 5)→ (1 2 3 4 5)

4.1 Worst Case

The Worst case happens when the elements are already sorted. The partition routine returns the value as i+1. Thus the quicksort has to call partition n times and there is a loop of n times inside the partition.Thus the order of the quickSort becomes
T(N) = T(N-1) + O(N)
$T(N) = O(N^2)$

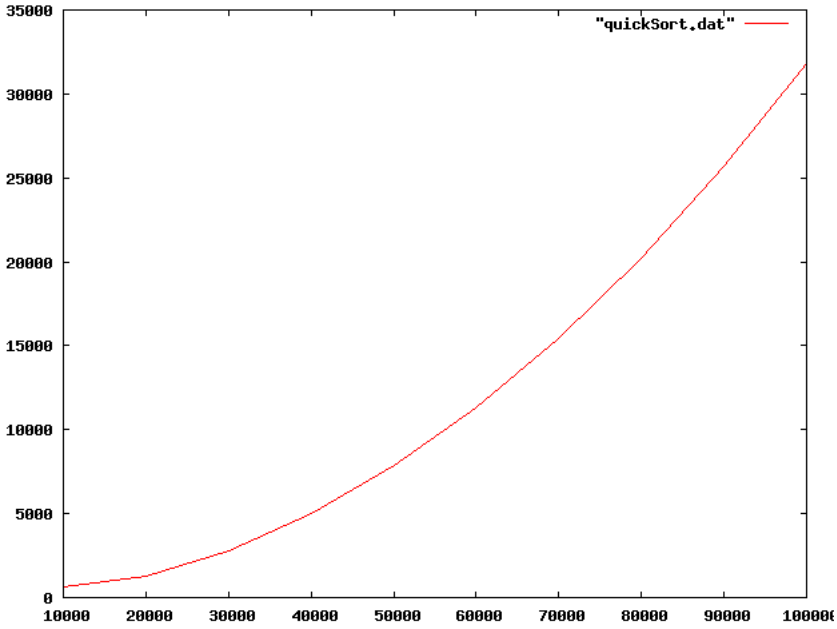

Figure 4: Plot for Quick Sort Worst Case

4.2 Best Case

The Best case is when the partiton function returns the middle element. when this happens, the array is split into half and N reduces by half every time. Thus in the best case, We get the time of QuickSort as
$T(N) = \leq 2T(n/2) + O(n)$ $T(N) = O(n \lg n)$

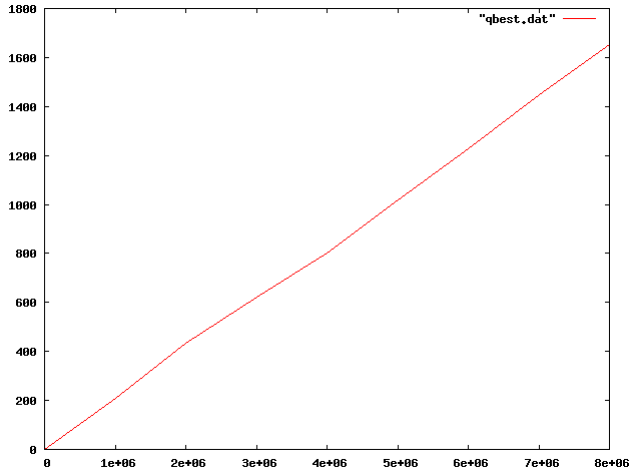

Figure 5: Plot for Quick Sort Best Case

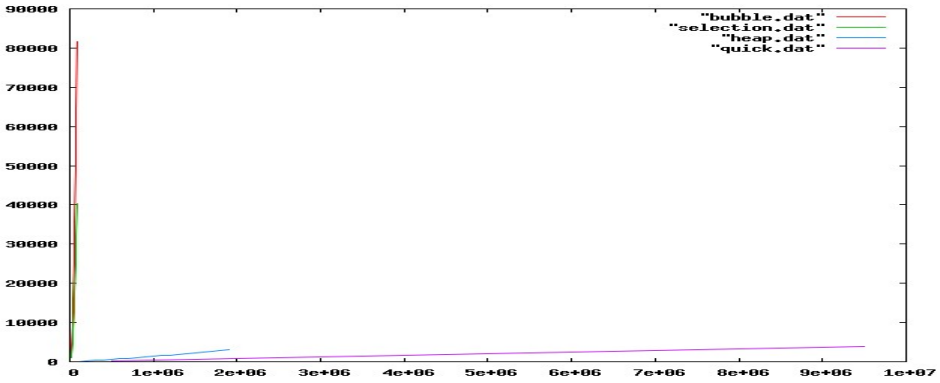

Figure 6: Comparing All Sorts

Comparing all the sorts—Bubble, Selection, Heap and Quick, there was huge difference in order of time.

References

[1] Cormen, Thomas H et al, 2005, *Introduction to Algorithms—Second Edition*, Prentice-Hall of India.

[2] Levitin, Anany, 2003, *Introduction to The Design & Analysis of Algorithms*, Pearson Education

LaTeX References

[3] LESLIE LAMPORT, 1985, *LaTeX—A Document Preparation System—User's Guide And Reference Manual*,Addison-Wesley, Reading.

[4] LaTeX Tutorials, 2003, *A Primer—Indian TeX Users Group*, India.

[5] Greenberg, Harvey J, 1999, *A Simplified Introduction to LaTeX* , Denver.

[6] *Internet Sources*