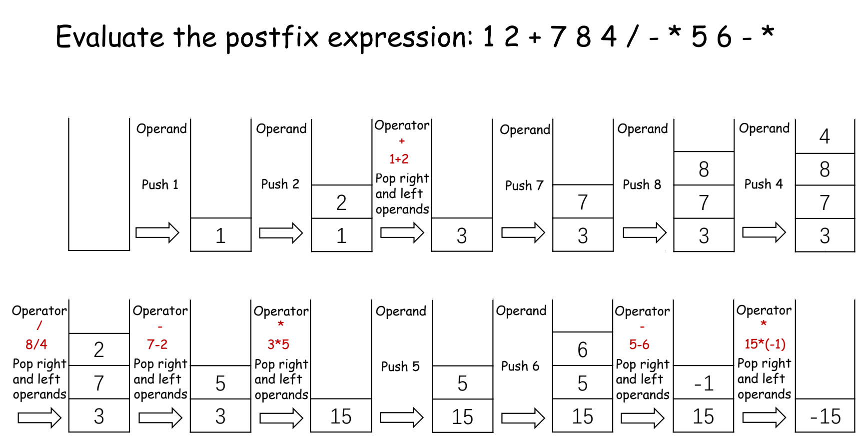
CSCI2100C 2019-20: Solution 2

This assignment is due at 11:59:59pm, 26th March 2020.

- **Q1. [30 marks]** Stacks and Queues.
  - **(i). [3 marks]** Assume that we have an empty stack S.
    - Given a series of stack operations on S as below:
    - Push$(S, 8)$, Push$(S, 5)$, Push$(S, 3)$, Pop(S), Pop(S), Push$(S, 7)$, and Pop(S).
    - Output the element returned by each Pop operation.

    The element returned by the first Pop operation is 3.
    The element returned by the first Pop operation is 5.
    The element returned by the first Pop operation is 7.
  - **(ii). [3 marks]** Assume that we have an empty queue Q.
    - Given a series of queue operations on Q as below:
    - Enqueue$(Q, 9)$, Dequeue(Q), Enqueue$(Q, 6)$, Enqueue$(Q, 3)$, Dequeue(Q), Enqueue$(Q, 4)$ and Dequeue(Q).
    - Output the element returned by each Dequeue operation.

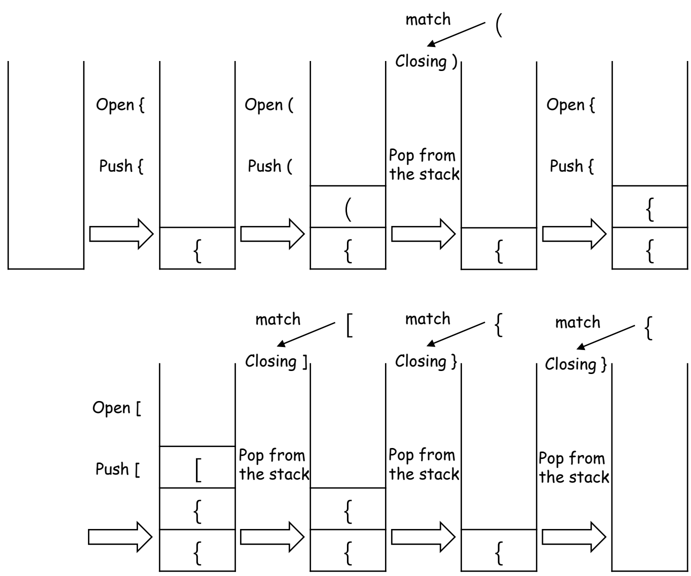
    The element returned by the first Dequeue operation is 9.
    The element returned by the first Dequeue operation is 6.
    The element returned by the first Dequeue operation is 3.
  - **(iii). [12 marks]** Given the postfix expression 1 2 + 7 8 4 / - * 5 6 - *, show how to use a stack to calculate the final results. Please show the stack status step by step (Hint. You may follow the steps as shown in CSCI2100C-Lecture8-Stack-Applications Page 20).



Solution of Q1 (iii)

Then we return -15 as the answer.

– **(iv). [12 marks]** Use a stack to check if the symbol list { () { [ ] } } is balanced. Show the stack status after each symbol checking (Hint. You may follow the steps as shown in CSCI2100C-Lecture8-Stack-Applications Page 8).



Solution of Q1 (iv)

After checking all symbols, the stack is empty: return true.

- **Q2. [28 marks]** Trees.

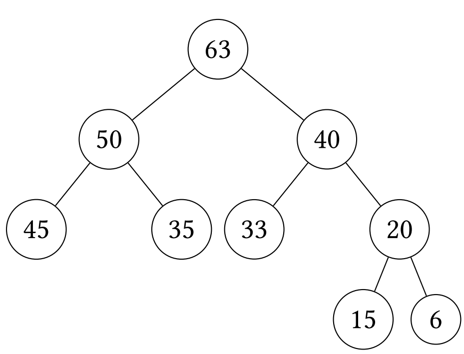


Figure 1. A Binary Tree T for Q2(i)

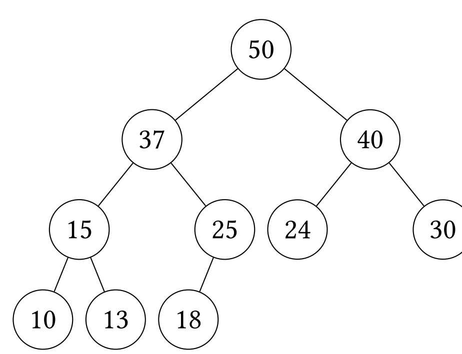


Figure 2. A Max Heap H for Q2(ii) and Q2(iii)

– **(i). [8 marks]** Given a binary tree T as shown in Figure 1, is T a max heap? Justify your answer. Next, write down the array representation of the binary tree T. Please fill the values in the array as shown below.

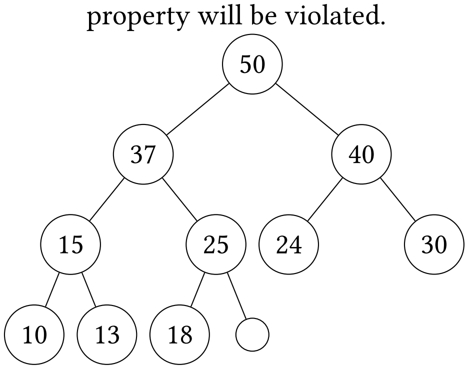
[1]	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**	**[12]**	**[13]**	**[14]**	**[15]**

It is not a completed tree. Although it is a max tree, it is not a max heap.

63	50	40	45	35	33	20							15	6
[1]	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**	**[12]**	**[13]**	**[14]**	**[15]**

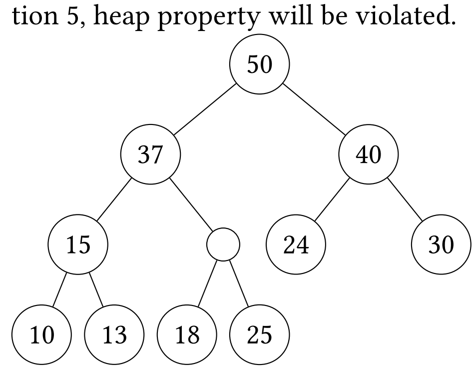
- **(ii). [10 marks]** Given the max heap H as shown in Figure 2, show the procedure of inserting 45 into the max heap step by step (Hint. You may follow the steps as shown in CSCI2100C-Lecture10-Tree Pages 15-16).

Step 1: The next position available is 11. If we put the node 45 at position 11, heap property will be violated.



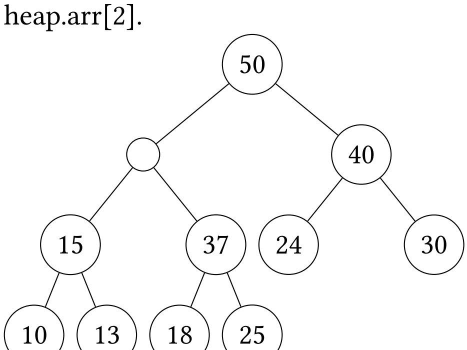
Solution of Step 1 for Q2(ii)

Step 2: Move parent here: heap.arr[11] ← heap.arr[5]. If we put the node 45 at position 5, heap property will be violated.



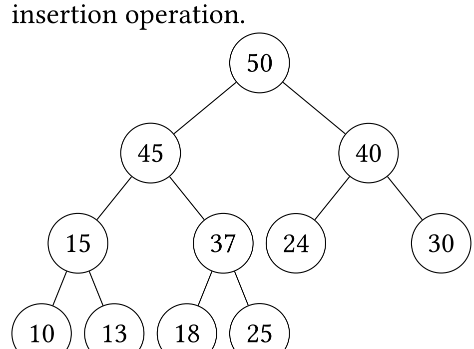
Solution of Step 2 for Q2(ii)

Step 3: Move parent here: heap.arr[5] ← heap.arr[2].



Solution of Step 3 for Q2(ii)

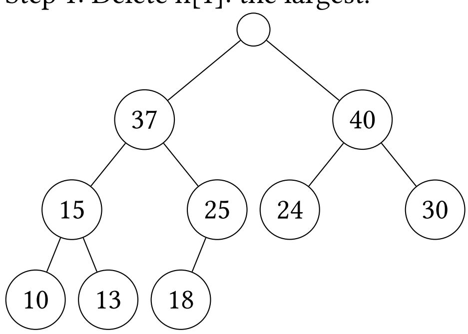
Step 4: We now attempt to put the node 45 at position 2. No violation of heap property. After inserting the node 45, we finish the insertion operation.



Solution of Step 4 for Q2(ii)

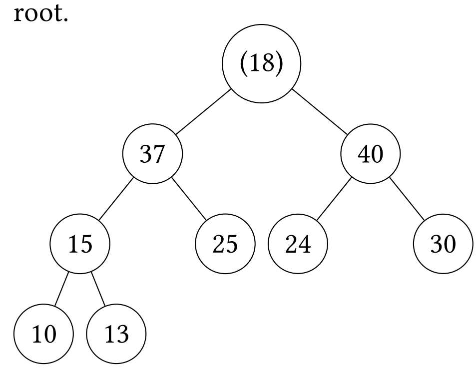
- **(iii). [10 marks]** Given a max heap H as shown in Figure 2, show the procedure of heap delete operation on the max heap step by step (Hint. You may follow the steps as shown in CSCI2100C-Lecture10-Tree Page 20).

Step 1: Delete h[1]: the largest.



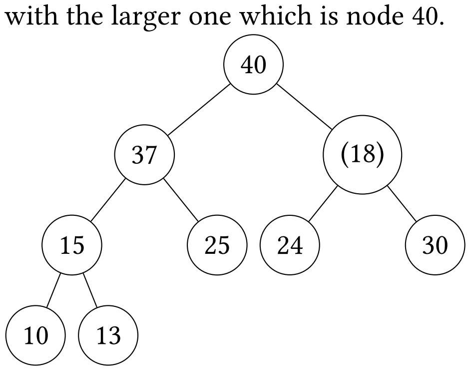
Solution of Step 1 for Q2(iii)

Step 2: Place h[n] which is node 18 at the root.



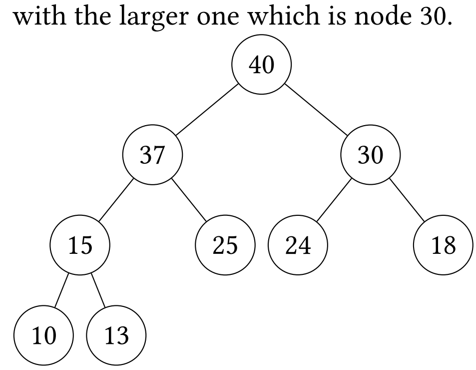
Solution of Step 2 for Q2(iii)

Step 3: Compare with two children, swap with the larger one which is node 40.



Solution of Step 3 for Q2(iii)

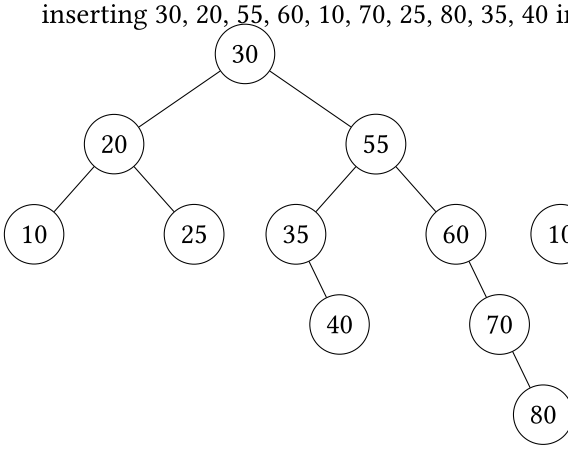
Step 4: Compare with two children, swap with the larger one which is node 30.



Solution of Step 4 for Q2(iii)

We then finish the deletion operation.

- **Q3. [24 marks]** Answer the following questions about the binary search tree.
  - **(i). [10 marks]** Given an empty binary search tree, draw the binary search tree after inserting 30, 20, 55, 60, 10, 70, 25, 80, 35, 40 in order.



Solution of Q3(i)

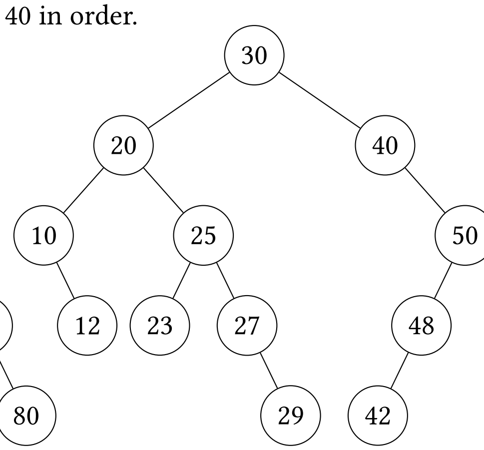


Figure 3. A Binary Search Tree for Q3

  - **(ii). [2 marks]** Given a binary search tree as shown in Figure 3, which node is the successor of node 29?

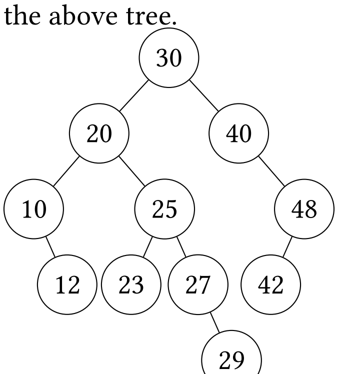
    Node 30 is the successor of node 29.

  - **(iii). [2 marks]** Given a binary search tree as shown in Figure 3, which node is the predecessor of node 42?

    Node 40 is the predecessor of node 42.

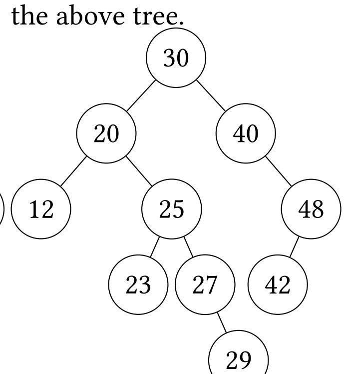
  - **(iv). [10 marks]** Given a binary search tree as shown in Figure 3, draw the binary search tree after deleting 50, 10, 20 in order.

Step 1: Delete node 50 from the above tree.



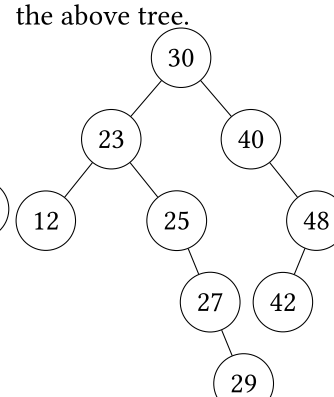
Solution of Step 1 for Q3(iv)

Step 2: Delete node 10 from the above tree.



Solution of Step 2 for Q3(iv)

Step 3: Delete node 20 from the above tree.



Solution of Step 3 for Q3(iv)

We then finish these three deletion operations.

- **Q4. [18 marks]** Given a binary search tree of n nodes and its ADT defined below, answer the following questions.

> Binary Search Tree **ADT**
> - isEmpty(root): Determine whether or not the binary tree with node **root** as the root is an empty tree.
> - leftChild(root): Return the left child of node **root**.
> - rightChild(root): Return the right child of node **root**.
> - parent(x): Return the parent of node **x**.
> - height(x): Return the height of the (sub)tree rooted at node **x**.
> - data(root): Return the data value in node **root**.
> - leftSize(root): Return the number of nodes in the left subtree of node **root**.
> - rightSize(root): Return the number of nodes in the right subtree of node **root**.

- **(i). [6 marks]** Show an algorithm in pseudo-code to find the maximum in the BST by using the above ADT operations.

```
Algorithm max(root)
  node = root
  while !isEmpty(node) and !isEmpty(rightChild(node))
     node = rightChild(node)
  return node
```

- **(ii). [6 marks]** Show an algorithm in pseudo-code to check if a binary search tree is balanced or not by using the above ADT operations (Hint: Refer to CSCI2100C-Lecture11-12-Binary-Search-Tree Page 24).

```
Algorithm isBalanced(root)
  if isEmpty(root)
     return True
  elseif abs(height(leftChild(root)) - height(rightChild(root))) ≤ 1
           and isBalanced(leftChild(root))
           and isBalanced(rightChild(root))
     return True
  else
     return False
```

- **(iii). [6 marks]** Show an algorithm in pseudo-code to find the k-th ($k \leq n$) largest element in the BST by using the above ADT operations.

```
Algorithm kthLargest(root, k)
  if k == rightSize(root) + 1
     return data(root)
  elseif k < rightSize(root) + 1
     return kthLargest(rightChild(root), k)
  else
     return kthLargest(leftChild(root), k - (rightSize(root) + 1))
```