Lab 1.

1. Write C code (functions/procedures and main program) to initialise and access a (doubly) linked (linear) list. Following operations must be implemented:

Insert (read an item from terminal)
Delete (read an item from terminal)
Traverse (from start) and print items
Traverse (from end, reverse order) and print items

2. Use the solution of Question 1 to implement

stack- Pop/Push
queue- insert (enQueue) and delete (deQueue)

Lab 2

1. Write code to generate random integers (in suitable range). Your output should be in a file. You may use C-library routines or any method (see e.g. https://en.wikipedia.org/wiki/List_of_random_number_generators) **10%**

2. Implement insertion sort to sort integers. Your program must read numbers from a file (say that created in 1st problem). You may "hard-code" the file name or any other method you are familiar with (e.g. command line arguments). The number of items can again be either read from terminal/file/command line argument. **20 %**

3. Implement heap sort to sort integers. Your program must read numbers from a file (say that created in 1st problem). You may "hard-code" the file name or any other method you are familiar with (e.g. command line arguments). The number of items can again be either read from terminal/file/command line argument. **35 %**

4. Use time command to find running time for programs 2 and 3 for

N=10,100,1000,10000,100 000 (1 lakh), 100 00 00 (1 Million).

Lab 3

1. Implement bucket sort, with each bucket as a queue (implemented using linked list). You are free to (re)-use code of Lab 2.

2. Implement merge sort. You are not allowed to use recursion/pointers in this question.

3. Use time command to find running time for above programs and also heap sort (of last lab) for

N=10,100,1000,10000,100 000 (1 lakh), 100 00 00 (1 Million).

Lab 4

1. Implement recursive quick sort, pivot is to be chosen at random.

2. Implement recursive merge sort. You are may use merge procedure (to merge two sorted arrays) of a previous lab.

3. Use time command to find running time for above programs and also non-recursive merge sort (of previous lab) for

N=10,100,1000,10000,100 000 (1 lakh), 100 00 00 (1 Million).

Lab 5

1 (a). Implement linear time selection algorithm. And use it to find the **median**. **40**

(b) Compare running time for random integers with faster of Merge/Heap sort for: **10**

N=10,100,1000,10000,100 000 (1 lakh), 100 00 00 (1 Million).

Please stop running a program as soon as running time for that program exceeds 3 minutes.

2 (a). Implement Strassen's Algorithm to multiply two $2_k \times 2_k 2\text{\textregistered} \times 2\text{\textregistered}$ matrices. **20**

(b) Compare running time with the usual method for multiplying two matrices (using random integers

in range 0..99) for k=4,7,9,11, 13, 15, 17 5

Lab 6

1**.** Implement $O(n^2)$ time algorithm for closest pair: find distances between each pair of points and return the one having minimum distance. **10**

2. Implement O(n log n) time algorithm for closest pair. You may resuse sorting routine(s) of earlier lab(s). **50**

3.Compare running time of above programs for 10,100,5000,25000,50000 pairs

Lab 7

1. Implement Binary Search Trees. Implement (at least) insert, search, deleteMin and delete. Try to insert 5000 random numbers

one by one. Then traverse the tree in inOrder. Finally, repeatedly find, print and delete the minimum item (for 1000 times)

2. Implement Priority Queues using r-way heaps. "r" should be a parameter (with

$r \geq 2$

$\text{\textregistered} \geq 2$). Implement (at least) insert, reduceKey and deleteMin. Try to insert 5000 random numbers (same as in Q1) one by one. Carry out deleteMin (and print) for 1000 times.

Lab 8

RB-Trees are binary search trees in which each node has a non-negative integral rank (rank is 0,1,2,...) such that

> Rank of failure nodes is zero
> Rank of parent of failure node is one
> If "x" is any node and p(x) is parent of x, then rank(x) $\leq$ rank(p(x)) $\leq$ rank(x)+1
> If g(x) is p(p(x)), the grand parent of x, then rank(x) $\leq$ rank(g(x))-1

Implement following operations on RB-trees (**ranks have to be explicitly mantained)**

(a) Search **10**

(b) Insert **20**

(c) Delete **35**

(d) Try to insert 500 random numbers one by one. Then, repeatedly find, print and delete the minimum item (for 100 times)

Lab 9

1. Implement Kruskal's Algorithm for finding Minimum Spanning Tree. This also involves implementing disjoint union-find (with path compression). Test your program for a graph (of your choice) having at least **6** vertices, number of edges should be at least twice the number of vertices. Do not hard code the graph. Your algorithm should also work (without any modification) on the graph suggested by TA. You may use sort routines of previous lab.

2. Implement Prim's Algorithm for MST using r-way heaps. You are free to use routines of Lab 8. Test your program for a graph (of your choice) having at least **6** vertices, number of edges should be at least twice the

number of vertices. Do not hard code the graph. Your algorithm should also work (without any modification) on the graph suggested by TA. Choose

$r=2+|E|/|V|$

Lab 10

Implement following operations on RB-trees (**ranks have to be explicitly mantained).**

(a) Join **20**

(b) Split **35**

(c) Traverse and print the nodes in inOrder (before and after) Join and Split **20**

You are free to reuse codes of lab 8 and 9.