Endsemester Examination
IT WorkShop

Submission Deadline: 02:00 AM, April 29, 2016 (You get 13 hours 30 minutes)

Marks of this assignment: 33
Marks assigned to viva: 12

Please read the following instructions carefully.

- The submission deadline is **firm**. Late submissions will be awarded 0. Submission details will be mailed soon.
- Cheating/plagiarism in any form is strictly forbidden. If caught, matter will be reported to higher authorities with the least punishment of decrementing the grade by two.
- The assignment is to be done in groups but members may get different marks depending on their performance in viva.
- You have to implement the algorithms given to you. Not any other algorithm with the same name that you can find somewhere else.
- All submissions must have names as Rollnumber1_Rollnumber2_Rollnumber3.zip (where Rollnumbers are the actual roll numbers of group members). 2 Marks will be deducted for not following this particular instruction.
- The assignment has to be done on Linux.

==========xxxxxx==========

1 Introduction to the problem

You are given five algorithms (4 are given explicitly and you have to write one) that sort an array A of integers. Before you move further, try to understand each of these algorithm and how they work. Once you have understood, implement each algorithm as a function in the same program. Since all five functions sort an array of integers, they all must make comparisons between two values. Your program should compute the total number of comparisons made between the elements of the array for each function until it outputs the sorted array. For example, if input array is {3,4,1} then output array is {1,3,4}. In the process, the three elements have to be compared with each other to find the correct ordering. The total number of such comparisons determine the time taken by the program to output the sorted array. We will use such a count as a measure of efficiency of the mentioned algorithms.

To compute the total number of comparisons, define variables corresponding to each algorithm, e.g., `int count_bubble` to compute the number of comparisons made by the function written using bubble sort algorithm. Similarly, a variable for all the functions (so 5 in total). Note that total here refers to the number of comparisons made by just one algorithm or one function in your program, that is why you have five variables for five functions. You may make function calls to each of these functions from main() as follows:

1. One by one so that only one sorting algorithm is used in one execution, i.e., you keep just one call active in the main and keep the rest of the function calls commented. This can give you the time taken by your program to run just one algorithm.
2. All five at once in main() so that all the functions in your program are run in just one execution. This can give the total number of comparisons for each function in just one execution of the program. Note however, that each function compares the elements in a different manner. Hence, each function must maintain its own copy of the array.

Your program should be able to handle large size of array (millions). Your goal is to find the behavior of these algorithms and the total number of comparisons made by them. You have to draw graphs to compare them. Think hard how to achieve it. Produce as many interesting results as you can. On X axis you can have the number of elements in the array and on Y axis the total number of comparisons to sort the array.

2 Preparing the data

2.1 Identify the statements that compare

Analyze the total number of comparisons made by each algorithm for a given input array A. You may do so by declaring a *global* variable `int` count_function individually for each function. E.g., for bubble sort it may be called as count_bubble and this variable should be incremented by writing count_bubble++ between lines 2 and 3 of the algorithm shown. It is so because the number of comparisons made in this algorithm is given by the number of times "if" statement is executed. Identify the comparing statements for the other functions too. The corresponding variable must count the total number of comparisons made by that algorithm.

2 Marks

2.2 Time taken by your program

On Linux, command "time ./a.out" will give you the amount of time taken by your program to run. Document these times for each algorithm for different size of arrays, i.e., check which algorithm runs faster on the same input. Note that the notion of faster and slower makes sense only for the same input (i.e., the same array). (Why?)

2 Marks

2.3 Number of comparisons depend on array A, its size n and the algorithm used

Find relation between the number of comparisons made and the size of the input array A for each function. For array A of size n, we define the number of comparisons made by algorithm algo as $C_{algo}(A)$. For example, if you are counting it for bubble sort, then the maximum value of $C_{bsort}(A) = \frac{n(n-1)}{2}$ (Why?). But in general it may vary. Solve the following exercise for a better insight.

Toy exercise: Compute $C_{bsort}(A)$ if (a). A={4,12,14,13,8}, (b). A={4,8,12,13,14} (c). A={14,13,12,8,4}

2 Marks

```
BUBBLESORT(A)
for i ← 1 to length[A]
    do for j ← length[A] downto i + 1
        do if A[j] < A[j - 1]
            then exchange A[j] ↔ A[j - 1]
```

Figure 1: Bubble sort: A is the array to be sorted. The goal is to compute $C_{bsort}(A)$ for each A that is given as input.

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    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[i + 1] ← A[i]
              i ← i - 1
       A[i + 1] ← key
```

Figure 2: Insertion sort ($A[1 \quad j-1]$ means $A[1, \cdots, j-1]$): A is the array to be sorted. Compute $C_{isort}(A)$.

```
MERGE(A, p, q, r)
n₁ ← q - p + 1
n₂ ← r - q
create arrays L[1   n₁ + 1] and R[1   n₂ + 1]
for i ← 1 to n₁
    do L[i] ← A[p + i - 1]
for j ← 1 to n₂
    do R[j] ← A[q + j]
L[n₁ + 1] ← ∞
R[n₂ + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
          then A[k] ← L[i]
               i ← i + 1
          else A[k] ← R[j]
               j ← j + 1
```

```
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)
```

Figure 3: Algorithm for Merge sort: (p,q,r represent indexes in array A). $L[1 \quad n_1+1]$ means $L[1, \cdots, n_1+1]$. Similarly for array R. Compute $C_{msort}(A)$.

```
PARTITION(A, p, r)
x ← A[r]
i ← p - 1
for j ← p to r - 1
    do if A[j] ≤ x
          then i ← i + 1
               exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

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

Figure 4: Quick sort: (p,r represent indexes in array A). Compute $C_{qsort}(A)$.

2.4 Write selection sort algorithm on your own

In selection sort, first find the smallest element in an array and exchange that with the first element of the array. Then find the second largest element and exchange that with the second element of the array. Continue this way until n-1. Write this algorithm in the format of other algorithms given to you. Write a function to implement selection sort and repeat the exercise done with all the algorithms presented before.
Help: Use Latex packages algorithm.sty and algorithmic.sty for writing it in your report.

2+2 Marks

2.5 Making library of sorting functions

You can do all the above task without making a library. This task asks you to create a library and use that library in your program to do the same things as above, i.e., instead of defining functions in main file you include the library created by you in the header. Ideally, you should be able to integrate it with the libraries on your system.

5 Marks

2.6 Plot the results

A better way to find the relations between the comparison made by two algorithms is by plotting curves. So run the program for a variety of the size of input and then plot. For each function, if size of array A is n, take many different values of n, like n=10,20,30...,100,...1000,....100000,...1000000. Generate A randomly. Note however, that you have to run all the functions for same A (Refer to the toy exercise given before). Corresponding to each n, there is a value $C_{algo}(A)$. Store both these values in a file. (You will have to extensively make use of shell scripting here. You cannot keep running manually the same program again and again thousands of times!! You will have to redirect the data to files too. Then use these files to generate plots using GNUplot. You will have to work hard to make your presentation look good.) Take enough number of points to give you a meaningful graph. On X-axis you will have the size of array,i.e., n, on Y-axis you will have the number of comparisons, i.e., $C_{algo}(A)$ (you may also plot time in the same fashion).

5 Marks

Make conclusions on which of the sorting algorithms is the best and which one is the worst for the set of inputs chosen by you. This is done in terms of the comparisons made. The best algorithm will make the least comparisons and the worst algorithm will make the most comparisons for the **same** n **and A**. Else comparison does not make sense (why?) Does it matter if you input different array (of the same size) each time?

2 Marks

3 Submitting the report

You have to submit a report that is prepared in Latex using the IEEEtran.cls class. The report can have at most 3 pages but keeping it around of 2 pages is preferable. The content of the report has to be built around the topics mentioned above. Better explanations in the report fetch you more marks. There are marks assigned to the explanation of all the concepts above. Your report should include all the figures that you have drawn using GNUplot and explain them well. At the same time you cannot exceed the space limit kept for the report (at most 3 pages in two column format).

4 Marks

4 Presenting your work

You should create a presentation of no more than 4 slides (including EVERYTHING) using the beamer class of Latex. You are supposed to present your work during the viva through the presentation. Most of the questions will be from what you have done in the project. So all the members of the group are required to participate equally in the activity, else there is high chance that the same team members may get different marks.

3 Marks

5 Running GDB and Matlab

During viva, all of you will be asked to run a program of yours using GDB. You should be able to use GDB up to some basic level. Some basic questions on Matlab will be asked too.

2+2 Marks

6 Viva

Apart from Section 5, the viva will be about what you have done in section 2, 3 and 4.

12 Marks