LAB ASSIGNEMENT FOR ADA

M. Tech. (I.T.) – 1st Year

Deadline: 30-August, 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: -

GROUP-X = Problem No. X.

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

Groups are as defined by TA Fahiem in his latest mail (declaring results for Tutorial 2)

---

1. An array could be used to implement a queue without considering the array as a circular structure, if we implement the *remove* operation in the following way. Each time we *remove* an element from the queue, shift down all the elements of the array. So, *remove* is always from the 0'th position of the array holding the queue. Implement a queue of your own itemtypes (structure) by implementing it's *remove*, *insert* and *empty* primitives using this method. Also implement a *remvtest(pq, px, pund)* function on the above queue which sets **pund* to *FALSE* and **px* to the item removed from a nonempty queue **pq*, and sets **pund* to *TRUE* if the queue is empty. Analyze time complexity for all the algorithms you have written.

2. An array could be used to implement a queue without considering the array as a circular structure, if we implement the *insert* operation in the following way. Whenever the *rear* equals the last index of the array, an attempt to insert a new item first shifts down all the items of the array so that the first item of the queue is in position 0 of the array. Implement a queue of your own itemtypes (structure) by implementing it's *remove*, *insert* and *empty* primitives using this method. Also implement a *remvtest(pq, px, pund)* function on the above queue which sets **pund* to *FALSE* and **px* to the item removed from a nonempty queue **pq*, and sets **pund* to *TRUE* if the queue is empty. Analyze time complexity for all the algorithms you have written.

3. Implement a queue using circular representation of array, where we can utilize all the cells of the array to keep items of the queue, i.e., max. no. of elements that the queue can keep equals the size

of the array. Implement the *remove*, *insert* and *empty* primitives and also implement a *remvtest(pq, px, pund)* function on the above queue which sets **pund* to *FALSE* and **px* to the item removed from a nonempty queue **pq*, and sets **pund* to *TRUE* if the queue is empty. Analyze time complexity for all the algorithms you have written.

4. Write and implement efficient algorithms to implement a queue of stacks. Analyze time complexity for all the algorithms you have written.

5. Write and implement efficient algorithms to implement a stack of queues. Analyze time complexity for all the algorithms you have written.

6. Write and implement efficient algorithms to implement a queue of queues. Analyze time complexity for all the algorithms you have written.

7. Show how to implement a queue of integers in C by using an array *queue*[100], where *queue*[0] is used to indicate the front of the queue, *queue*[1] is used to indicate its rear and *queue*[2] through *queue*[99] are used to contain the queue elements. Show how to initialize such an array to represent the empty queue and write routines *remove, insert* and *empty* for such an implementation. Analyze time complexity for all the algorithms you have written.

8. Show how to implement a queue in C in which each item consists of a variable number of integers. Analyze time complexity for all the algorithms you have written.

9. A *deque* is an ordered set of items from which items may be deleted at either end and into which items may be inserted at either end. Call the two ends of *deque* left and right. How can a *deque* be represented in C array? Write four C routines,

   remvleft, remvright, insertleft, insertright,

   to remove and insert elements at the left and right end of *deque*. Make sure that the routine work properly for the empty *deque* and that they detect overflow and underflow. Analyze time complexity for all the algorithms you have written.

10. Implement an ascending priority queue using an array. Implement its *pqinsert, pqmindelet* and *empty* primitives. For insertion, insert a new item anywhere in the array. For deletion, search the entire array to find the minimum item. Analyze time complexity for all the algorithms you have written.

11. Implement an ascending priority queue using an array. Implement its *pqinsert, pqmindelete* and *empty* primitives. For insertion, find the proper position of the new item in the array and insert in there by shifting other array elements. For deletion, remove just the first available item of the array. Analyze time complexity for all the algorithms you have written.

12. Implement a descending priority queue using an array. Implement its *pqinsert, pqmaxdelete* and *empty* primitives. For insertion, insert a new item anywhere in the array. For deletion, search the entire array to find the maximum item. Analyze time complexity for all the algorithms you have written.

13. Implement a descending priority queue using an array. Implement its *pqinsert, pqmaxdelete* and *empty* primitives. For insertion, find the proper position of the new item in the array and insert in there by shifting other array elements. For deletion, remove just the first available item of the array. Analyze time complexity for all the algorithms you have written.

14. Write a set of routines for implementing several stacks and queues within a single array. Analyze time complexity for all the algorithms you have written.

15. Write an efficient algorithm to perform concatenation of two linked lists. Analyze time complexity for all the algorithms you have written.

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

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

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

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

20. Write an efficient algorithm which can move *node(p)* forward n positions in a list. Analyze time complexity for all the algorithms you have written.