National Institute of Technology Calicut
Department of Computer Science and Engineering
B. Tech. (CSE) – Third Semester
CS2092D: Programming Laboratory
Assignment –2 PART–B

Submission deadline (on or before):
18 th September 2019, 10:00:00 PM

Policies for Submission and Evaluation

You must submit your assignment in the moodle (Eduserver) course page, on or before the submission deadline. Also, ensure that your programs in the assignment must compile and execute without errors in Athena server. During evaluation your uploaded programs will be checked in Athena server only. Failure to execute programs in the assignment without compilation errors may lead to zero marks for that program.

Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straightaway awarded zero marks for this assignment and considered by the examiner for awarding F grade in the course. Detection of ANY malpractice regarding the lab course will also lead to awarding an F grade.

Naming Conventions for Submission

Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar or .tar.gz). The name of this file must be ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>.zip (For example: ASSG2B_BxxyyyyCS_LAXMAN.zip). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive. The source codes must be named as :

ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME> _<PROGRAM-NUMBER>.<extension>

(For example: ASSG2B_BxxyyyyCS_LAXMAN_01.c).

If you do not conform to the above naming conventions, your submission might not be recognized by some automated tools, and hence will lead to a score of 0 for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

Violations of academic integrity will be severely penalized. Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign an F grade in the course. The department policy on academic integrity can be found at:

http://minerva.nitc.ac.in/cse/sites/default/files/attachments/news/Academic-Integrity_new.pdf

General Instructions

Programs should be written in C language and compiled using the C compiler in Linux platform. Invalid input should be detected and suitable error messages should be generated. Sample inputs are just indicative. Please do the programs in your free time either from System Software Lab (SSL) / Network Systems Lab (NSL) / Hardware & Embedded Systems Lab (HL), when the lab is not used for regular lab hours or do the programs using your own computer. Even if the programs work in your own computer, there is a chance that they may not work properly in the computers in SSL / NSL/ HL, due to some compatibility issues of the C compiler or the machine. Hence, before the evaluation day, check that your programs are ready for execution in the computers in NSL/SSL/HL.

Evaluation of questions from the following questions will be conducted on **19 th September 2019** Thursday 2.00 pm - 5.00 pm.

NOTE

For all the questions in this assignment, input should be read from a file and output should be written to a file.

Questions

1. Given an array ***X*** of integers of size ***n***, write a C program to count the number of **inversions** in the array using **merge sort**. If ***i*** < ***j*** and ***X[i]*** > ***X[j]***, the (***X[i]***, ***X[j]***) is called an inversion, where ***i*** and ***j*** are indices of the array.

Your program should implement the following functions.

main() - repeatedly reads a character ***'r', 'i'*** or ***'d'*** and calls the sub-functions appropriately until character ***'t'*** is read.

read(X, *n*) – read ***n*** integers and store it in an array ***X***.

inversion(*X, n*) – finds the inversions in the array ***X*** using merge sort where ***n*** is the size of array ***X***.

display(*X, n*) – display ***n*** elements of the array ***X***.

INPUT FORMAT:

First line contains a character from {***'r', 'i', 'd' ,'t'***} followed by zero or one integer.

• Character ***'r'*** is followed by an integer ***n***, the size of the input array. In this case, the second line gives the list of ***n*** integers separated by a space.

• Character ***'i'*** is to find the inversions in the array using merge sort.

• Character ***'d'*** is followed by the integer ***n***, the size of the output array. In this case, the next line print the list of ***n*** integers separated by space.

• Character ***'t'*** terminates the program.

OUTPUT FORMAT:

For option ***'i'***, display the number of inversions in the array.

For option ***'d'***, print ***n*** elements of array separated by space in a new line.

Sample Input:

```
r 5
2 4 1 3 5
d 5
i
t
```

Sample Output:

```
2 4 1 3 5
3
```

2. Given an array ***X*** of integers of size ***n***, write a C program to find a Noble integer in it. An integer **x** is said to be Noble in the array, if the number of integers greater than ***x*** is greater than or equal to ***x***. If there are many Noble integers, return the maximum among them. If there is no such integers, then return -1.

Your program should implement the following functions.

main() – repeatedly reads a character ***'r', 's'*** or ***'d'*** and calls the sub-functions appropriately until character ***'t'*** is entered.
read(*X*, *n*) – read ***n*** integers and store it in an array ***X***.

search(*X, n*) – finds Noble integer in the array ***X*** where ***n*** is the size of array ***X***.

display(*X, n*) – display ***n*** elements of the array ***X***.

INPUT FORMAT:

First line contains a character from {***'r', 's', 'd' ,'t'***} followed by zero or one integer.

• Character ***'r'*** is followed by an integer ***n***, the size of the input array. In this case, the second line gives the list of ***n*** integers separated by a space, to be read into the array ***X***.

• Character ***'s'*** is to find the Noble integer in the array.

• Character ***'d'*** is followed by the integer ***n***, the size of the output array. In this case, the next line print the list of ***n*** integers separated by space.

• Character ***'t'*** terminates the program.

OUTPUT FORMAT:

For option ***'s'***, display the Noble integer in the array.

For option ***'d'***, print ***n*** elements of array separated by space in a new line.

Sample Input:
```
r 4
7 3 16 10
d 4
s
t
```
Sample Output:
```
7 3 16 10
3
```

3. Given two arrays of integers ***X*** and ***Y*** of size **m** and **n** respectively, write a C program to find whether ***Y*** is a subset of ***X*** or not. Both the arrays may or may not be in sorted order. It may be assumed that the elements in both arrays are distinct.
Your program should implement the following functions.

main() – repeatedly reads a character ***'r'*** or ***'c'*** and calls the sub-functions appropriately until character ***'t'*** is entered.

read(*X*, *m*) – read ***m*** integers, and store it in an array ***X***.

checkSubset(*X, m, Y, n*) – checks the array ***Y*** of size ***n*** is a subset of the array ***X*** of size ***m***.

INPUT FORMAT:

First line contains a character from {***'r', 'c', 't'***} followed by zero or one integer.

• Character ***'r'*** is followed by an integer which represents the size of the input array. In this case, the second line gives the list of integers separated by a space, to be read into the array.

• Character ***'c'*** is to find whether the given second array is a subset of the given first array.

• Character ***'t'*** terminates the program.

OUTPUT FORMAT:

For option ***'c'***, display either ***true*** or ***false***. If the second array is a subset of the first array, then ***true*** will be displayed otherwise ***false***.

Sample Input:
```
 r 6
11 1 13 21 3 7
r 4
11 3 7 1
c
t
```
Sample Output:
```
true
```

4. Given an array ***A*** of ***n*** characters, sort the array in ascending order using heap sort. Assume that $1 \leq n \leq 100$.

Your program should implement the following functions:

> **main()** – repeatedly reads a character ***'r', 's'*** or ***'d'*** and calls the sub-functions appropriately until character ***'t'*** is entered.
>
> **read(*A*, n)** – read ***n*** characters and store it in an array ***A***.
>
> **heapSort(*A, n*)** – perform heap sort on array ***A*** of size ***n.*** Use display() function to print the sorted array at the end of the function.
>
> **display(*A, n*)** – display ***n*** elements of the array ***A***.

INPUT FORMAT:

First line contains a character from {***'r', 's', 'd' ,'t'***} followed by zero or one integer.

• Character ***'r'*** is followed by an integer ***n***, the size of the input array. In this case, the second line gives the list of ***n*** characters separated by a space, to be read into the array.

• Character ***'s'*** is to sort the given array using heap sort.

• Character ***'d'*** is followed by the integer ***n***, the size of the output array. In this case, the next line print the list of ***n*** integers separated by space.

• Character ***'t'*** terminates the program.

OUTPUT FORMAT:

For option ***'s'***, print the output for heap sort on the given character array.

For option ***'d'***, print ***n*** elements of array separated by space in a new line.

<u>Sample Input:</u>

```
r 7
A P M N D B C
d 7
s
t
```

<u>Sample Output:</u>

```
A P M N D B C
A B C D M N P
```

5. You are given an unsorted array of integers ***X*** of size ***m***, and another integer ***k***. Write a C program to find ***k***th smallest element in the array using **maxheap**. If ***k*** > ***n***, then **k = k (modulo n)**.

Your program should implement the following functions.

main() - repeatedly reads a character ***'r', 's'*** or ***'d'*** and calls the sub-functions appropriately until character ***'t'*** is entered.

read(*X*, n) – read ***n*** integers and store it in an array ***X***.

kthSmallest(*X, n, k*) – find the ***k***th smallest element from array ***X*** of size ***n*** using ***maxheap***.

display(*X, n*) – display ***n*** elements of the array ***X***.

INPUT FORMAT:

First line contains a character from {***'r', 's', 'd' ,'t'***} followed by zero or one integer.

• Character ***'r'*** is followed by an integer ***n***, the size of the input array. In this case, the second line gives the list of space separated ***n*** integers, to be read into the array.

• Character ***'s'*** is followed by an integer ***k***, and find the ***k***th smallest element from array.

• Character ***'d'*** is followed by the integer ***n***, the size of the output array. In this case, the next line print the list of ***n*** integers separated by space.

• Character ***'t'*** terminates the program.

OUTPUT FORMAT:

For option ***'s'***, print the ***k***th smallest element in the given array.

For option ***'d'***, print ***n*** elements of array separated by space in a new line.

Sample Input:

```
r 6
7  10  4  3  20 15
d 6
s 3
t
```

Sample Output:

```
7  10  4  3  20 15
7
```

6. Consider an unsorted array ***X*** of integers with size ***n*** and another integer ***k***. Write a C program to find ***k***th largest element in the given integer array using **partition()** method of **quicksort**. If ***k*** > ***n***, then **k = k (modulo n)**.

Your program should implement the following functions.

main() - repeatedly reads a character ***'r', 's'*** or ***'d'*** and calls the sub-functions appropriately until character ***'t'*** is entered.

read(*X, n*) – read ***n*** integers and store it in an array ***X***.

kthLargest(*X, m, k*) – find the k^{th} largest element from array ***X*** of size ***m*** using **partition()** method of **quicksort**.
display(*X, n*) – display ***n*** elements of the array ***X***.

INPUT FORMAT:

First line contains a character from {***'r', 's', 'd' ,'t'***} followed by zero or one integer.
• Character ***'r'*** is followed by an integer ***n***, the size of the input array. In this case, the second line gives the list of space separated ***n*** integers, to be read into the array.
• Character ***'s'*** is followed by an integer ***k***, and find the k^{th} largest element from array.
• Character ***'d'*** is followed by the integer ***n***, the size of the output array. In this case, the next line print the list of ***n*** integers separated by space.
• Character ***'t'*** terminates the program.

OUTPUT FORMAT:

For option ***'s'***, print the k^{th} largest element in the given array.
For option ***'d'***, print ***n*** elements of array separated by space in a new line.

Sample Input
```
r 6
7 10 4 3 20 15
d 6
s 3
t
```
Sample Output
```
7 10 4 3 20 15
10
```

7. Given an array ***A*** of ***n*** integers, divide the elements of this array into buckets on the basis of the number of one bits in its binary representation. Print the elements of each bucket in a newline. The buckets should appear in the output in ascending order, i.e the bucket that stands for lesser number of 1's in binary representation should appear before any other bucket which stands for higher number of 1's in binary representation. The elements of each bucket should appear in ascending order too. That is if two integers appear in the same bucket, the one with the lower value should appear in the bucket list before the one with higher value. Assume that $1 \leq \mathbf{n} \leq 100$.

Your program should implement the following functions:

main() - repeatedly reads a character ***'r', 's'*** or ***'d'*** and calls the sub-functions appropriately until character ***'t'*** is entered.
read(*A, n*) – read ***n*** integers and store it in an array ***A***.
bucketSort(*A, n*) – perform bucket sort on array ***A*** of size ***n***. Use display() function at the end to print the contents of buckets.

display(*A, n*) – display ***n*** elements of the array ***A***.

INPUT FORMAT:

First line contains a character from {***'r', 's', 'd' ,'t'***} followed by zero or one integer.

• Character ***'r'*** is followed by an integer ***n***, the size of the input array. In this case, the second line gives the list of ***n*** integers separated by a space, to be read into the array.

• Character ***'s'*** is to sort the given array using bucket sort.

• Character ***'d'*** is followed by the integer ***n***, the size of the output array. In this case, the next line print the list of ***n*** integers separated by space.

• Character ***'t'*** terminates the program.

OUTPUT FORMAT:

For option ***'s'***, print the output for bucket sort. The contents inside each bucket are displayed in each line.

For option ***'d'***, print ***n*** elements of array separated by space in a new line.

Sample Input:

```
r 5
1 2 3 4 5
d 5
s
t
```

Sample Output:

```
1 2 3 4 5
1 2 4
3 5
```