UNIVERSITY OF BIRMINGHAM

School of Computer Science

First Year Undergraduate

06-30175

30175 LC Data Structure & Algorithms

Main Summer Examinations 2022

[Answer all questions]

30175 LC Data Structure & Algorithms

Note

Answer ALL questions. Each question will be marked out of 20. The paper will be marked out of 60, which will be rescaled to a mark out of 100.

Question 1

A non-circular, singly linked list Abstract Data Type has a private `Node` pointer, called `first`, which points to the first node of the list. This ADT does not maintain a header pointer to the last node of the list, but it does maintain a variable `size` to keep track of the number of nodes in the list.

Given a pointer to a node `n`, assume that values can be read or written to by using the field names `n.val` and `n.next` for the value and next fields of the node respectively.

Assume that a garbage collector is being used, so nodes do not need to be explicitly freed. Use `END` for invalid or *null* address pointer values.

(a) `delete_nth(int n)` is a **non-recursive** method of the ADT that removes node number *n* from the list and throws an `IllegalArgumentException` if no such element exists or if *n* is less than 0. The first node is numbered 0.

Fill in the missing part of the pseudocode for this function below:

```
1 void delete_nth(int n)
2 {
3   // WRITE THE CODE THAT SHOULD BE HERE
4   return
5 }
```

[10 marks]

(b) Assume that this ADT has two different different methods to delete all the elements from the singly linked list as follows:

```
1  void delete_all_from_start()
2  {
3    while (size > 0)
4      delete_nth(0)
5    return
6  }
7
8  void delete_all_from_end()
9  {
10   while (size > 0)
11     delete_nth(size - 1)
12   return
13 }
```

Give the complexities of these two methods in big O notation and explain how you derived them. Write the pseudocode for a much simpler and more efficient method to delete all elements from the list. **[10 marks]**

Question 2

(a) The following is the pseudocode for a recursive algorithm to print the values stored in a Binary Tree (in no particular order):

```
printTree(BTNode t)
{
  if ( t != END )
  {
   print(t.val)
   printTree(t.left)
   printTree(t.right)
  }
}
```

Give the pseudocode of a **non-recursive** algorithm to print the contents of a Binary Tree (in no particular order) with a complexity that is no worse than the above recursive algorithm.

Hint: you may use a Stack Abstract Data Type in your answer. **[10 marks]**

(b) Consider an AVL-Tree of positive integers that does not allow duplicate values in the tree. The insertion of the value 10 into a **minimal** such AVL-tree that is of height 3 triggers a double rotation.

Draw the diagram of this tree both before the insertion and after the insertion has concluded. **[10 marks]**

Question 3

(a) A hash table is implemented using an array of size 8. The objects A to F are inserted, in this order into the hash table. The Hash1 and Hash2 values for these objects are:

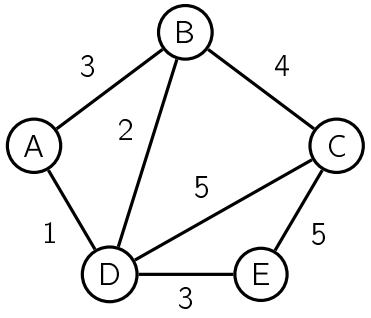
	A	B	C	D	E	F
Hash1:	11	1	1	9	0	1
Hash2:	1	3	7	1	3	5

[Note, if the list of array indexes probed for object X are 2, 5 and finally 0, then it can be written in the form: "$X \rightarrow 2 \rightarrow 5 \rightarrow 0$"]

(i) Assuming a Linear Probing strategy is employed using the Hash1 hash values, list the sequence of addresses probed for each object and draw the final state of the resulting hash table array. **[5 marks]**

(ii) Assuming a Double Hashing strategy is employed, using the Hash1 hash values as the primary hash values, and the Hash2 values as the secondary one, list the sequence of addresses probed for each object and draw the final state of the resulting hash table array. **[5 marks]**

(b) Consider the following graph:



Demonstrate the execution of the Jarník-Prim algorithm for finding the minimal spanning tree on this graph by writing out a table of the execution steps in the following format where the first row, following initialisation, is already provided (initial node to add is node A):

A	B	C	D	E	Finished
0,A	∞, B	∞, C	∞, D	∞, E	
⋮	⋮	⋮	⋮	⋮	⋮

Each row should show the results of one iteration of the algorithm after a node is added to the spanning tree, where the *Finished* column identifies the node that is finished in that iteration, and the remaining columns show the current distance of the node of that column from the tree, the node that connects it to the tree and a tick mark if the node of the column is finished. **[10 marks]**