# Midterm

Correct answers marked with an asterisk (\*)

1) Given the stack: Tom, Sam (top is Tom), what is the stack after the following four operations?

1) Push Hal  
2) Push Raj  
3) Pop  
4) Pop

a. Tom, Sam, Raj, Hal

b. Hal, Tom, Sam

c. Raj, Hal

\*d. Tom, Sam

2) Given a stack myData: 34, 56, 78, 12, 66 (top is 34), what is the output after the following operations?

Push(myData 43)  
Pop(myData)  
Pop(myData)  
print(Peek(myData))  
Pop(myData)  
print(Peek(myData))

a. 43 34

b. 34 56

\*c. 56 78

d. 12 66

3) Given a stack myData: 34, 78 (top is 34), what is the output after the following operations?

Peek(myData)  
Push(myData, 2)  
Push(myData, 15)  
Pop(myData)  
Pop(myData)  
print(IsEmpty(myData))

a. true

b. 34 true

c. 78 false

\*d. false

4) Suppose a stack is implemented using a singly-linked list. Which statement about the StackPush() function is NOT true?

a. A new node is allocated

b. The new node's next pointer is assigned with the stack's current top node

c. The stack's top pointer is reassigned

\*d. The stack's top node's data is returned

5) Suppose a stack is implemented using a singly-linked list. Which statement replaces XXX to complete the StackIsEmpty() implementation, such that true is returned if the stack is empty and false is returned otherwise?

StackIsEmpty(stack) {  
 XXX  
}

a. return stack⇢top != null

\*b. return stack⇢top == null

c. return (boolean) stack⇢top⇢data

d. return stack⇢top⇢next != null

6) For an unbounded, array-based stack, which operations take O(1) time in the BEST case?

a. ArrayStackPush() only

b. ArrayStackPop() only

\*c. ArrayStackPush() and ArrayStackPop()

d. ArrayStackPush(), ArrayStackPop(), and ArrayStackResize()

7) In a queue, a dequeue operation always removes \_\_\_\_\_ element.

a. a random

b. the middle

c. the back

\*d. the front

8) Given the queue myData 12, 24, 48 (front is 12), where will the new item 72 be enqueued?

\*a. After 48

b. After 24

c. Before 12

d. After 12

9) Given the queue myData 12, 24, 48 (front is 12), what will be the queue contents after the following operations?

Enqueue(myData, 72)  
Dequeue(myData)

a. 12, 24, 72

b. 12, 48, 72

c. 24, 48, 12

\*d. 24, 48, 72

10) Given the queue myData 12, 24, 36 (front is 12), what is the result of the following operations?

Enqueue(myData, 48)  
Enqueue(myData, 60)  
Dequeue(myData)  
print(Peek(myData))  
print(IsEmpty(myData))

a. 12 false

b. 24 true

\*c. 24 false

d. 12 true

11) Given the queue myData 12, 24, 36 (front is 12), what is the result of the following operations?

Dequeue(myData)  
Dequeue(myData)  
Dequeue(myData)  
print(GetLength(myData))

a. 3

b. 2

c. 1

\*d. 0

12) An array-based queue is full if \_\_\_\_\_.

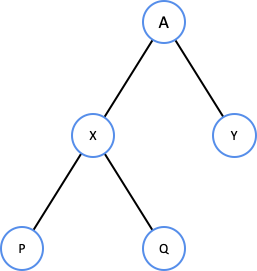
a. frontIndex == 0

b. frontIndex == allocationSize - 1

\*c. length == maxLength

d. length == allocationSize

13) Which is an internal node?



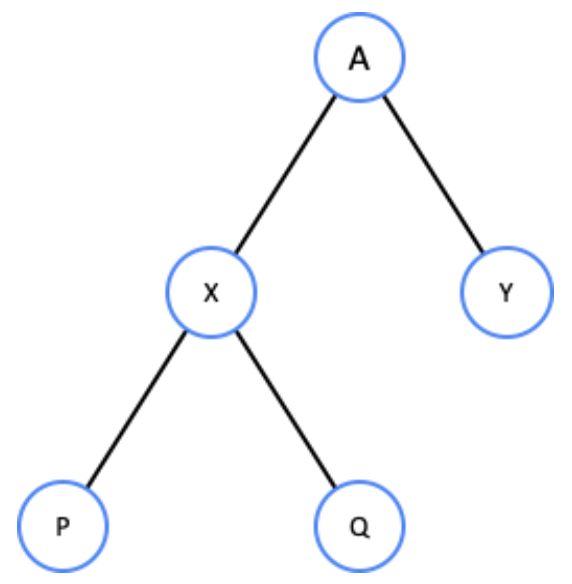
\*a. X

b. Y

c. P

d. Q

14) Which nodes are node P's ancestors?



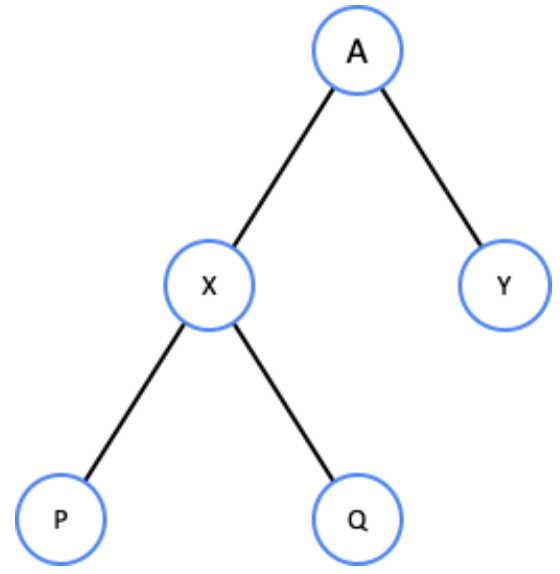
a. X

\*b. X, A

c. X, Y

d. X, Y, A

15) Node Q's depth is \_\_\_\_\_.



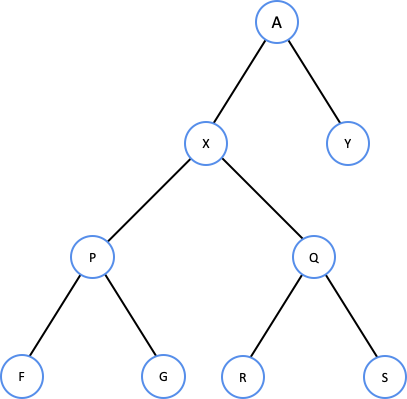
a. 0

b. 1

\*c. 2

d. 3

16) Which best describes the following binary tree?



a. Not full, complete, not perfect

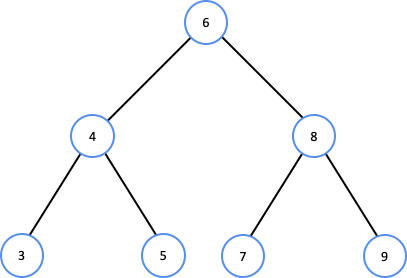
\*b. Full, not complete, not perfect

c. Full, complete, not perfect

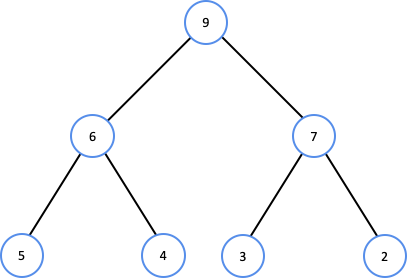
d. Full, complete, perfect

17) Which of the following is a valid binary search tree?

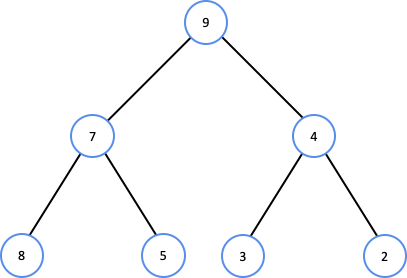
\*a.



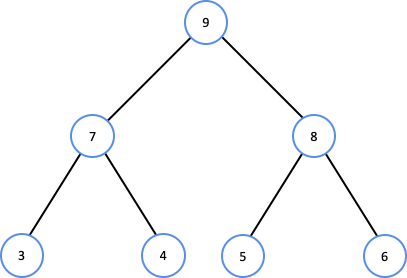
b.



c.



d.



18) Which of the following rules does a valid BST follow?

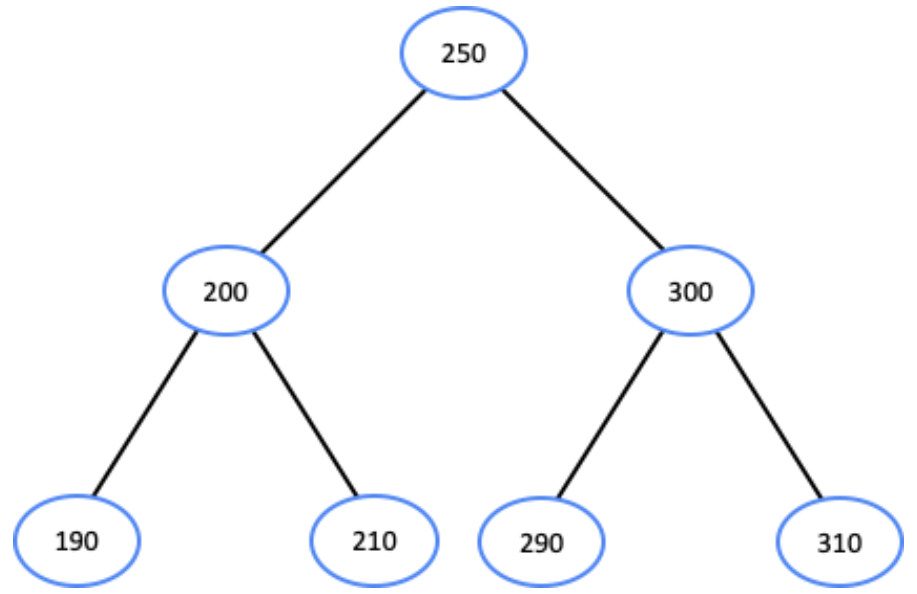
a. Right subtree keys ≤ node’s keys

b. Left subtree keys ≥ node’s keys

\*c. Left subtree keys ≤ node’s keys

d. Right subtree keys ≤ left subtree keys

19) Which nodes are visited when searching for 150?



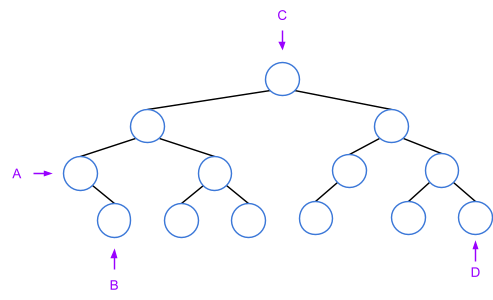
\*a. 250, 200, 190

b. 250, 200, 190, 210

c. 200, 190

d. 190, 210, 290, 310

20) The tree below has keys omitted, but is a valid BST. An**inorder**traversal of the following BST prints node \_\_\_\_\_'s key first and node \_\_\_\_\_'s key last.



\*a. A, D

b. B, D

c. C, D

d. B, C

21) Which statement replaces XXX to complete the BSTPrintInOrder() function?

BSTPrintInOrder(node) {  
 if (node == null) {  
 return  
 }  
 XXX  
 Print(node⇢key + " ")  
 BSTPrintInOrder(node⇢right)  
}

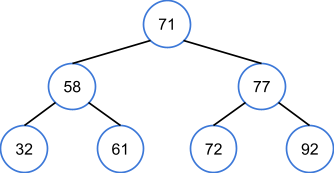
a. BSTPrintInOrder(node)

\*b. BSTPrintInOrder(node⇢left)

c. BSTPrintInOrder(node⇢right)

d. BSTPrintInOrder(null)

22) A**preorder**traversal of the following BST visits nodes in what order?



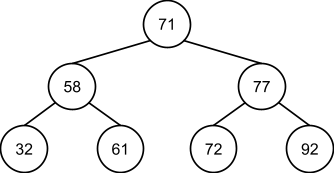
a. 32, 58, 61, 71, 72, 77, 92

b. 92, 77, 72, 71, 61, 58, 32

\*c. 71, 58, 32, 61, 77, 72, 92

d. 32, 61, 58, 72, 92, 77, 71

23) A**postorder**traversal of the following BST visits nodes in what order?



a. 32, 58, 61, 71, 72, 77, 92

b. 92, 77, 72, 71, 61, 58, 32

c. 71, 58, 32, 61, 77, 72, 92

\*d. 32, 61, 58, 72, 92, 77, 71

24) What is the height of a BST built by inserting nodes in the order 12, 24, 23, 48, 47?

a. 1

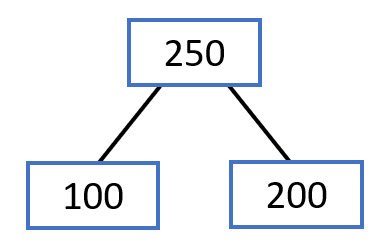
b. 2

\*c. 3

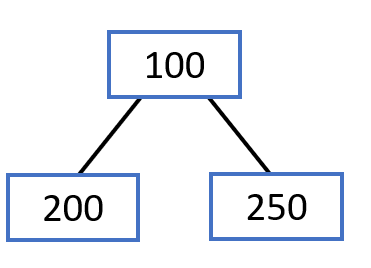
d. 4

25) Which binary tree satisfies the**min**-heap property?

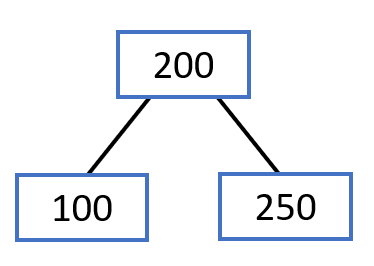
a.



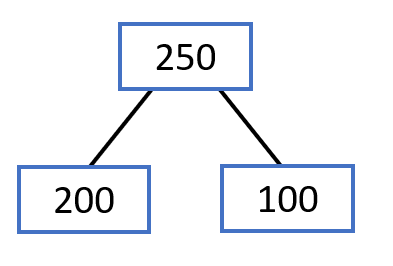
\*b.



c.

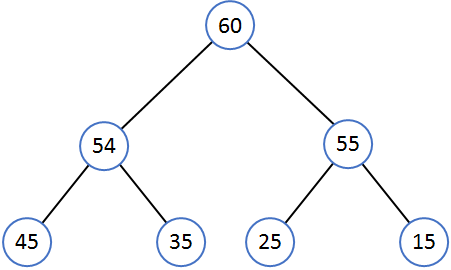


d.

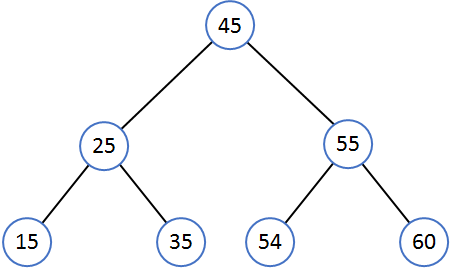


26) Which binary tree satisfies the**max**-heap property?

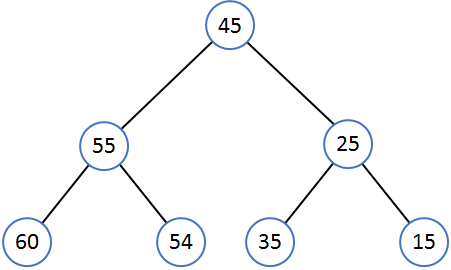
\*a.



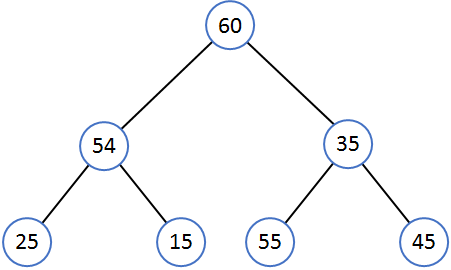
b.



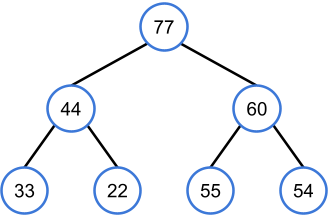
c.



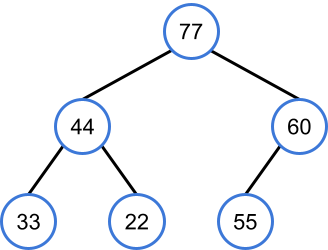
d.



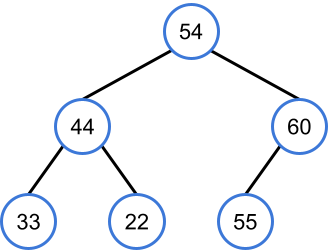
27) Removing from the following max-heap yields which max-heap?



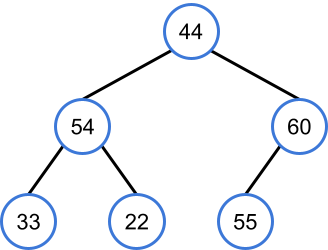
a.



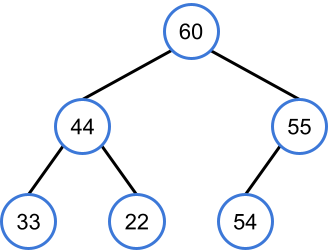
b.



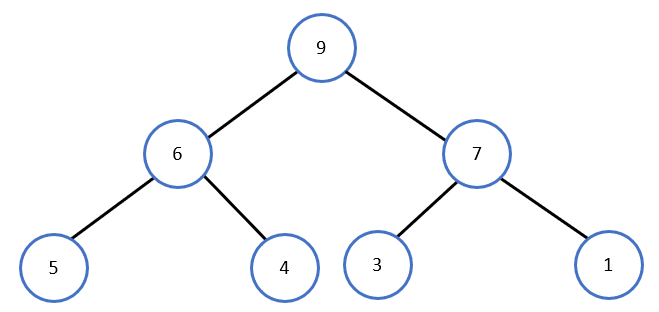
c.



\*d.



28) Which array stores the following heap?



\*a.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 6 | 7 | 5 | 4 | 3 | 1 |

b.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 6 | 5 | 4 | 7 | 3 | 1 |

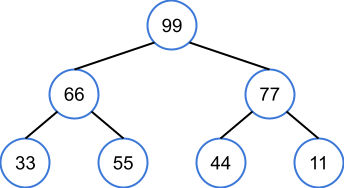
c.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 5 | 6 | 7 | 9 |

d.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 5 | 6 | 9 | 7 | 3 | 1 |

29) What are the parent and child indices for node 77 in the following max-heap?



a. Parent index: 0; child indices: 2, 3

b. Parent index: 0; child indices: 3, 4

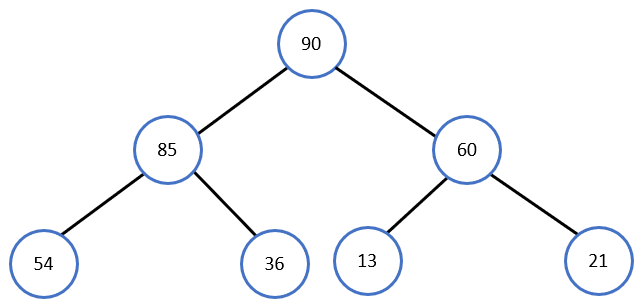
\*c. Parent index: 0; child indices: 5, 6

d. Parent index: 1; child indices: 6, 7

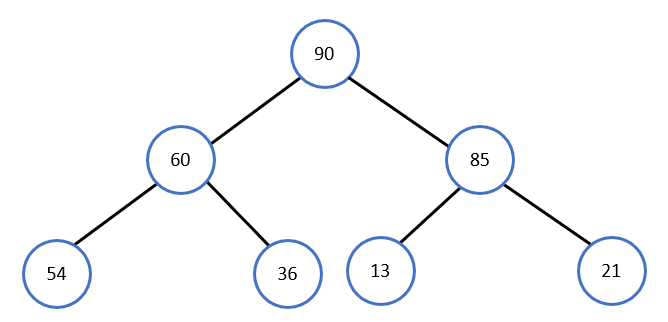
30) Heapifying the following array yields which max-heap?



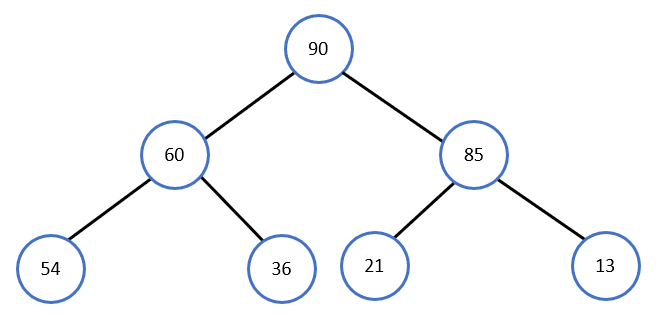
a.



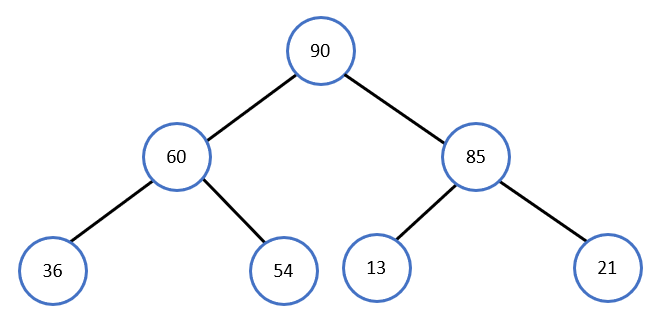
\*b.



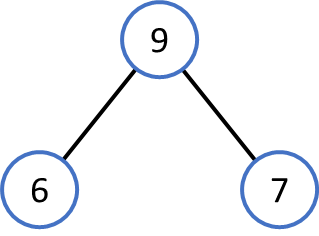
c.



d.



31) Is the following tree an AVL tree?



a. Yes, since the tree is a binary search tree (BST)

b. Yes, since both the left and right subtrees have height 0

\*c. No, since the tree is not a binary search tree (BST)

d. No, since both the left and right subtrees have height 0

32) What is the minimum possible height of an AVL tree with the following keys?

320, 470, 500, 540, 700, 650, 870

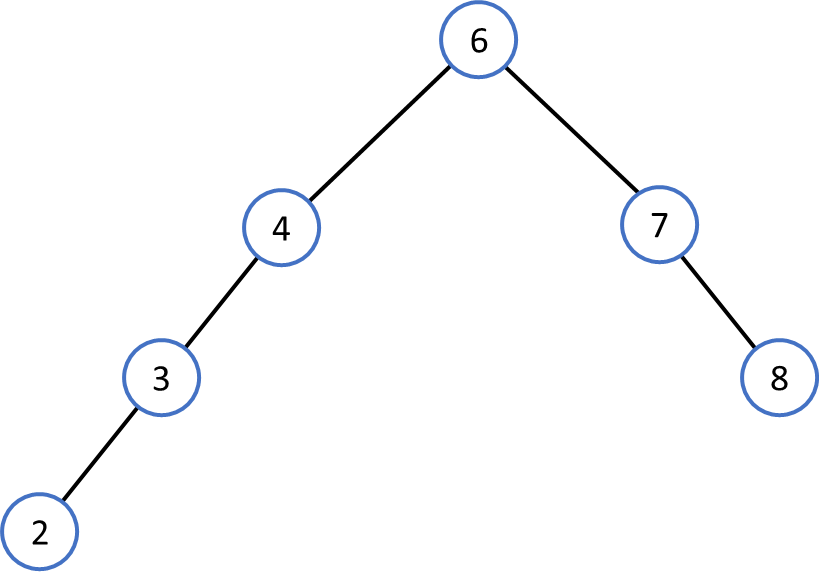
a. 1

\*b. 2

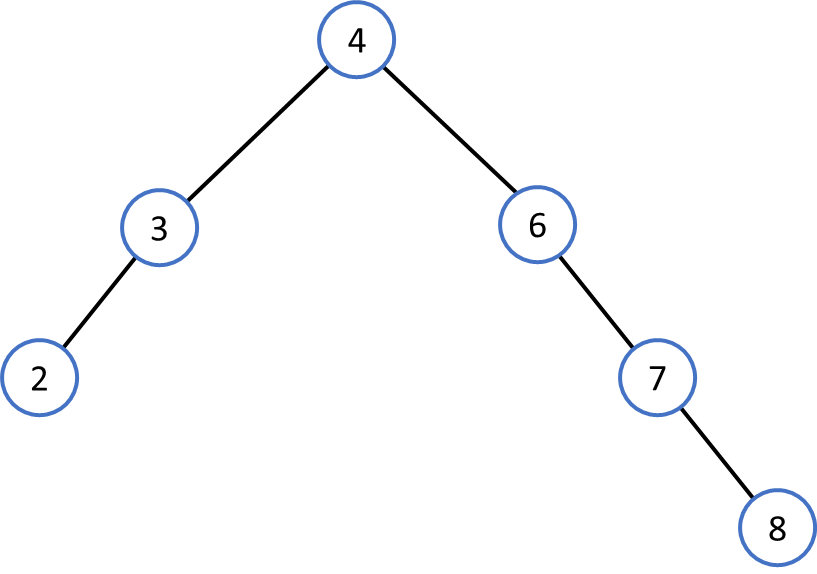
c. 3

d. 4

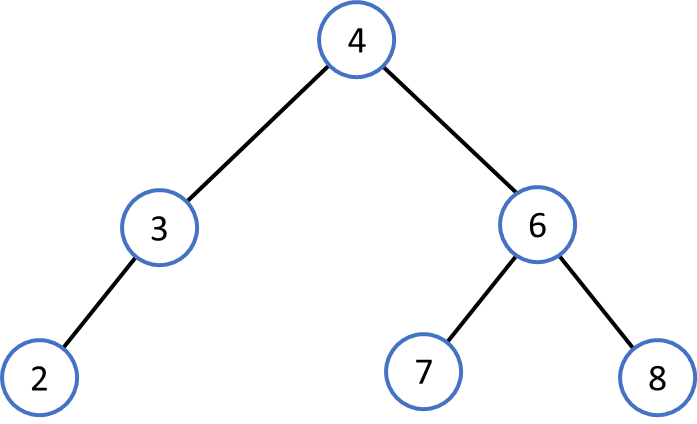
33) Identify the correct rotation to maintain the height balance of the AVL tree.



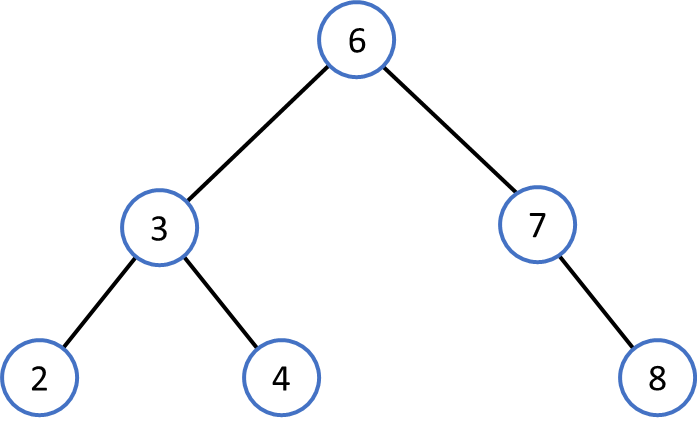
a.



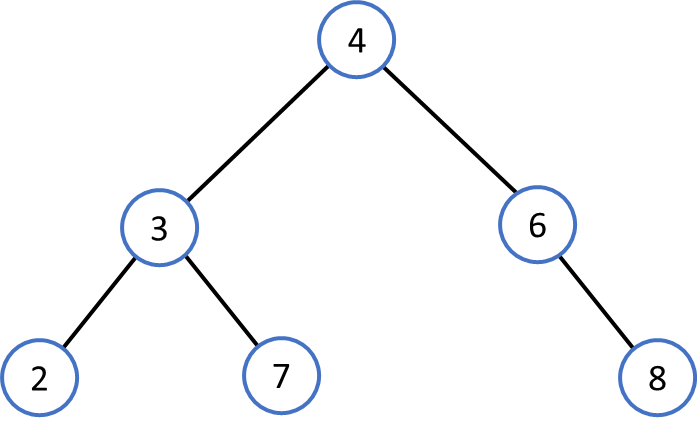
b.



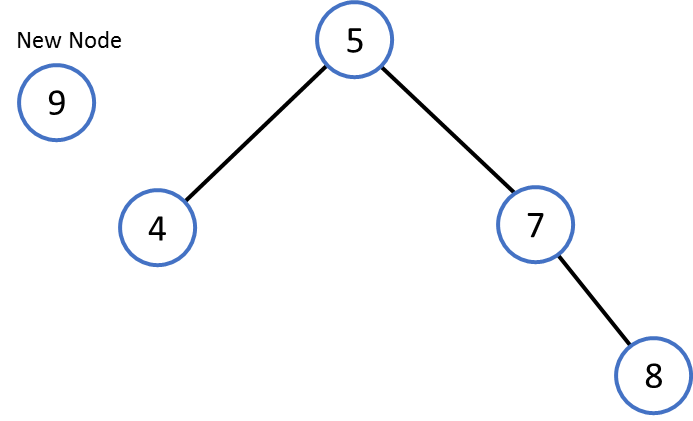
\*c.



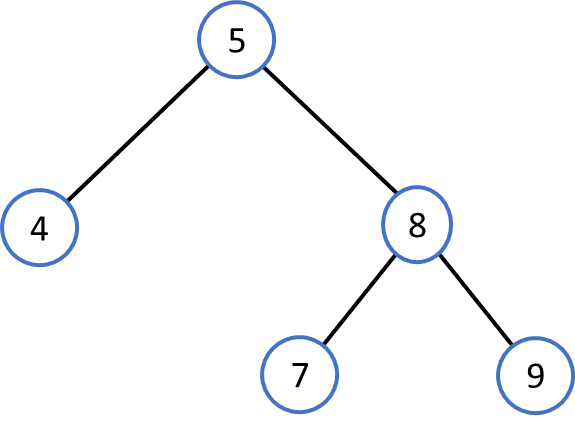
d.



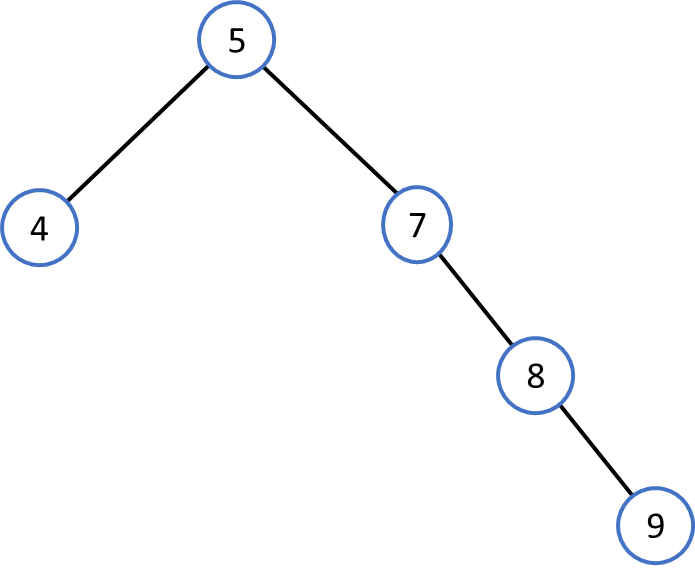
34) Which AVL tree results from inserting the new node into the AVL tree below?



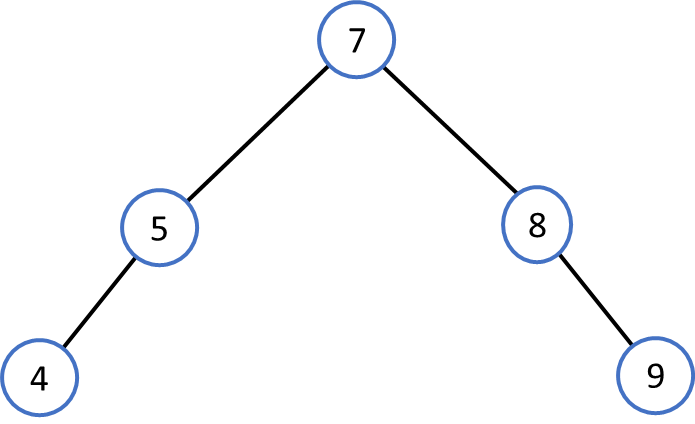
\*a.



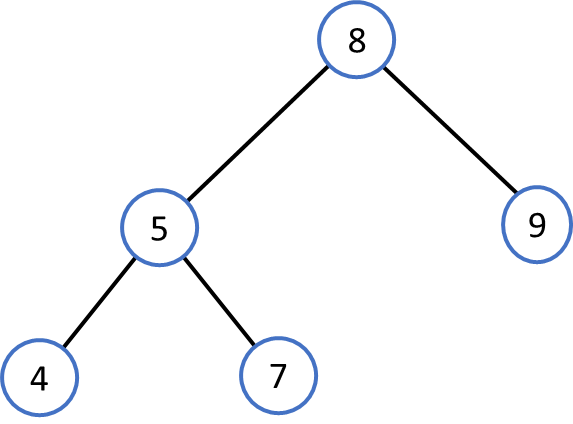
b.



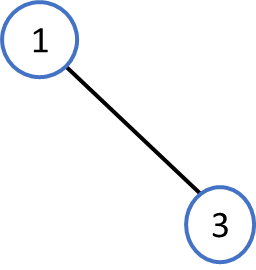
c.



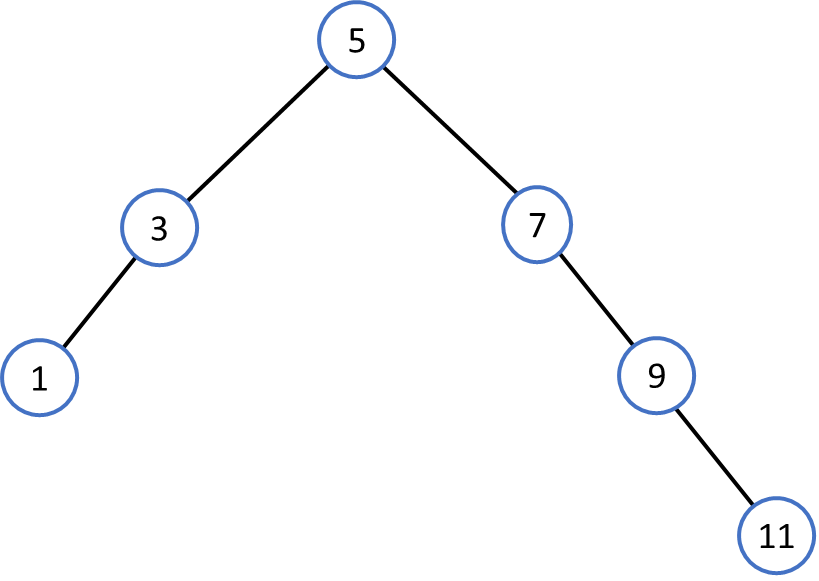
d.



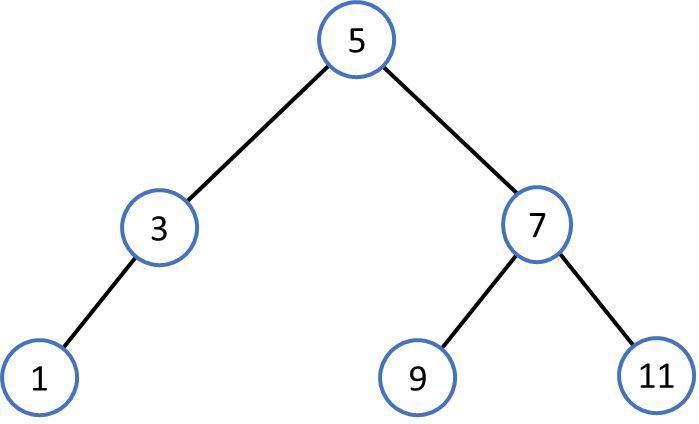
35) Which AVL tree results from inserting 5, 7, 9, and 11, in that order, into the AVL tree below?



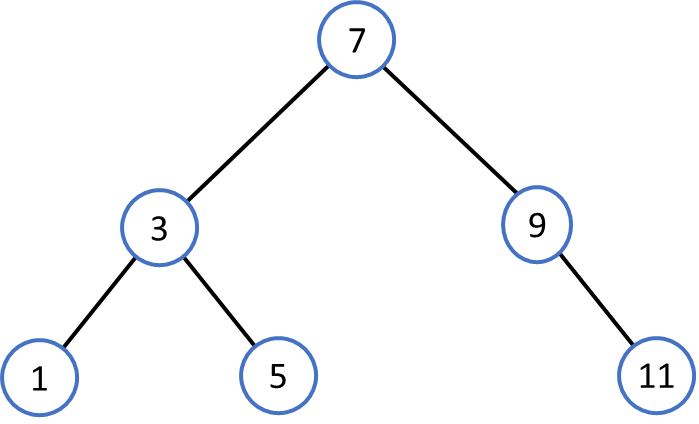
a.



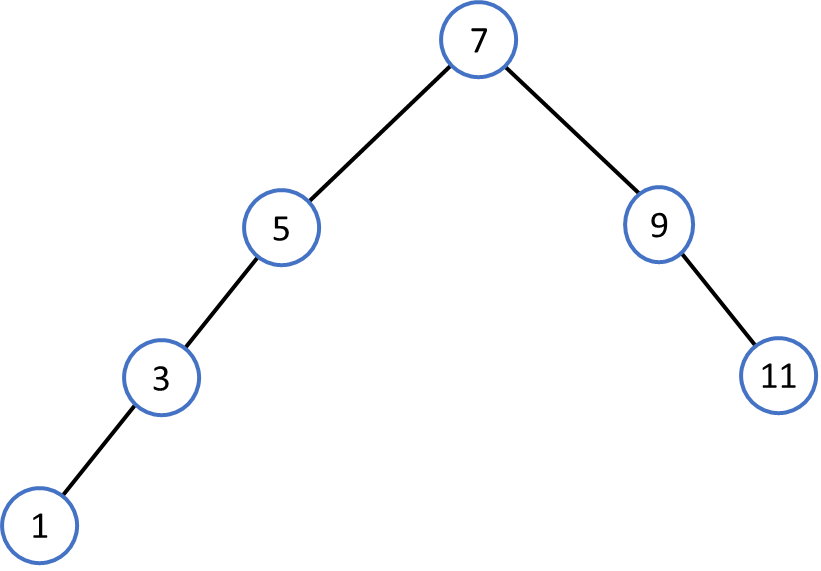
b.



\*c.



d.



36) What function is called first by AVLTreeRemoveKey()?

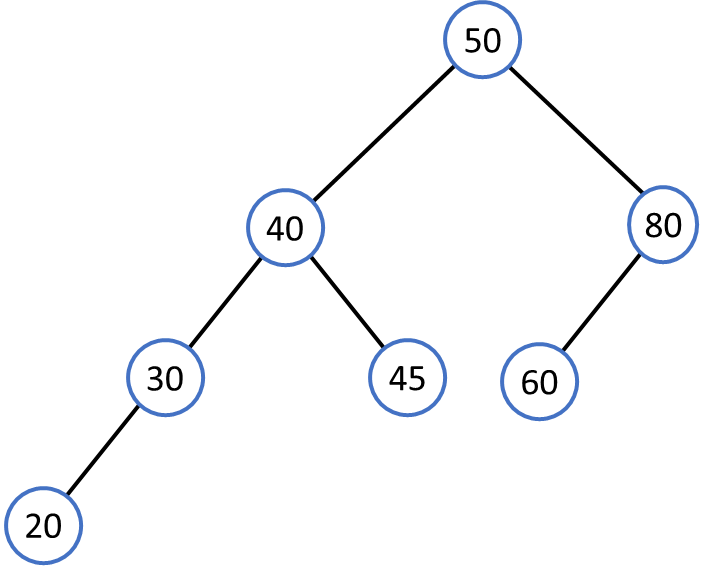
a. AVLTreeRemoveNode()

b. AVLTreeRebalance()

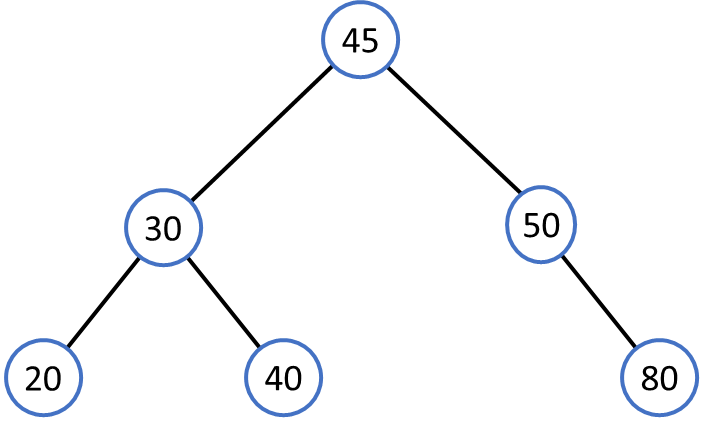
\*c. BSTSearch()

d. AVLTreeGetBalance()

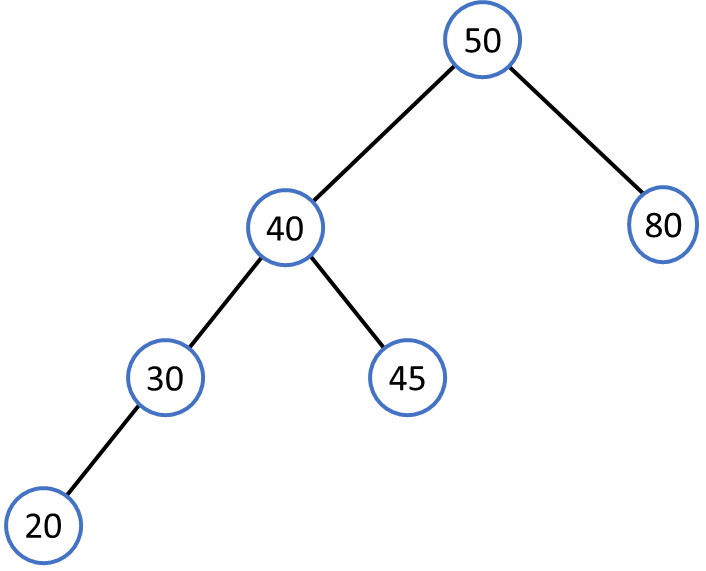
37) Which AVL tree results from removing 60 from the tree below?



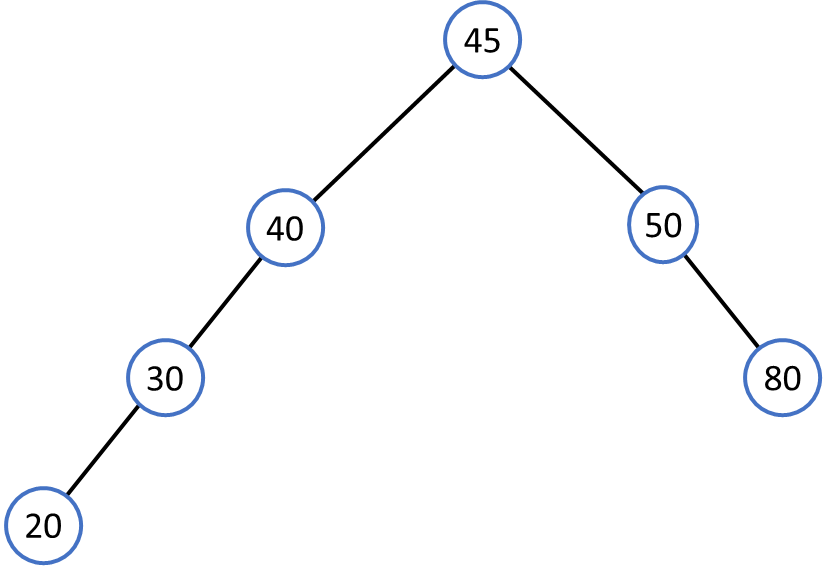
a.



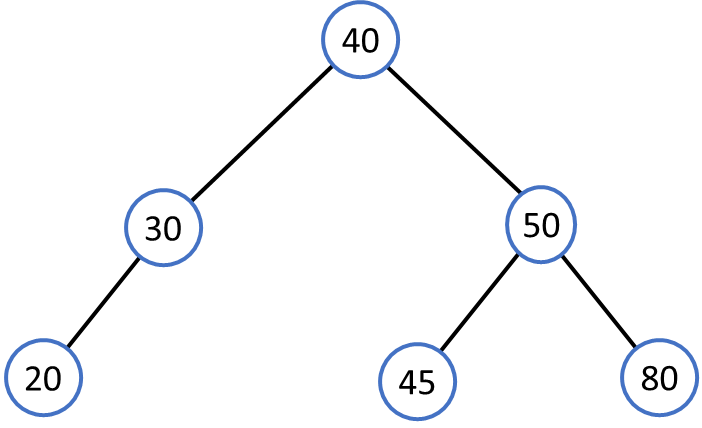
b.



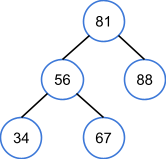
c.



\*d.



38) If key \_\_\_\_\_ is inserted into the AVL tree below, no rotations are required.



a. 11

b. 61

c. 70

\*d. 85