Lab-05 Time complexity for Sorting Algorithms

Task Date: Monday, 07 November 2016
Total Marks: 70
Average expected time for completion: 3 hours

Instructions:

- Do not consult solution and descriptions of your code with your peers.
- In case of cheating and plagiarism a grade F will be awarded to student in particular course.
- Indent your code properly.
- Implement the task in programming language C++.
- Good Luck☺.

Task-1

Implement the following functions [20]

```
Find sum of first n integers and its time complexity.
How to find sum of First n integers in O(1).
Find value x in an array and compute time complexity of algorithm.
Multiply two matrices and compute the time complexity.
```

Task-2

Sort the given arrays using following sorting algorithms and compute time complexity for each. [50]

Note: Description of each algorithm is given at the end task.

```
Bubble Sort
Selection Sort
Insertion sort
Quick Sort
Merge Sort
```

Main function has following data members:

```
const int N=16; //size of array
int start; //starting index of array
int end; //last index of array.
int arr[16]={5,14,10,27,45,30,50,7,20,40,37,43,10,14,50,53};
int arr1[16]={1,2,3,5,7,11,13,17,19,21,23,19,31,37,39,41};
```

```
int arr2[16]={30,28,26,24,22,20,18,16,14,12,10,8,6,4,2,1};
```

Function has following prototypes:

```
void computeSum(int n)
void findElement(int a[],int x)
void matricesMultiplication()
void bubbleSort(int a[], int N)
void selectionSort(int a[], int N)
void insertionSort(int arr[], int N)
void quicksort(int a[], int p, int r)
void mergesort(int a[], int p, int r)
```

Bubble Sort compares all the element one by one and sort them based on their values.
It is called Bubble sort, because with each iteration the smaller element in the list bubbles up towards the first place, just like a water bubble rises up to the water surface.
Sorting takes place by stepping through all the data items one-by-one in pairs and comparing adjacent data items and swapping each pair that is out of order.
Worst Case Time Complexity: O(n2)
Best Case Time Complexity: O(n2)
Average Time Complexity: O(n2)

Selection sort is conceptually the simplest sorting algorithm.
This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted.
Worst Case Time Complexity: O(n2)
Best Case Time Complexity: O(n2)
Average Time Complexity: O(n2)

Insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.
Insertion sort is only good sort small arrays. Insertion sort is stable sort
In insertion sort each element is compared to element behind of it till starting element of array. If the element is smaller then swapping is perform.
Worst Case Time Complexity : O(n2)
Best Case Time Complexity : O(n)
Average Time Complexity : O(n2)

Quick Sort, as the name suggests, sorts any list very quickly.
Quick sort is not stable search, but it is very fast and requires very less aditional space.
It is based on the rule of Divide and Conquer rule.
This algorithm divides the list into three main parts :
Elements less than the Pivot element
Pivot element
Elements greater than the pivot element
Hnece after Quick Sort, pivot will be set at its position, with all the elements smaller to it on its left and all the elements larger than it on the right.
Quick sort is not a stable sorting technique, so it might change the occurence of two similar elements in the list while sorting.
Quick Sort is quite fast, and has a time complexity of O(n log n).
Worst Case Time Complexity : O(n2)
Best Case Time Complexity : O(n log n)
Average Time Complexity : O(n log n)

Merge Sort follows the rule of Divide and Conquer. But it doesn't divides the list into two halves. In merge sort the unsorted array is divided into N subarrays, each having one element, because an array of one element is considered sorted. Then, it repeatedly merge these subarrays, to produce new sorted subarrays, and at lasts one sorted array is produced.
It is also a stable sort, the "equal" elements are ordered in the same order in the sorted list.
Worst Case Time Complexity : O(n log n)
Best Case Time Complexity : O(n log n)
Average Time Complexity : O(n log n)