CS 202 : Data Structure and Algorithm

Assignment 2

Pranav Gupta (B15227)

6 March, 2017

Contents

0.1 Problem Statement

Implement the following sorting algorithms using C++ programing language and sort the input sequence in ascending order.

1. Insertion sort
2. Bubble sort
3. Selection sort
4. Rank sort
5. Merge sort
6. Quick sort

0.2 My Conclusions From Assignment :

1. Selection Sort,Insertion Sort and Bubble sort are all **in place** sorting algorithms so these can used where input is small(also mostly sorted) and memory is costly.

2. One of the good things about selection sort is that it never makes more than n swaps so can be used quite efficiently where memory write is a costly operation(as selection sort first iterates and then selects the only the required element to be swapped).

3. Also the algorithm like merge sort needs O(n) extra space so the program reported segmentation fault as our OS keeps bounds on allocation of memory to a certain program and that limit was violated in case of this large input for merge sort and hence program crashed.

4. Insertion sort should not be used where memory write is a costly operation as it makes too much swaps.

5. We also got to know about the stability of sorting algorithms which means that if we have some identical elements in our array then after sorting the indexing of those identical elements should not change. Insertion Sort,Merge Sort,Bubble Sort are some examples of stable sorting algorithms. Some sorting algorithms are not stable, like Heap Sort, Quick Sort, etc.

6. In computer graphics bubble sort is popular for its capability to detect a very small error (like swap of just two elements) in almost-sorted arrays and fix it with just linear complexity (2n)

7. For smaller input size insertion sort worked better.

0.3 Insertion Sort

0.3.1 Pseudo Code

INPUT : $A[1..n]$, array of integers

OUTPUT : Rearrangement of A such that $A[1] \leq A[2] \leq ... \leq A[n]$

for $j = 2$ to n **do**

    $key = A[j]$ ▷ Insert $A[j]$ into the sorted sequence $A[1....j-1]$

    $i = j - 1$

    while $i > 0$ and $A[i] > key$ **do**

        $A[j+1] = A[i]$

        $i = i + 1$

    end while

    $A[i+1] = key$

end for

0.3.2 Running Time Analysis

Best case : $O(n)$

Worst case : $O(n^2)$

0.4 Bubble Sort

0.4.1 Pseudo Code

INPUT : $A[1..n]$, array of integers
OUTPUT : Rearrangement of A such that $A[1] \leq A[2] \leq ... \leq A[n]$
$j = n$
while $j \geq 2$ **do**
▷ Bubble up the smallest element to its correct position
  for $i = 1$ to $j - 1$ **do**
    if **then**$A[i] > A[i + 1]$
      $temp = A[i]$
      $A[i] = A[i + 1]$
      $A[i + 1] = temp$
    end if
    $j = j - 1$
  end for
end while

0.4.2 Running Time Analysis

Best case : $O(n)$

Worst case : $O(n^2)$

0.5 Selection Sort

0.5.1 Pseudo Code

INPUT : $A[1..n]$, array of integers
OUTPUT : Rearrangement of A such that $A[1] \leq A[2] \leq ... \leq A[n]$
$sorted = false$
$j = n$
while $j > 1$ and $sorted = false$ **do**
  $pos = 1$
  $sorted = true$
▷ Find the position of the largest element
  for $i = 2$ to j **do**
    if $A[pos] <= A[i]$ **then**
      $pos = i$
    else
      $sorted = false$
    end if

end for

▷ Move $A[j]$ to the position of largest element by swapping

$temp = A[pos]$

$A[pos] = A[j]$

$A[j] = temp$

$j = j - 1$

end while

0.5.2 Running Time Analysis

Best case : $O(n)$

Worst case : $O(n^2)$

0.6 Rank Sort

0.6.1 Pseudo Code

INPUT : $A[1..n]$, array of integers
OUTPUT : Rearrangement of A such that $A[1] \leq A[2] \leq ... \leq A[n]$
for $j = 1$ to n **do**
    $R[j] = 1$
end for
▷ Rank the n elements in A into R
for $j = 2$ to n **do**
    for $i = 1$ to $j - 1$ **do**
        if $A[i] \leq A[j]$ **then**
            $R[j] = R[j] + 1$
        else
            $R[i] = R[i] + 1$
        end if
    end for
end for
▷ Move to correct place in $U[1..n]$
for $j = 1$ to n **do**
    $U[R[j]] = A[j]$
end for
▷ Move the sorted entries into A
for $j = 1$ to n **do**
    $A[j] = U[j]$
end for

0.6.2 Running Time Analysis

Best case : $O(n^2)$

Worst case : $O(n^2)$

0.7 Merge Sort

0.7.1 Pseudo Code

```
INPUT : A[1..n], array of integers
OUTPUT : Rearrangement of A such that A[1] ≤ A[2] ≤ ... ≤ A[n]
procedure MERGE(A, p, q, r)
    n1 = q − p + 1
    n2 = r − q
    A1[n1 + 1] = ∞
    A2[n2 + 1] = ∞
    i = 1
    j = 1
    for k = p to r do
        if A1[i] ≤ A2[j] then
            A[k] = A1[i]
            i = i + 1
        else
            A[k] = A2[j]
            j = j + 1
        end if
    end for
end procedure
procedure MERGE-SORT(A, p, r)
    if p < r then
        q = (p + r)/2
        MERGE-SORT (A, p, q )
        MERGE-SORT (A, q + 1, r)
        MERGE (A, p, q, r)
    end if
end procedure
```

0.7.2 Running Time Analysis

Best case : $O(n \log n)$

Worst case : $O(n \log n)$

0.8 Quick Sort

0.8.1 Pseudo Code

```
INPUT : A[1..n], array of integers
OUTPUT : Rearrangement of A such that A[1] ≤ A[2] ≤  ≤ A[n]
procedure PARTITION(A, low, high)
    pivot = A[high]                                  ▷ Pivot
    i = low − 1
    for int j = low to high do
        if A[j] <= pivot then
            i + +
            exchange A[i] with A[j]
        end if
        exchange A[i + 1] with A[high]
        return i + 1
    end for
end procedure
procedure QUICK-SORT(A, low, high)
    if low < high then
        pivot = PARTITION (A, low, high)
        QUICK-SORT (A, low, pivot)
        QUICK-SORT (A, pivot + 1, high)
    end if
end procedure
```

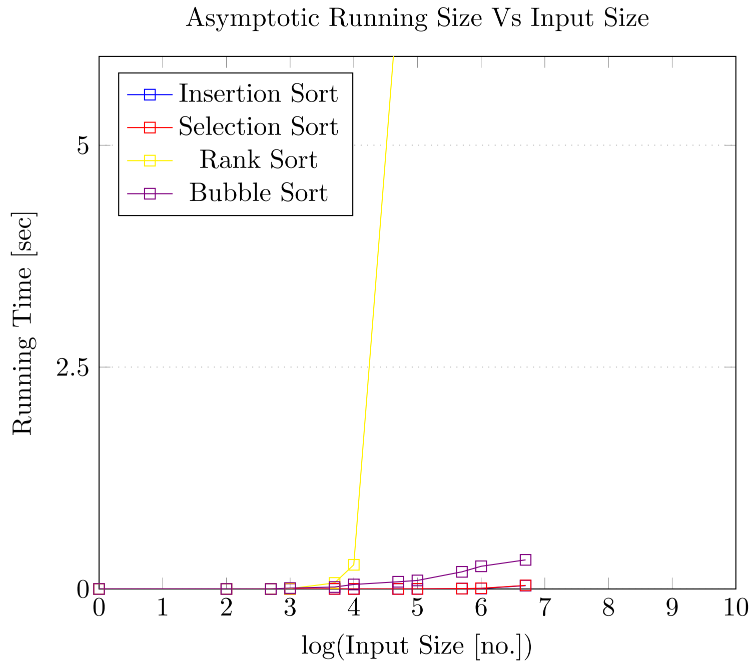
0.8.2 Running Time Analysis

Best case : $O(n \log n)$

Worst case : $O(n^2)$

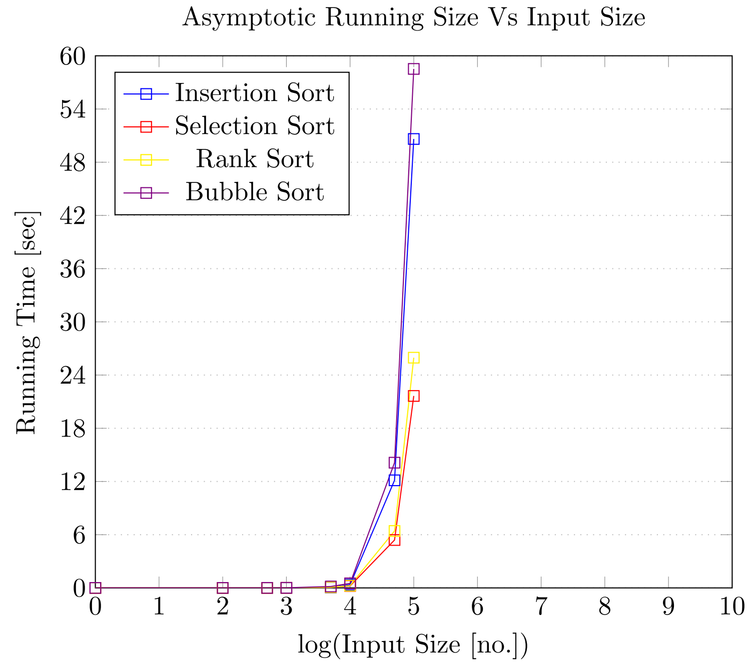
0.9 Plots of "Asymptotic Running Size Vs Input Size" for different algorithms

0.9.1 Comparison Algorithms (Sorted Input)



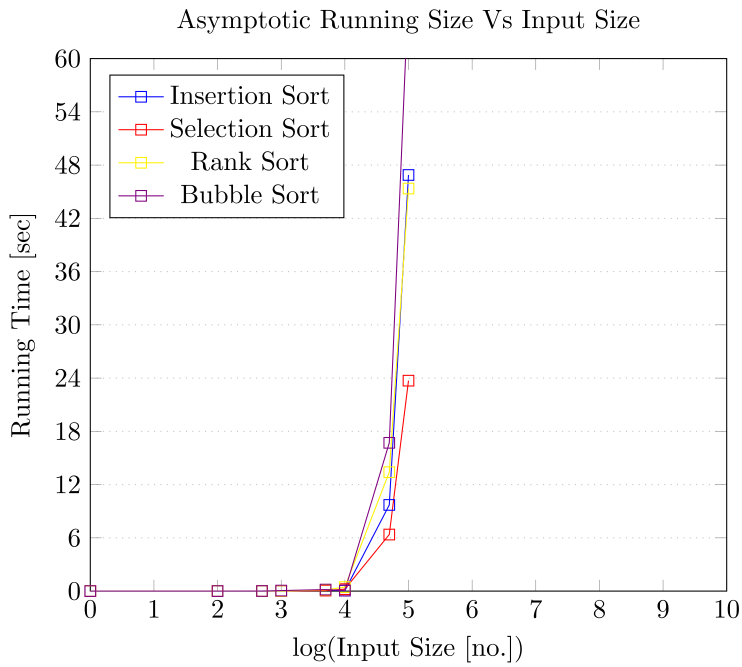
Time Complexity : *Rank > Selection > Bubble > Insertion*

0.9.2 Comparison Algorithms (Reverse Sorted Input)



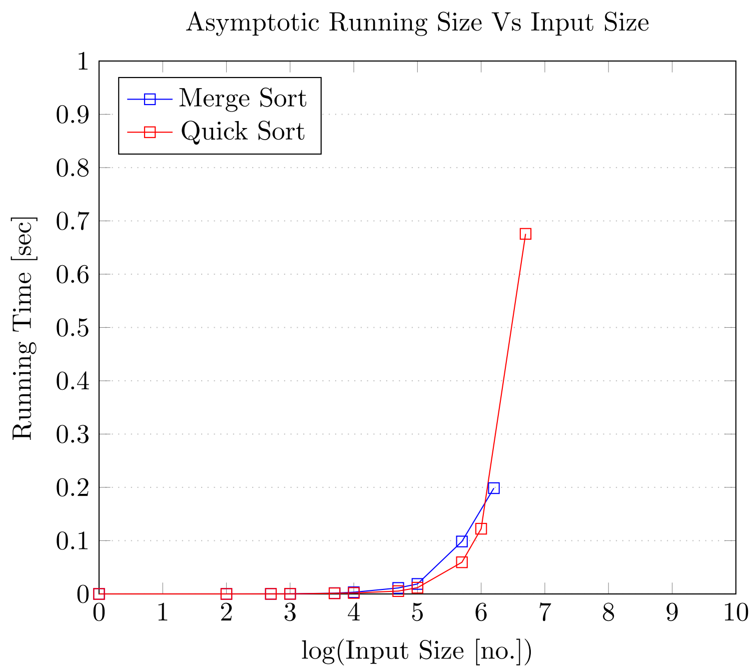
Time Complexity : *Bubble > Insertion > Rank > Selection*

0.9.3 Comparison Algorithms (Random Input)

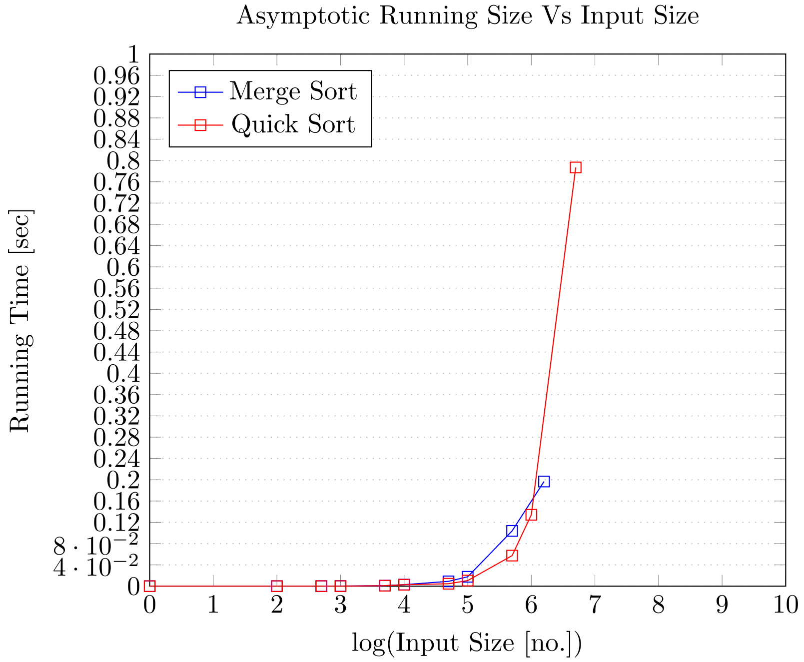


Time Complexity : *Bubble* > *Insertion* > *Rank* > *Selection*

0.9.4 Divide and Conquer Algorithms (Sorted Input)



0.9.5 Divide and Conquer Algorithms (Reverse Sorted Input)



0.9.6 Divide and Conquer Algorithms (Random Input)