**CS 201: Data Structures and Algorithms**

**SESSIONAL 2 SOLUTIONS**

**SPRING 2014**

**Date: 16th April, 2014 Marks: 30 Time: 90 minutes**

**Question 1:** You have an **unsorted singly linked list** with a head pointer and **NO** tail pointer and **NO** sentinel nodes. Write a function to delete the **second last node** of the list. In case the linked list has only one or less nodes, the function should indicate failure, otherwise the function should indicate success (and modify the original list). **NOTE:** we do not know the total nodes in the list in advance. If you use any helper functions then implement them also. (MARKS: 10)

**SOLUTION**

template <class type>

bool LinkedList<type>::DeleteSecondLastNode()

{

if (!head && !head->next) //checking if the list is empty or has only one node

return false;

//find the second last node

node \*curr = head;

node \*prev = NULL;

while(curr->next->next)

{

prev = curr;

curr = curr->next;

}

//check if head is deleted or if it is the general case

if (head = = curr)

head = head->next;

else

prev->next = curr->next;

delete curr;

return true;

}

**QUESTION 2:** Write a **recursive** function called trimTree, which removes (deletes) all the nodes at depth *d* or more in a BST. A Node\* type pointer to the root of the tree, and integer *d* are inputs to the function. Remember, the depth of the root is 0. Also, make sure there are no memory leaks and all the new leaves in the resultant tree have null children. **Note:** We do not know the size/total depth of the tree in advance. (MARKS: 10)

**SOLUTION**

template <class type>

void BinarySearchTree<type>::TrimTree(node \*n, int d)

{

if (!n)

return;

trimTree(n->left,d-1);

trimTree(n>right,d-1);

if (d <= 0)

delete n;

if (d = = 1)

{

n->left = NULL;

n->right = NULL;

}

}

**QUESTION 3:** Answer the following questions briefly (in NO MORE THAN 3 LINES) and to the point. There is *no partial credit* for any of these. You have to get your answers exactly right.

1. Following is an array to be converted into a binary Min-Heap, using repeated MinHeapify/InsertInHeap calls. Show the contents of this array after the Min-Heap has been constructed. (MARKS: 2)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **…** |
| - | 7 | 14 | 10 | 15 | 5 | 13 | 4 | 21 | 2 | 11 | … |

SOLUTION

2 4 5 7 11 13 10 21 15 14 (values start at index 1)

1. Consider the following heap of letters in array format:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| - | Z | W | Y | T | G | K | V | R | S | F | A | - | - |

Delete the maximum key. Give the resulting binary heap in array format. Circle those values that changed positions. (MARKS: 2)

SOLUTION

Y W V T G K A R S F (values start at index 1 and values that change are Y, V and A)

1. Given a BST with *n* ≥ *m* keys, what is the **quickest** time, in Big-O, in which you can find the *m*th smallest key. For example, if the keys stored in the tree are 10, 20, 35, 60, 70, then for *m*=3, we have the third smallest key, which is 35. How will you do it? (Don’t write code, just describe the method and state any assumptions you make). Give the time complexity in terms of *m* and *n*. (MARKS 1)

SOLUTION

I will accept any reasonable answer here. Some examples of answers given by students:

a. O(m) if you have a BST skewed to the right then the smallest key is at the root node.

b. O(log n + m) if you have a balanced BST then do an inorder traversal and stop immediately as soon as the mth key is printed. To reach the smallest key you need to traverse O(log n) nodes and from their you need an additional m traversals.

1. We wish to sort an array of numbers. This is what we do: Insert all elements one by one into a binary search tree. Repeatedly find and remove the minimum element from the tree and place it at the next location in the array. When the tree is empty, array will be sorted. What are the **best** and **worst** asymptotic running times/time complexity for this algorithm, on *n* numbers? (MARKS: 2)

SOLUTION

Worst case in case of a skewed tree: O(n2)

Best case in case of a balanced/complete BST: O(n log n)

1. If we replace the BST in (d) with a binary heap, how will that affect the running time? (MARKS: 1)

Always O(n log n) as the heap is a complete tree

1. If we were to store a binary MinHeap in a linked tree, rather than an array, how will it affect the asymptotic running times/time complexity of the functions: getMin, deleteMin and insert? Based on your answer, explain why we choose to store heaps in arrays instead? You can assume that we know the size of the heap in advance. (MARKS: 1)

**SOLUTION**

For the heap, whether it is a linked or an array structure, the following always hold:

getMin: O(1)

deleteMin: O(log n)

insert: O(log n)

Normally we use the array implementation so that we don’t need to store extra pointers and it is easier to implement using arrays.

1. Given an ordered doubly linked list with *n* keys and head and tail pointers, you want to convert it into a separate BST. What’s the quickest asymptotic time/running time required to accomplish this task and why? (MARKS: 1)

**SOLUTION**

O(n) as we can start the traversal from the head or tail and build a skewed BST.