

CSE221_LabAssignment02_Fall2022

Submission Guidelines :

1. You can code all of them either in Python or Java. But you should choose a specific language for all tasks.
2. The first three tasks are to be completed in the lab and task 4 is homework.
3. For each task write separate python files like task1.py, task2.py and so on.
4. For each problem, take input from files called "inputX.txt", and output at "outputX.txt", where X is the task number. So, for problem 2, the input file is basically this, "input2.txt". Same for output.
5. For each task include the input files (if any) in the submission folder.
6. Finally zip all the files and rename this zip file as per this format :
LabSectionNo_ID_CSE221LabAssignmentNo_Summer2022.zip. [Example :
LabSection01_21101XXX_CSE221LabAssignment02_Summer2022.zip]
7. Don't copy from your friends
8. You **MUST** follow all the guidelines, naming/file/zippping convention stated above.
Failure to follow instructions will result in straight 50% mark deduction.

Problem Descriptions

The following tasks need to be solved using Sorting Algorithms. Sorting algorithm is an algorithm that is used to arrange data in certain order, i.e- either ascending or descending order. You have learned several sorting algorithms such as bubble sort, insertion sort, selection sort, quick sort, merge sort etc. The problems below are related or similar to some of the sorting algorithms you learned in class. Read the task descriptions carefully and implement the algorithm using either Java or Python. The output format **MUST** match exactly as shown in each task.

Task : 1 [5 Marks]

Suppose you are given a task to rank the students. You have gotten the marks and id of the students. Now your task is to rank the students based on their marks using only **insertion sort** from highest to lowest.

Here is the pseudocode for insertion sort for **ascending order**. You need to change it for **descending order**.

```
for i ← 0 to n - 1
    temp ← A[i + 1]
    j = i
    while j >= 0
        if(A[j] > temp)
            A[j + 1] ← A[j]
        else
            break
        j = j - 1
    end for
    A[j + 1] ← temp
end for
```

Implement this pseudocode to complete your task.

First line of the input file will contain an integer **N**. The next line will contain N number of id of the students. The next line will contain the N number of the marks of corresponding students. Output will be ranking the id based on their marks (descending order).

Input 1: 5 111 452 343 224 125 40 50 20 10 10	Input 2: 6 1 2 3 4 5 6 50 60 80 20 10 30
Output 1: 452 111 343 224 125	Output 2: 3 2 1 6 4 5

Task : 2 [5 Marks]

Although Insertion Sort has an $O(n)$ time in best case but in the worst case it is bad as selection or bubble. Merge Sort, a divide and conquer algorithm, however has a terrific worst case time complexity.

i) Derive the worst case time complexity of Merge Sort using either tree method or substitution. [2]

ii) Given a bunch of numbers in the input file, write the code of Merge Sort. The sorting algorithm pseudocode is given below. [3]

MERGE (A, p, q, r)

$n1 \leftarrow q - p + 1$

$n2 \leftarrow r - q$

Create arrays $L[1 \dots n1 + 1]$ and $R[1 \dots n2 + 1]$

FOR $i \leftarrow 1$ to $n1$

 DO $L[i] \leftarrow A[p + i - 1]$

FOR $j \leftarrow 1$ to $n2$

 DO $R[j] \leftarrow A[q + j]$

$L[n1 + 1] \leftarrow \infty$

$R[n2 + 1] \leftarrow \infty$

$i \leftarrow 1$

$j \leftarrow 1$

FOR $k \leftarrow p$ TO r

 DO IF $L[i] \leq R[j]$

 THEN $A[k] \leftarrow L[i]$

$i \leftarrow i + 1$

 ELSE $A[k] \leftarrow R[j]$

$j \leftarrow j + 1$

MERGE - SORT (A, p, r)

if $p < r$ // Check for base case

$$q = (p + r)/2$$

MERGE – SORT (A, p, q)

MERGE – SORT (A, q + 1, r)

MERGE (A, p, q, r)

Take help from pseudocode or use your way to complete the task.

First line will contain N . The next line will contain N number of elements. The output will contain a sorted N number of elements (ascending order).

Input 1: 5 5 -1 10 2 8	Input 2: 6 10 20 3 40 5 6
Output 1: -1 2 5 8 10	Output 2: 3 5 6 10 20 40

Task : 3 [5 Marks]

Merge Sort is an out of place algorithm. A solution to this problem is the Quick Sort algorithm.

i) On the same set of inputs used in Task 2, implement Quick Sort. You have to write the code for the partition function as well. The pseudocodes are given below.

*# low → Starting index, high → Ending index */*

quickSort(arr[], low, high)

if (low < high)

pi is partitioning index, arr[pi] is now at right place

pi = partition(arr, low, high)

quickSort(arr, low, pi – 1) # Before pi

quickSort(arr, pi + 1, high) # After pi

}

This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot

partition (arr[], low, high)

pivot (Element to be placed at right position)

pivot = arr[high];

i = (low - 1) # Index of smaller element and indicates the

right position of pivot found so far

for j in range (low, high- 1):

If current element is smaller than the pivot

if (arr[j] < pivot){

i=i+1; # increment index of smaller element

swap arr[i] and arr[j]

}

}

swap arr[i + 1] and arr[high])

return (i + 1)

}

ii) Implement an algorithm “findK” that uses the “partition” algorithm to find the kth

(smallest) element from an array without sorting. E.g. for the array in our example, the 5th element will be “9”

Input:

The array: 1 3 4 5 9 10 10

K=5

K=7

K= 2

Output:

9

10

3

Task 4 [10 Marks Homework]

You are assigned the task to automate the patient management system for a hospital. Three categories of patients visit the hospital: 1. Severely ill 2.

Moderately ill 3. Regular check up

Your system must be able to list the patients according to the seriousness, 1 being the most serious and 3 being the least. Moreover, the system must be able to handle incoming patients while doctors are checking other patients.

Solving this problem requires knowledge of Queue which you have learnt in CSE220.

Input: The input file will contain names of the patients as strings with a priority (1, 2 or 3) associated with each name and an operation called “see doctor.” When the “see doctor” function will be called one patient from the queue, of course the patient who came first with seriousness 1, will be served first.

ABC 3

EWQ 3

SDF 2

KLM 1

see doctor

see doctor

FDS 1

OPN 3

TYU 3

see doctor

XCV 2

see doctor

see doctor

Output: Every time the “see doctor” function is called, erase the correct patient from the queue and print the name.

Methodology: Create a function called **enqueue**(name of the patient) which will insert the name in the queue (use a list). Create a function called **seeDoctor**() which will basically deque the correct patient from the queue. Create a function named **printQueue**() which will print all the patients in the queue. You may create additional functions needed to solve the problem. Use a **list** as the queue.

- a) You must build this system in 2 ways. Use the **bubble sort** algorithm everytime a new patient comes in, where you sort the patients according to the seriousness and delete when seeDoctor method is called.
- b) The second way is to use **heap sort**. Everytime a patient comes in you heapify (swim up) and sink when the seeDoctor method is called.
- c) Do you see any difference in output when you call the seeDoctor() function ? If yes, what do you think is the reason? If you don't see any difference do not panic. As long as the correct patient is sent to the doctor your system is correct.
- d) You then compare the time complexity of the 2 techniques by plotting a line graph input in the x axis and time taken in the y axis. See lab 1 to solve.