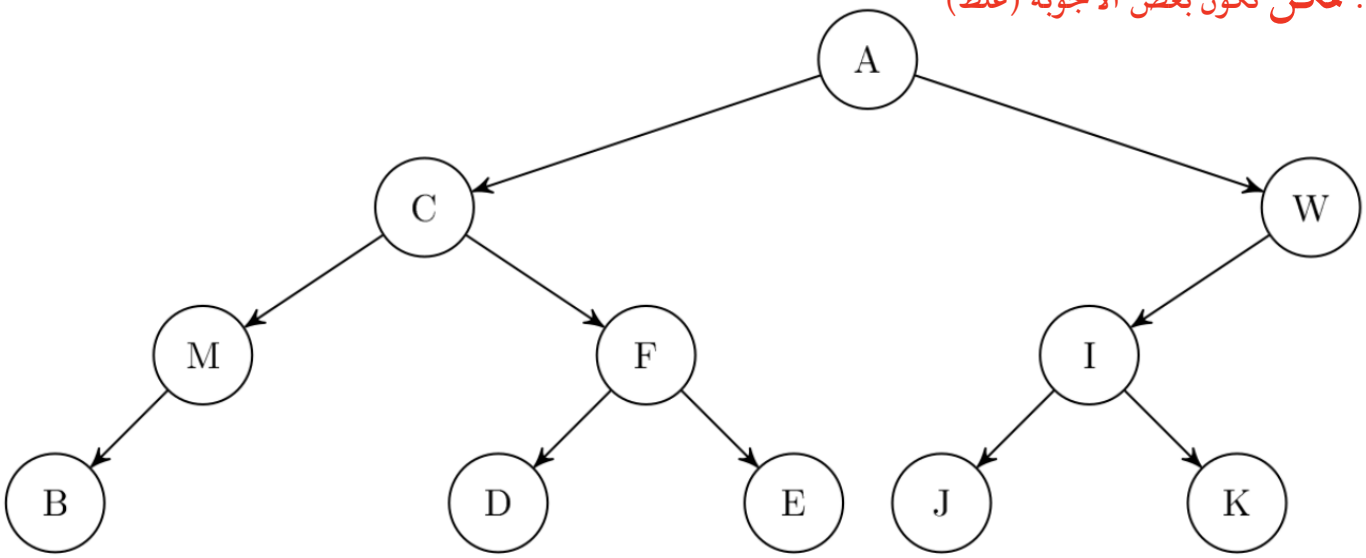
ملاحظة : **ممكن** تكون بعض الأجوبة (غلط)



Q1: preorder

The preorder traversal of this tree is

- ACMBDFEWIJK
- ACMBFDEWIKJ
- None
- ACMBDFEWIJK
- ACMDBEFWIJK

Q2: inorder

The inorder traversal of this tree is

- BMCDFEJAKIW
- BMDCFEAJIKW
- None
- BMCDFEAJIKW
- BMCDFEAJKIW

B M C D F E A J I K W

Q3: postorder

The postorder traversal of this tree is

- EFDBKGIJHCA
- None
- CWAEFBJKIMA
- CWAEFBJMIKA
- CWAEFBJKIMA

QUESTION 4

The maximum number of leaf nodes in a binary tree of height 4 is (an empty tree has height 0):

- (●) 8
- ( ) 15
- ( ) 16
- ( ) None
- ( ) 7

QUESTION 5

The method private BTNode<T> copy(BTNode<T> t) creates recursively a copy of the subtree t. Choose the correct option to complete the code of this method:

```
1 private BTNode<T> copy(BTNode<T> t) {
2   ...
3   ...
4   ...
5   ...
6   ...
7   ...
8 }
```

Line 2:

- (●) if (t == null)
- ( ) if (t.left == null || t.right == null)
- ( ) if (t.left == null && t.right == null)
- ( ) None
- ( ) if (root != null)

QUESTION 6

Line 3:

- (●) return null;
- ( ) return copy(t);
- ( ) return copy(root);
- ( ) None
- ( ) return root;

QUESTION 7

Line 4:

- ( ) BTNode<T> p = new BTNode<T>(root);
- ( ) None
- (●) BTNode<T> p = new BTNode<T>(t.data);
- ( ) BTNode<T> p = new BTNode<T>(t);
- ( ) BTNode<T> p = new BTNode<T>(root.data);

QUESTION 8

Line 5:

- ( ) t.left = copy(t.left);
- ( ) p.right = copy(t);
- ( ) None
- (●) p.left = copy(t.left);
- ( ) t.left = copy(t.left);

QUESTION 9

Line 6:

- ( ) t.right = copy(t.left);
- ( ) p.right = copy(t.left);
- ( ) t.right = copy(t.right);
- ( ) None
- (●) p.right = copy(t.right);

QUESTION 10

Line 7:

- ( ) copy(t.left); copy(t.right);
- ( ) return copy(t);
- (●) return p;
- ( ) None
- ( ) return t;

QUESTION 11

The method height, user of the ADT BT, returns the height of the tree. The height of an empty tree is 0. Choose the correct option to complete the code of this method:

```
1  public static <T> int height(BT<T> bt) {
2    ...
3    ...
4    ...
5    return recHeight(bt);
6  }
7  private static <T> int recHeight(BT<T> bt) {
8    int lh = 0; int rh = 0;
9    if(...) {
10     ...
11     ...
12   }
13   if(...) {
14     ...
15     ...
16   }
17   ...
18 }
```

Line 2:

- (●) if (bt.empty())
- ( ) None
- ( ) if (bt.root == null)
- ( ) if (root == null)
- ( ) if (bt.full())

QUESTION 12

Line 3:

- ( ) return recHeight(bt.root);
- ( ) return 1;
- (●) return 0;
- ( ) return recHeight(bt);
- ( ) None

QUESTION 13

Line 4:

- ( ) return recHeight(bt);
- ( ) bt.find(Relative.Parent);
- (●) bt.find(Relative.Root);
- ( ) bt.find(bt.root);
- ( ) None

QUESTION 14

Line 9:

- ( ) if(current.left != null) {
- ( ) None
- (●) if(bt.find(Relative.LeftChild)) {
- ( ) if(bt.find(Relative.Parent)) {
- ( ) if(bt.find(Relative.Root)) {

QUESTION 15

Line 10:

- ( ) None
- ( ) lh = height(bt);
- ( ) lh = recHeight(bt.left);
- ( ) lh = recHeight(current.left);
- (●) lh = recHeight(bt);

QUESTION 16

Line 11:

- ( ) current = findParent(root, curre
- (●) bt.find(Relative.Parent);
- ( ) None
- ( ) bt.find(Relative.LeftChild);
- ( ) bt.find(Relative.Root);

QUESTION 17

Line 13:

- (●) if(bt.find(Relative.RightChild)) {
- ( ) if(bt.find(Relative.Parent)) {
- ( ) None
- ( ) if(current.right != null) {
- ( ) if(bt.find(Relative.Root)) {

QUESTION 18

Line 14:

- ( ) rh = recHeight(current.right);
- (●) rh = recHeight(bt);
- ( ) rh = recHeight(bt.right);
- ( ) None
- ( ) rh = height(bt);

QUESTION 19

Line 15:

- ( ) current = findParent(root, curr
- (●) bt.find(Relative.Parent);
- ( ) bt.find(Relative.RightChild);
- ( ) None
- ( ) bt.find(Relative.Root);

QUESTION 20

Line 17:

- ( ) return Math.min(lh, rh) + 1;
- (●) return Math.max(lh, rh) + 1;
- ( ) return lh+rh+1;
- ( ) return Math.max(lh, rh);
- ( ) None

QUESTION 21

Consider the following elements {10, 4, 8, 6, 1}. The order of the elements after 3 swaps of the Insertion-sort algorithm (increasing order) is

- ○ None
- ○ {4, 8, 10, 6, 1}
- ○ {4, 6, 8, 1, 10}
- ◉ {4, 6, 8, 10, 1}
- ○ {4, 8, 6, 10, 1}

QUESTION 22

Consider the following elements {10, 4, 8, 6, 1}. The order of the elements after 3 swaps of the Selection-sort algorithm (increasing order) is

- ○ {1, 6, 4, 8, 10}
- ○ {1, 4, 8, 6, 10}
- ○ {1, 6, 8, 4, 10}
- ○ None
- ◉ {1, 4, 6, 8, 10}

QUESTION 23

Consider the following elements {10, 4, 8, 6, 1}. The order of the elements after 5 swaps of the Bubble-sort algorithm (increasing order) is

- ○ {4, 8, 6, 10, 1}
- ○ {4, 8, 6, 1, 10}
- ○ None
- ○ {4, 6, 1, 8, 10}
- ◉ {4, 6, 8, 1, 10}

QUESTION 24

Consider the following elements of key-value pairs
{(55,”A”), (130,”C”), (130,”B”), (7,”D”), (46,”E”), (51,”F”)}.
The order of elements after sorting the keys according to first and then second digit of Radix-sort (base 10) is:

- ○ {(7, "D"), (130, "C"), (130, "B"), (46, "E"), (55, "A"), (51, "F")}
- ◉ {(7, "D"), (130, "C"), (130, "B"), (46, "E"), (51, "F"), (55, "A")}
- ○ {(7, "D"), (130, "B"), (130, "C"), (46, "E"), (51, "F"), (55, "A")}
- ○ None
- ○ {(130, "C"), (130, "B"), (51, "F"), (55, "A"), (46, "E"), (7, "D")}