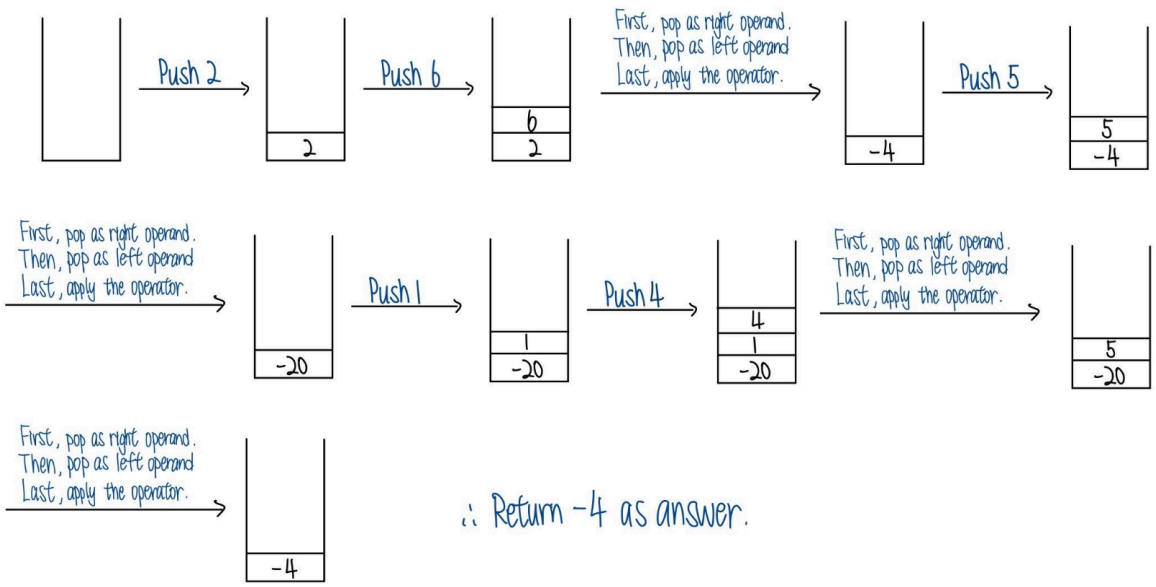
Q1. [16 marks] Answer the following questions about Stacks.

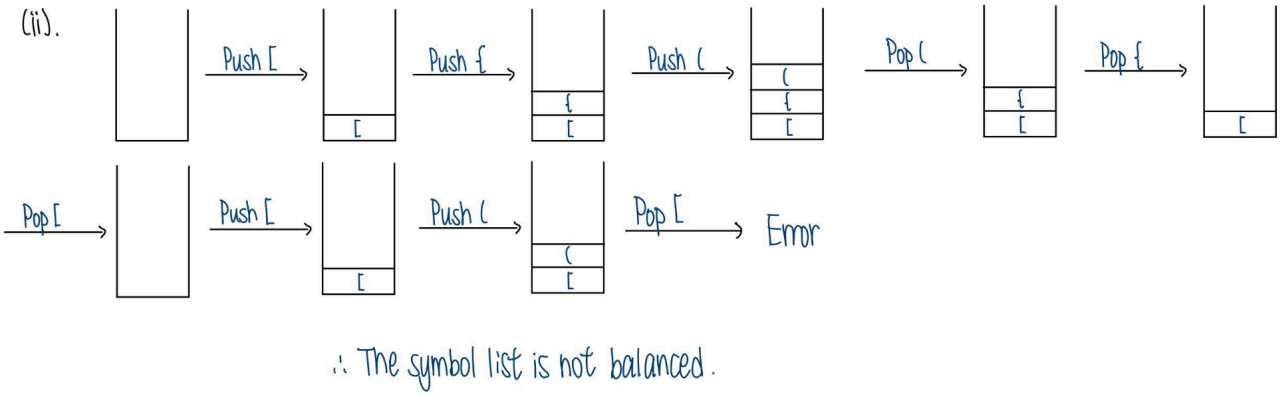
- **(i). [8 marks]** Given the postfix expression 2 6 - 5 * 1 4 + /, show how to use a stack to calculate the final results. Please show the stack status step by step (Hint. You may follow the steps as shown in `CSCI2100D-Lectures-9-Stack-Queue`).

(i).



(ii). [8 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 `CSCI2100D-Lectures-9-Stack-Queue`).

(ii).



- **Q2. [10 marks]** Answer the following questions about Queues.
  - **(i). [5 marks]** Assume that we have an empty queue Q. Given a series of queue operations on Q as below, write down the element returned by each DEQUEUE operation and show the final queue.
    * ENQUEUE$(Q, 1)$, ENQUEUE$(Q, 3)$, DEQUEUE(Q), ENQUEUE$(Q, 2)$, ENQUEUE$(Q, 5)$, DEQUEUE(Q), DEQUEUE(Q), ENQUEUE$(Q, 4)$, DEQUEUE(Q).

(i).

Operation	*Result of Q*	*Return*
Enqueue(Q,1)	[1]	
Enqueue(Q,3)	[1,3]	
Dequeue(Q)	[3]	Return 1
Enqueue(Q,2)	[3,2]	
Enqueue(Q,5)	[3,2,5]	
Dequeue(Q)	[2,5]	Return 3
Dequeue(Q)	[5]	Return 2
Enqueue(Q,4)	[5,4]	
Dequeue(Q)	[4]	Return 5

Elements returned: 1,3,2,5
Final queue: [4]

(ii). [5 marks] Suppose that we use a circular queue, what is the minimum size required to support the above series of queue operations? If the value of front is initially set to 0, what will the final values of the front and rear be?

(ii). Note that the maximum number of elements of the queue is 3 (after Enqueue(Q,5)). Therefore, a circular queue with size 4 is needed.

Operation	*Result of Q*	*Front*	*Rear*
Enqueue(Q,1)	[1]	0	1
Enqueue(Q,3)	[1,3]	0	2
Dequeue(Q)	[3]	1	2
Enqueue(Q,2)	[3,2]	1	3
Enqueue(Q,5)	[3,2,5]	1	0
Dequeue(Q)	[2,5]	2	0
Dequeue(Q)	[5]	3	0
Enqueue(Q,4)	[5,4]	3	1
Dequeue(Q)	[4]	0	1

Final values of the front = 0
Final values of the rear = 1

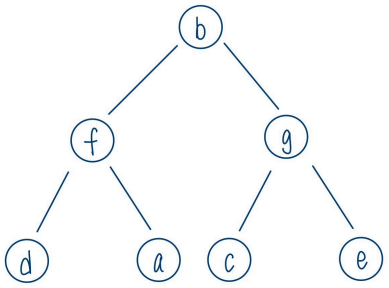
- **Q3. [30 marks]** Answer the following questions about Trees.
  - **(i) [7 marks]** Suppose we have a binary tree T_1. The inorder traversal sequence and postorder traversal sequence of T_1 are given below. Please draw the tree out and also give the preorder traversal sequence of T_1.
    * Inorder traversal sequence of T_1: d, f, a, b, c, g, e
    * Postorder traversal sequence of T_1: d, a, f, c, e, g, b

(i). Denote I and P be the inorder and postorder traversal sequence respectively.

By I, b is the root element, and it divides d,f,a and c,g,e into left (L_1) and right subtree (R_1) respectively.

At L_1, f is the root and d,a are its left and right child respectively.
Similarly, at R_1, c,e are its left and right child respectively.

Therefore, T_1 can be drawn as:



So, the preorder traversal is: b, f, d, a, g, c, e

(ii). [8 marks] Given a binary tree T_2 as shown in Fig. 1, is T_2 a max heap? Justify your answer. Next, write down the array representation of the binary tree T_2. Please fill the values in the array as shown below.

60	52	42	50	14	12	38	46	18				
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]

(ii). T_2 is a max heap as all the elements of child nodes are smaller than their parent nodes' element and it is a complete binary tree.

(iii). [15 marks] Design an algorithm in pseudo-code to find the maximum of **leaf nodes** in a binary tree. Your algorithm should use the following Binary Tree ADT operations. For example, the maximum of all leaf nodes of the binary tree T_2 in Fig. 1 is 46.

Binary Tree **ADT**

* **`Create(bintree):`** Create an empty binary tree.
* **`Boolean IsEmpty(bintree):`** If **bintree** is empty return TRUE else FALSE.
* **`MakeBT(bintree1,element,bintree2):`** Return a binary tree whose left subtree is **bintree1** and right subtree is **bintree2**, and whose root node contains the data **element**.
* **`Lchild(bintree):`** If **bintree** is empty return error else return the left subtree of **bintree**.
* **`Rchild(bintree):`** If **bintree** is empty return error else return the right subtree of **bintree**.
* **`Data(bintree):`** If **bintree** is empty return error else return the element data stored in the root node of **bintree**.

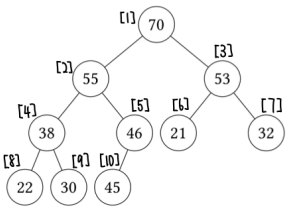
(iii).

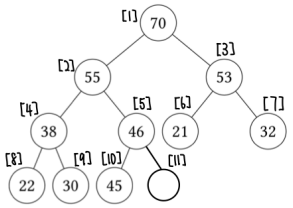
```
max(btree)
if IsEmpty(btree)
      return NULL
else if IsEmpty(Lchild(btree)) and IsEmpty(Rchild(btree))
      return Data(btree)
else
       leftMax = max(Lchild(btree))
       rightMax = max(Rchild(btree))
       if leftMax > rightMax
           return leftMax
       else
           return rightMax
```

Q4. [18 marks] Answer the following questions about the Heap.

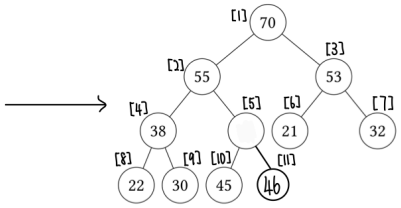
- **(i). [9 marks]** Given the max heap H as shown in Fig. 2, show the procedure of inserting 47 into the max heap step by step (Hint. You may follow the steps as shown in `CSCI2100D-Lectures-11-12-BinaryTreeADT-and-Heap`).

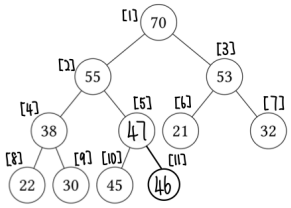
(i).





Create [11], but we cannot direct insert 47 in [11] due to the restriction of max heap.
So, we move [5] to [11]

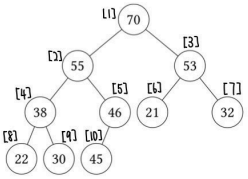


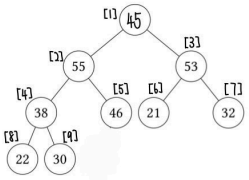


Now, we can insert 47 in [5] as it does not violates the restriction of max heap.

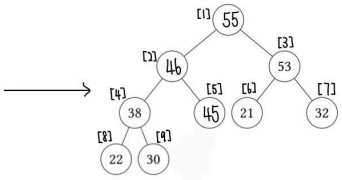
(ii). [9 marks] Given a max heap H as shown in Fig. 2, show the procedure of heap delete operation on the max heap step by step (Hint. You may follow the steps as shown in `CSCI2100D-Lectures-11-12-BinaryTreeADT-and-Heap`).

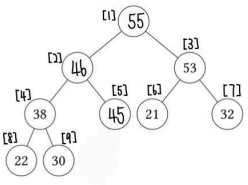
(ii).





Delete [1], place [10] in [1] and then delete [10]





Swap 45 with its larger child

Deletion is done as it does not violate the restriction of max heap.

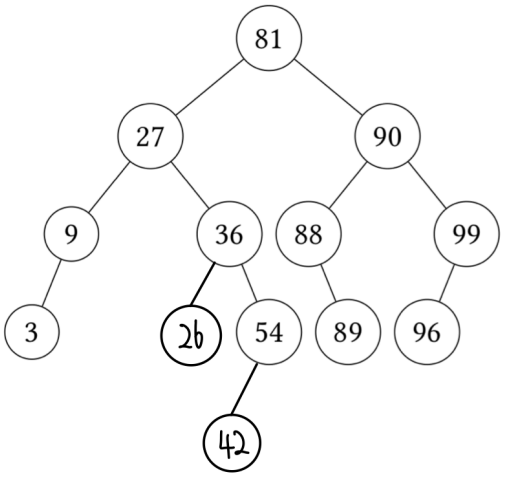
- **Q5. [18 marks]** Answer the following questions about the Binary Search Tree. (Hint. You may follow the steps as shown in `CSCI2100D-Lectures-13-Binary-Search-Tree`).
  - **(i). [4 marks]** Given a binary search tree T_3 as shown in Fig. 3, show the nodes examined when searching for 85.

(i). When searching for 85, first examine the root node (81), since 85 > 81,
then examine the right child of the root node (90), since 85 < 90,
then examine its left child (88).
Since 85 < 88, we intend to examine its left child, but it has no left child,
so return NULL.

Nodes examined: 81, 90, 88

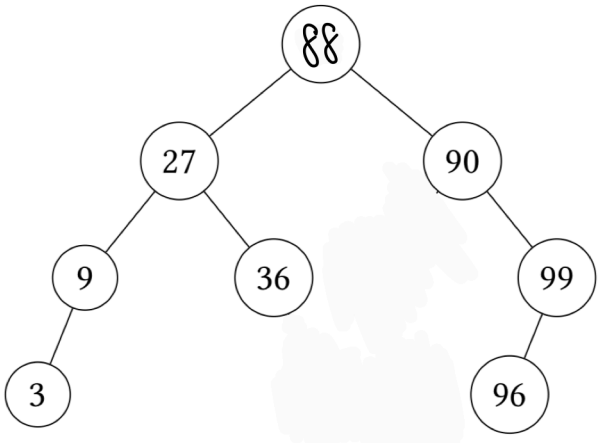
(ii). [7 marks] Given a binary search tree T_3 as shown in Fig. 3, draw the binary search tree after inserting 26, 42 in order.

(ii).



(iii). [7 marks] Given a binary search tree T_3 as shown in Fig. 3, draw the binary search tree after deleting 81, 54 in order.

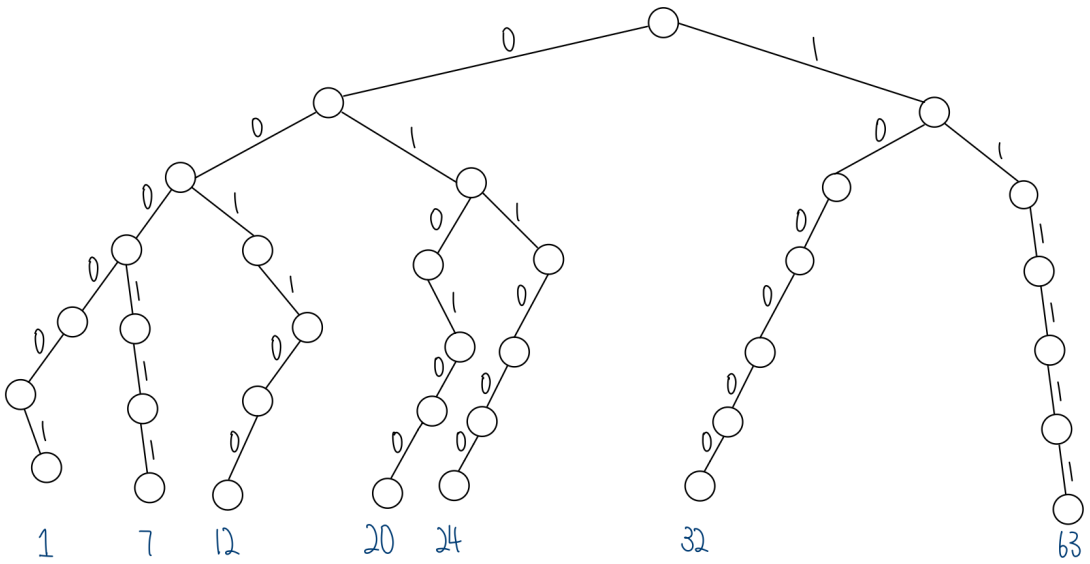
(iii).



- **Q6. [8 marks]** In computer science, a trie is a tree data structure to locate specific keys within a set. These keys are most often strings, with links between nodes defined are individual characters. In order to access a key, the trie is traversed depth-first, following the links between nodes, which represent each character in the key.
  Though tries can be keyed by character strings, they need not be. In real-world computer architectures, integers are typically represented in binary using a fixed number of bits. For example, representing 24 in binary with 6 bits would be 011000. Therefore, we can also use a binary trie to store a set of integers. The root node represents an empty binary string. Starting from the most significant bit, each node's left child represents the next bit being 0, and its right child represents the next bit being 1 in the binary representation. Fig. 4 shows a trie T_4 which maintains a set containing 0, 3, 5, 6 represented in 3-bit binary.

  (i). [4 marks] Draw a trie to maintain the set containing 1, 7, 12, 20, 24, 32, 63 represented in 6-bit binary.

(i). First, convert the integers into 6-bit binary:
1 = 000001, 7 = 000111, 12 = 001100, 20 = 010100,
24 = 011000, 32 = 100000, 63 = 111111



(ii). [4 marks] Suppose you have a trie maintains a set containing several integers represented in D-bit binary. Can you design an algorithm to find the largest element in this set that is not greater than the input k in O(D) time? Please introduce your design idea briefly.

(ii).

1. Start at the root node of the trie.
2. Convert the input *k* into a binary number and store in *result*.
3. Loop for all bits that are not greater than *k*.
4. For each element:
   If the current node has a child node equals to the current bit of *k*, append that bit to *result* and move to that child node.
   Otherwise, append 0 to *result* and move its child node.
5. After the loop, *result* is the required number.
6. Convert *result* from a binary number to an integer and return it.
