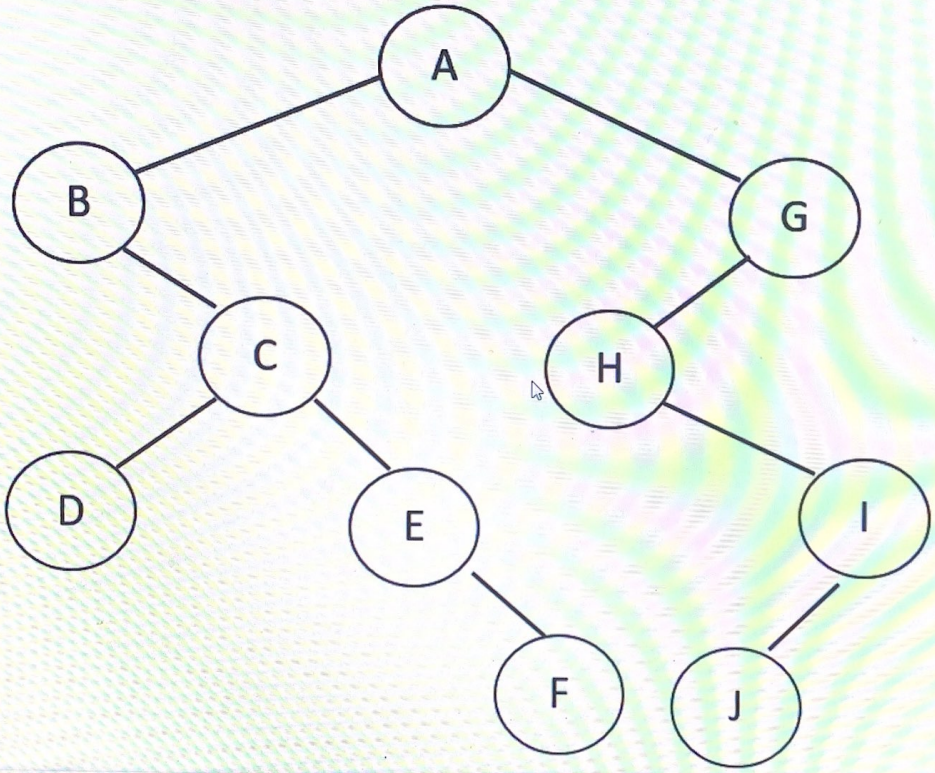
Fba

# H.W 3

Question Completion Status:

Consider the following tree:



Question Completion Status:

The output of traversing the tree in **In-order** is:

- B , C , D , E , F , A , G , H , J , I
- A , B , C , D , E , F , G , H , I , J
- B ,C , D , E , F , A , G , J , I ,H
- B , D , C , E , F , A , H , J , I , G
- None

**QUESTION 2**

The output of traversing the tree in **Pre-order** is:

- None
- A, B, D, C, E, F, G, I, H, J
- A, B, C, D, E, F, G, H, I, J
- B, C, D, E, F, A, G, H, I, J
- B, D, C, E, F, A, H, J, I, G

**QUESTION 3**

The output of traversing the tree in **Post-order** is:

- ○ F, E, D, C, B, I, J, H, G, A
- ○ B, D, F, E, C, J, I, H, G, A
- ○ None
- ○ B, D, C, E, F, A, H, J, I, G
- ○ D, F, E, C, B, J, H, I, G, A

QUESTION 4

# Question 2:

Write the method **public static <T> boolean** `inBoth(BT<T> bt, T e)`, user of the `ADT Binary Tree`, which returns true if `e` exists in both sub trees of the root of `bt`, and false otherwise.

```
public static <T> boolean inBoth(BT<T> bt, T e) {
  if (...)
    ...
  ...
  if (...)
    ...
  ...
  if (...)
    ...
  ...
}
private static <T> boolean contains(BT<T> bt, T e) {
  ...
  ...
  boolean f = ...
  if (...) {
    ...
    ...
  if (...) {
    ...
    ...
  return f;
}
```

Question Completion Status:

**Line 2:**

- if (!bt.full())
- None
- if (bt.find(Relative.Root))
- if (bt.empty())
- if (!bt.empty())

QUESTION 5

**Line 3:**

- return false;
- return true;
- bt.find(Relative.LeftChild);
- None
- bt.find(Relative.Root);

Question completion status:

**Line 4:**

- if(contains(bt, e)) return true;
- None
- bt.find(Relative.Root);
- bt.find(Relative.LeftChild);
- bt.find(Relative.Parent);

QUESTION 7

**Line 5:**

- if (bt.find(Relative.LeftChild) && contains(bt, e))
- if (!bt.find(Relative.LeftChild) || !contains(bt, e))
- if(contains(bt, e)) return true;
- None
- if (contains(bt, e)) {

**Line 6:**

- bt.find(Relative.Parent);
- None
- return true;
- return false;
- bt.find(Relative.RightChild);

QUESTION 9

**Line 7:**

- if (contains(bt, e)) return true;
- bt.find(Relative.RightChild);
- if (bt.find(Relative.RightChild)) return true;
- bt.find(Relative.Parent);
- None

**Line 8:**

- ○ if(contains(bt, e))
- ○ if (contains(bt, e)) return true;
- ○ if (bt.find(Relative.RightChild) && contains(bt, e))
- ○ if (!bt.find(Relative.RightChild) || !contains(bt, e))
- ○ None

QUESTION 11

**Line 9:**

- ○ return true;
- ○ None
- ○ if (contains(bt, e))
- ○ return false;
- ○ return contains(bt, e);

**Line 10:**

- return contains(bt, e);
- return false;
- return bt.retrieve().equals(e);
- return true;
- None

QUESTION 13

**Line 13:**

- if (bt.retrieve()== e)
- if (!bt.retrieve().equals(e))
- T tmp = null;
- if (bt.retrieve().equals(e))
- None

**Line 14:**

- return true;
- return false;
- if (bt.find(Relative.LeftChild)) {
- None
- return contains(bt, e);

QUESTION 15

**Line 15:**

- boolean f = bt.retrieve();
- boolean f = false;
- boolean f= (bt.find(Relative.LeftChild) && bt.find(Relative.RightChild);
- None
- boolean f = true;

Question Completion Status:

**Line 16:**

- if (bt.find(Relative.Parent)) {
- if (!bt.find(Relative.LeftChild)) {
- if (bt.find(Relative.LeftChild)) {
- None
- if (contains(bt, e)) {

QUESTION 17

**Line 17:**

- return contains(bt, e); }
- f = contains(bt, e);
- f = false;
- None
- bt.update(e); }

*Click Save and Submit to save and submit. Click Save All Answers to save all answers.*

**Line 18:**

- None
- bt.find(Relative.Parent); }
- bt.find(Relative.Root); }
- bt.find(Relative.LeftChild) && bt.find(Relative.RightChild));
- bt.find(Relative.RightChild));

QUESTION 19

**Line 19:**

- if (!f && bt.find(Relative.RightChild)){
- if (contains(bt, e)) {
- if (bt.retrieve()) {
- None
- if (!bt.find(Relative.RightChild)){

**Line 20:**

- contains(bt, e);
- f = false;
- f = contains(bt, e);
- None
- bt.find(Relative.Parent); }

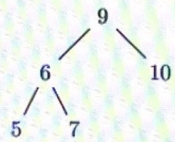
QUESTION 21

**Line 21:**

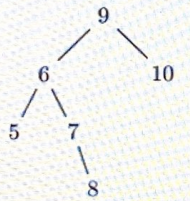
- bt.update(e); }
- contains(bt, e);
- bt.find(Relative.Root); }
- bt.find(Relative.Parent); }
- None

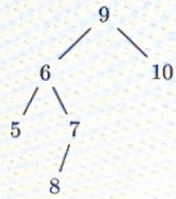
Question 3:

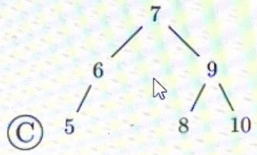
**Convention**: When necessary, the key must be replaced by the **smallest key in the right subtree**. Answers that do not follow this convention are considered wrong and receives the mark 0.

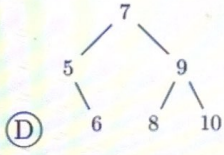
1. After inserting the key 8 in the BST 

, the tree becomes:

(A) 


(B) 

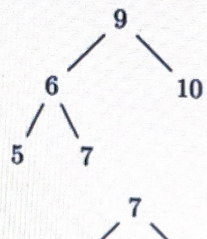

(C) 


(D) 


(E) None

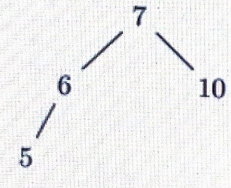
- A
- B
- C
- D
- E

2. After deleting the key 9 from the BST

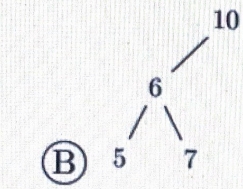



, the tree becomes:

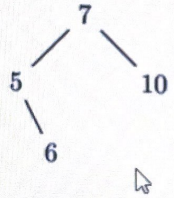
(A)



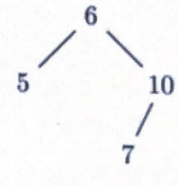

(B)




(C)




(D)




(E) None

- A
- B
- C
- D
- E