LAB ASSIGNEMENT FOR ADA

M. Tech. (I.T.) – 1st Year

Deadline: 06-September, 2018, 04:00 PM

Important Instructions: ***You are required to submit hand-written copies of your algorithms and their time complexity analysis to the TAs in the LAB. Although the assignments are group wise, each student needs to submit these handwritten copies individually. Deadline for submission of these hand-written copies are same as the deadline of the assignments, and those will be collected by our TAs during your program evaluations in the LAB.***

You need to explain your algorithms and codes and time complexity analysis by yourself. You have to generate suitable input and output values (appropriate test cases) to demonstrate the correct functioning of your program. Not only the final output, you are supposed to display (print) the snapshots of your data-structures after every step of execution of your program (for example, after every iteration, in an iterative algorithm).

NOTE: LAB Assignments are group wise but their evaluations would be individual. TAs are requested to ask sufficient questions to each individual member of a group and assign marks according to their individual performances. However, all the members of a group may show/submit a single copy of their programs.

ASSIGNMENT OF PROBLEMS TO GROUPS

In this assignment, you have to solve **two problems**. One is group-specific and the other is common for all groups.

Assignment of group-specific problem is as follows:-

GROUP-X = Problem No. X.

(Example: Group-1 = Problem No. 1, Group-2 = Problem No. 2., … , Group-20 = Problem No. 20)

***** PROBLEM NUMBER 21 IS COMMON FOR ALL GROUPS *****

Implement the following using C/C++

1. Implement the routines *empty*, *push*, *pop*, and *popandtest* using the dynamic storage implementations of a linked stack. Analyze time complexity for all the algorithms you have written.

2. Implement the routines *empty*, *insert*, and *remove* using a dynamic storage implementation of a linked queue. Analyze time complexity for all the algorithms you have written.

3. Implement the routines *empty*, *pqinsert*, and *pqmindelelet* using a dynamic storage implementations of a linked priority queue. Analyze time complexity for all the algorithms you have written.

4. Write a routine to interchange the *m*'th and *n*'th element of a dynamic allocation based linear linked list. Analyze time complexity for all the algorithms you have written.

5. Write a function *search(l,x)* that accepts a pointer *l* to a linked list of integers and an integer *x* and returns a pointer to a node containing *x* , if it exists, and the null pointer otherwise. Write another function *srchinsert(l,x)*, that adds *x* to *l* if it is not found and always returns a pointer to a node containing *x*. Analyze time complexity for all the algorithms you have written.

6. Write an efficient algorithm and routine to concatenate two circular lists. Is it more efficient on circular lists than on linear lists? Explain. Analyze time complexity for all the algorithms you have written.

7. Write an efficient algorithm for reversing a circular list, so that the last element becomes the first, and so on. Is it more efficient on circular lists than on linear lists? Explain. Analyze time complexity for all the algorithms you have written.

8. Write an efficient algorithm to combine two ordered circular lists into a single ordered circular list. Is it more efficient on circular lists than on linear lists? Explain. Analyze time complexity for all the algorithms you have written.

9. Write an efficient algorithm to form a circular list containing the union of the elements of two given circular lists. Is it more efficient on circular lists than on linear lists? Explain. Analyze time complexity for all the algorithms you have written.

10. Write an efficient algorithm to form a circular list containing the intersection of the elements of two given circular lists. Is it more efficient on circular lists than on linear lists? Explain. Analyze time complexity for all the algorithms you have written.

11. Write a routine *place(l,x)* to insert a new item *x* in an ordered circular list *l*. Analyze time complexity for all the algorithms you have written.

12. Write a program to solve the Joshepus problem by using a circular list. Analyze time complexity for all the algorithms you have written.

13. Write a function *multint(p,q)* to multiply two long positive integers represented by singly linked circular lists. Analyze time complexity for all the algorithms you have written.

14. Write an efficient algorithm and a routine to concatenate two doubly linked circular lists. Analyze time complexity for all the algorithms you have written.

15. Write an efficient algorithm for reversing a doubly linked circular list, so that the last element becomes the first, and so on. Analyze time complexity for all the algorithms you have written.

16. Write an efficient algorithm to combine two ordered doubly linked circular lists into a single ordered doubly linked circular list. Analyze time complexity for all the algorithms you have written.

17. Write an efficient algorithm to form a doubly linked circular list containing the union of the elements of two given doubly linked circular lists. Analyze time complexity for all the algorithms you have written.

18. Write an efficient algorithm to form a doubly linked circular list containing the intersection of the elements of two given doubly linked circular lists. Analyze time complexity for all the algorithms you have written.

19. Write a routine *addsame* to add two long integers of the same sign represented by doubly linked list. Analyze time complexity for all the algorithms you have written.

20. Write a function *multint(p,q)* to multiply two long integers represented by doubly linked circular lists. Analyze time complexity for all the algorithms you have written.

Implement the following using Only C++

21. Write a class *CircList* to implement a circular list. Analyze time complexity for all the algorithms you have written.