Indian Institute of technology, Guwahati,

Department of Computer Science and Engineering

Data Structure Lab: (CS210) **Assignment: 1**

Date: 11th August, 2016.

Implement first three in Lab hour and submit rest by 17th August, 2016.

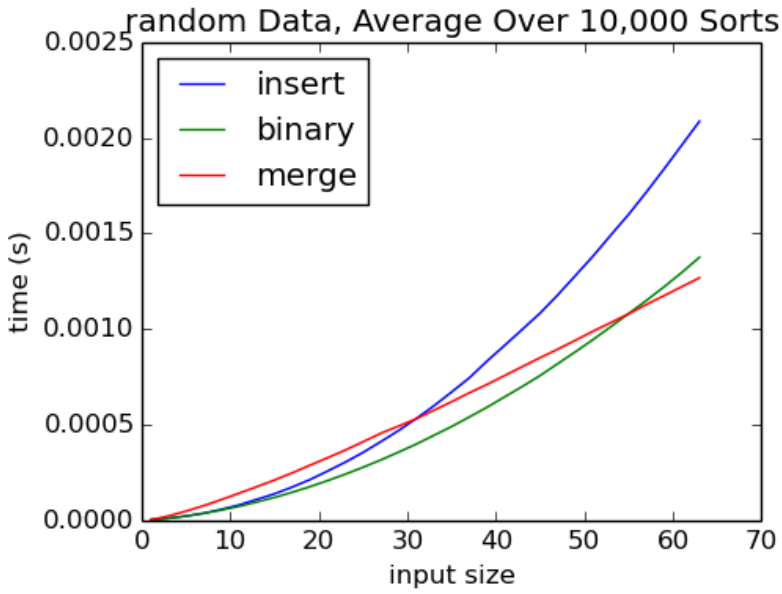
1. Write Insertion sort algorithm.

2. Write the binary search version of the insertion sort. In binary search version, binary search will be used to find the proper position of for x[k] in the sorted file of x[0], x[1[, …, x[k-1].

3. Write a function which will generate the array of n random elements for given input size n. Call above two functions with the randomly generated array. In each of the above cases, count the number of comparisons. Generate a graph showing the number of comparisons in each case.

4. Write Merge sort algorithm.

5. The merge sort is actually slower than simple $O(n^2)$ sorts for small input sizes. The following figure was created by timing merge sort, insertion sort, and binary insertion sort on small randomly ordered lists from size 2 to size 64. As you can see, binary insertion sort is the fastest of the three algorithms until around n = 55. At that point, merge sort becomes faster and it remains faster for all larger inputs.



As reminder, the following pseudocode describes the overall logic of the merge sort Algorithm:

```
merge_sort(sub-list)
    If sub-list is has more than one entry:
        Recursively merge_sort the left half
        Recursively merge_sort the right half
        Merge the two sorted halves.
```

This logic recursively splits the original list into smaller and smaller sub-lists until the recursion bottoms out at lists of size one. This means that every time a large list is sorted, there are many recursive calls to merge sort that have small input sizes. In light of the figure above, that approach doesn't make much sense: merge sort is not a competitive sorting algorithm on small inputs. It would make more sense to recursively break the input into smaller and smaller pieces until some threshold is reached, and then switch strategies to a sorting algorithm that is more efficient on those small inputs.

The following pseudocode describes this alternate approach:

```
merge_sort(sub-list)
    If sub-list is has fewer than MERGE_SORT_THRESHOLD entries:
        Sort the sub-list with binary insertion sort.
    Otherwise:
        Recursively merge_sort the left half
        Recursively merge_sort the right half
        Merge the two sorted halves.
```

Choosing an appropriate value for MERGE_SORT_THRESHOLD requires some experimentation. One of the requirements of this assignment is that you select an appropriate threshold value and provide data to justify your choice.

6. Let A and B be two sequences of n integers each. Given an integer m, describe an O (n log n) time algorithm for determining if there is an integer a in A and b in B such that m = a + b.

7. Let S be a sequence of n elements. An inversion in S is a pair of elements x and y such that x appears before y in S but x > y. Write an algorithm running in O (n log n) for determining the number of inversions in S.