QUIZ-II (Data Structures Lab – CSC204)

QUIZ-II (Data Structures Lab – CSC204)

i. This Quiz Test contains 16 questions with total marks of 16.
ii. Answer all the questions.
iii. Duration of the test is 40 Mins.

How many binary trees can be formed with three labelled nodes A, B, and C, such that post order traversal of each tree is C, A, B ? 1 point

- ☐ 13
- ☐ 18
- ☐ 3
- ☐ 1
- ☐ 10
- ☐ 9
- ☐ 30
- ☐ 5

1 point

Suppose we implement a queue P using two stacks $S1$ and $S2$ based on the following pseudo-code. Assume that we perform n insert and m $(\leq n)$ delete operations in an arbitrary order on empty queue P. If we perform x push operations and y pop operations in this process, then select the correct statement(s) which is/are true for all m and n.

```
void insert(P, z) {
  push (S1, z);
}
 void delete(P){
   if(stack-empty(S2)) then
   if(stack-empty(S1)) then {
    print("P is empty");
    return;
  }
  else while (!(stack-empty(S1))){
    z=pop(S1);
    push(S2, z);
}
  z=pop(S2);
 }
```

$m + n \leq x < 2n \text{ and } 2m \leq y \leq m + n$

☐ Option 1

$2m \leq x < 2n \text{ and } 2m \leq y \leq m + n$

☐ Option 3

$2m \leq x < 2n \text{ and } 2m \leq y \leq 2n$

☐ Option 4

$m + n \leq x < 2n \text{ and } 2m \leq y \leq 2n$

☐ Option 2

Option 4 Option 2

1 point

Height of a node is defined as the number of edges from that particular node to the deepest leaf. Height of the root node is the height of the entire tree. A partially written C function ***height*** is given below. Which code segment(s) from the given options can be used to complete the function ***height***, such that height of a binary tree can be calculated by passing the head pointer of the tree as an argument of ***height*** function?

```
int height (struct node * head) {
        if (! head) return 0;
        if ((! head→left) && (! head→right))
        return 0;
        int l, r;
        ______________________
        ______________________
        ______________________
    }
```

$$l = head \rightarrow left;\quad r = head \rightarrow right;\quad return(1 + ((l < r)?\, l : r);$$

☐ Option 2

$$l = height(head \rightarrow left);\quad r = height(head \rightarrow right);\quad return(1 + ((l > r)?\, l : r);$$

☐ Option 5

$$l = head \rightarrow left;\quad r = head \rightarrow right;\quad return(1 + ((l > r)?\, l : r);$$

☐ Option 1

$$l = height(head \rightarrow left);\quad r = height(head \rightarrow right);\quad return(1 + ((l < r)?\, l : r);$$

☐ Option 4

$r = height(head \rightarrow right);$
$l = height(head \rightarrow left);$
$return(1 + ((l > r)?\, l : r);$

- [ ] Option 3

$l = height(head \rightarrow left);$
$r = height(head \rightarrow right);$
$return(1 + ((l < r)?\, r : l);$

- [ ] Option 6

1 point

Assume that S is an empty stack, and Q represents a queue containing 11 integers. $Head(Q)$ returns the front element of the queue, without removing the element from Q. Similarly, $top(S)$ returns the top element from the stack S without removing it from S. In the given algorithm, what is the maximum number of iterations that the ***while*** loop may iterate?

```
$while(!\,is\_empty(Q))\{$
        $if(is\_emplty(S) || Top(S) \leq Head(Q))\{$
                $X = Dequeue(Q);$
                $Push(S, X);\ \}$
        $else\ \{$
                $X = Pop(S);$
                $Enqeue(Q, X)$
                $\}$
$\}$
```

- [ ] 163
- [ ] 121
- [ ] 256
- [ ] 132
- [ ] 97
- [ ] 11

Find out the number of ways to insert elements 1, 2, 3, 4, 5, and 6 into a binary search tree, such that the tree will contain height 6. Assume the height of the root node is 1.

1 point

- ☐ 32
- ☐ 128
- ☐ 16
- ☐ 64
- ☐ 51
- ☐ 43
- ☐ 31

1 point

Consider the given C function, what does this function do if it is invoked by passing the root node of a binary tree as its argument?

```
int fun(struct node * head)
{
        if head == NULL
                return 0;
        if head → left == NULL && head → right == NULL
                return 0;
        else
                return (fun(head−> left) + fun(head−> right));
}
```

- ☐ None of these.
- ☐ It will count the number of leaf nodes.
- ☐ It will count the number of nodes in the left subtree of root including root node.
- ☐ It will count the number of internal nodes.
- ☐ It will count the number of nodes having one child.
- ☐ It will count total number of nodes.

1 point

Select the correct statement(s):

S1: If a binary tree contains 100 leaf nodes, then the tree will contain 102 nodes which have exactly two children.

S2: If a binary tree contains 100 leaf nodes, then the tree will contain 99 nodes which have exactly two children.

S3: If a binary tree contains 250 leaf nodes, then the tree will contain 252 nodes which have exactly two children.

S4: If a binary tree contains 250 leaf nodes, then the tree will contain 249 nodes which have exactly two children.

- ☐ S2 and S4
- ☐ S1 and S2
- ☐ S1 and S4
- ☐ S3 and S4
- ☐ S1 and S3
- ☐ S1, S2, and S3
- ☐ S2 and S3
- ☐ S1, S2, S3, and S4

Find out the maximum height of an AVL tree that consists of 7 nodes. Assume the height of the root node is 1.

1 point

- ☐ 4
- ☐ 12
- ☐ 2
- ☐ 6
- ☐ 3
- ☐ 1
- ☐ 5

1 point

Construct binary search trees **T1** to **T3** using the keys as attached with them. Compute the sum of balance factor of each node for each tree. Select the correct option(s) which show(s) the correct sum of balance factors for the constructed trees.

T1: 10, 20, 40, 35, 45, 55, and 80

T2: 40, 35, 50, 80, 10, 20, and 45

T3: 80, 50, 40, 35, 20, 10, and 45

- ☐ T1: -14, T2: 2, T3: 14
- ☐ T1: 14, T2: 2, T3: -14
- ☐ T1: -15, T2: 2, T3: 14
- ☐ T1: 10, T2: 2, T3: -10
- ☐ T1: -14, T2: 1, T3: 14

The pre-order traversal of a binary search tree is given as: 13, 8, 6, 3, 7, 9, 10, 16, 15, 19, 18, 25. Find out the post-order traversal of this tree. 1 point

- ☐ 3, 7, 6, 10, 9, 8, 15, 18, 25, 19, 16, 13
- ☐ 3, 7, 6, 10, 9, 8, 18, 15, 25, 19, 16, 13
- ☐ 7, 3, 6, 10, 9, 8, 15, 18, 25, 19, 13, 16
- ☐ 3, 6, 7, 8, 9, 10, 13, 15, 16, 18, 19, 25
- ☐ 3, 7, 6, 10, 9, 8, 15, 18, 25, 16, 19, 13
- ☐ None of the options
- ☐ 3, 7, 6, 10, 8, 9, 15, 18, 25, 16, 19, 13
- ☐ 7, 3, 6, 10, 9, 8, 15, 18, 25, 19, 16, 13

1 point

You have been provided following segment of the code.

```
struct data
{
 struct data *left;
 int key;
 struct data *right;
};

int Perform(struct data *p)
{
  int compute = 0;
  if (p != NULL)
  {
   if (p-> left != NULL)
    compute = 1 + Perform(p-> left);
   if (p-> right != NULL)
    compute = max(compute, 1 + Perform(p-> right));
  }
  return (compute);
}
```

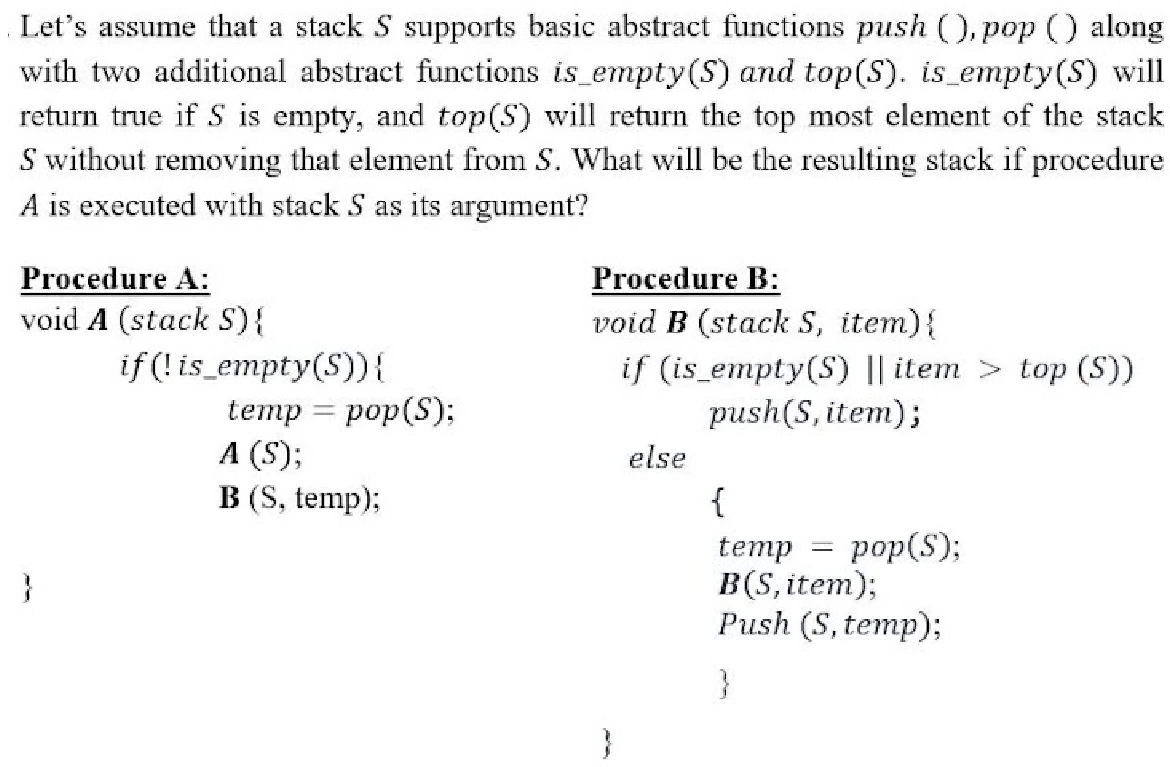
What does the value returned by the function ***Perform*** represents when a pointer to the root of a non-empty tree has been passed as an argument to the function? Choose the appropriate option(s).

- ☐ The total number of internal nodes present in the tree.
- ☐ The total number of nodes in the tree.
- ☐ The total number of nodes having no child.
- ☐ The total number of leaf nodes present in the tree.
- ☐ The number of edges of the deepest path from the root node.
- ☐ The total number of nodes having more than two children.

Suppose you build a B-tree of order 4 with 14 successive insertions starting from the scratch. Find out maximum number of splitting operations that may take place. 1 point

- ☐ 14
- ☐ 7
- ☐ 15
- ☐ 5
- ☐ 8
- ☐ 6
- ☐ 10
- ☐ 9

1 point



Let's assume that a stack S supports basic abstract functions $push\ ()$, $pop\ ()$ along with two additional abstract functions $is\_empty(S)$ and $top(S)$. $is\_empty(S)$ will return true if S is empty, and $top(S)$ will return the top most element of the stack S without removing that element from S. What will be the resulting stack if procedure A is executed with stack S as its argument?

Procedure A:

```
void A (stack S){
        if(! is_empty(S)){
                temp = pop(S);
                A (S);
                B (S, temp);

}
```

Procedure B:

```
void B (stack S, item){
  if (is_empty(S) || item > top (S))
        push(S, item);
  else
        {
        temp = pop(S);
        B(S, item);
        Push (S, temp);
        }
}
```

☐ Only median element of S will be deleted.

☐ Elements of S will be reversed.

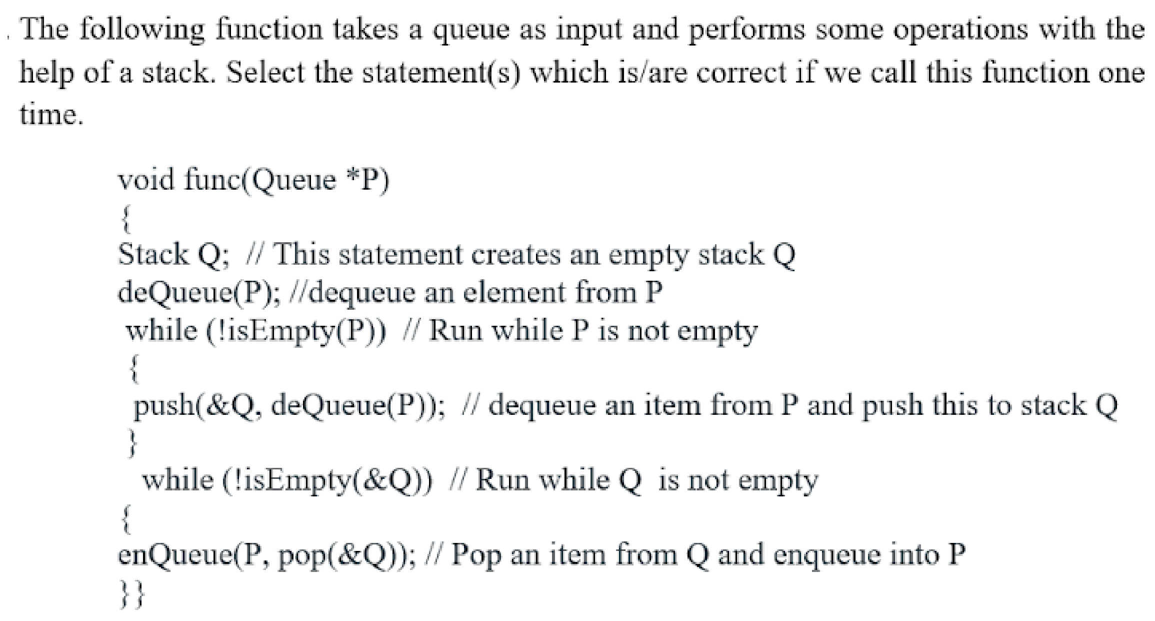
☐ S will be sorted in descending order (smallest element will be present at the top of the stack).

☐ S will be sorted in ascending order (largest element will be present at the top of the stack).

☐ S will be remain unaffected.

☐ None of these.

1 point



The following function takes a queue as input and performs some operations with the help of a stack. Select the statement(s) which is/are correct if we call this function one time.

```
void func(Queue *P)
{
Stack Q;  // This statement creates an empty stack Q
deQueue(P); //dequeue an element from P
 while (!isEmpty(P))  // Run while P is not empty
  {
  push(&Q, deQueue(P));  // dequeue an item from P and push this to stack Q
  }
   while (!isEmpty(&Q))  // Run while Q  is not empty
{
enQueue(P, pop(&Q)); // Pop an item from Q and enqueue into P
}}
```

- ☐ The queue P will not be affected.
- ☐ None of the options.
- ☐ The function removes five elements of queue P.
- ☐ The function removes one element of the queue P.
- ☐ The queue P will become empty.
- ☐ The function removes one element and reverses the remaining elements from the queue P.

1 point

We define a *3-ary* max heap using a set of array elements in which nodes can have at most *3* children. In this heap, the root node is stored at $A[0]$. The nodes in the next level are stored from left to right from $A[1]$ to $A[3]$. Furthermore, the nodes in the next level are stored from left to right from $A[4]$ location onwards. Which of the following option(s) represent(s) the valid *3-ary* max heap?

- ☐ 25, 20, 21, 22, 15, 18, 12, 2, 3, 5, 1, 4, 23, 10
- ☐ 25, 20, 21, 22, 15, 18, 12, 2, 3, 5, 1, 4, 6, 13
- ☐ 25, 20, 21, 22, 15, 18, 10, 2, 3, 5, 1, 4, 6, 13
- ☐ 25, 20, 21, 22, 15, 18, 12, 2, 3, 5, 1, 4, 6, 10
- ☐ 25, 20, 21, 22, 15, 18, 12, 2, 3, 5, 1, 4, 6, 17
- ☐ 25, 20, 21, 22, 15, 18, 12, 2, 3, 5, 7, 4, 23, 10

1 point

You have been given following key-values of the nodes to construct an AVL tree: $67, 4, 43, 15, 70, 29, 18, 47, 32$. By keeping the constraint of the AVL tree intact, find out the post-order traversal of the resulting AVL tree obtained after deleting nodes with key-values 4 and 70 from the constructed AVL tree.

- ☐ 15, 32, 18, 47, 29, 43, 67
- ☐ 67, 43, 32, 47, 29, 43, 67
- ☐ 18, 15, 32, 29, 47, 67, 43
- ☐ 32, 15, 18, 29, 67, 47, 43
- ☐ 47, 15, 43, 29, 18, 67, 32
- ☐ 29, 32, 15, 43, 47, 67, 29

Send me a copy of my responses.

Page 2 of 2

Back Submit Clear form