***Problem 1 (Binary Trees):***

template <class Elem>

int myfunction(BinNode<Elem>\* subroot)

{

if (subroot == NULL)

return 0;

return 0 + count(subroot->left())

+ count(subroot->right());

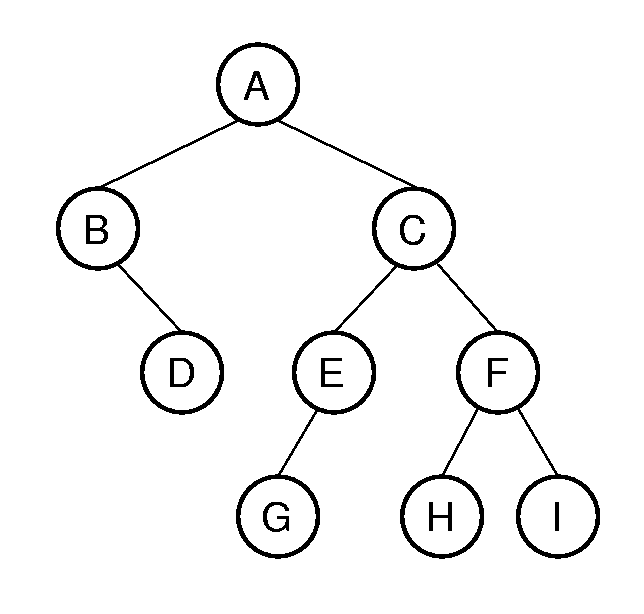
**Consider the function above.What would be its output if applied to a tree with n nodes?**

1. **The number of children on the right and left.**
2. **The number of children on the right.**
3. **The number of children on the left.**
4. **Zero**
5. **None of the above.**

***Solution 1:***

**The correct answer is Zero, because this function will always count nothing. In the original function we had a 1 instead of this zero, and that function was used to count the number of nodes in a binary tree. However, as this function passes through a node which has children, it will add a zero to the number of subroots on its left and on its right. So, it will keep on adding zeros all the time, and eventually this function will always output a Zero whatever the number of nodes in a binary tree is.**

***Problem 2 (Binary Trees):***



**Traverse the above Binary tree in pre-order and inorder.**

***Solution 2:***

**In Pre-order, we visit the root itself then we visit its left subtree and after that its right subtree. Therefore in this case we start by visiting A, then we must visit its left subtree so we visit B, after that we visit the right child of B which is D, then we visit C which is in the right subtree of A, and we continue to visit E and its child G. Finally, we still have to visit the right subtree of C so we visit F and then its left child H and then its right child I.**

**Therefore the final answer is: A,B,D,C,E,G,F,H,I.**

**In inorder, we visit left-root-right. Therefore we should start with the leftmost node which is B, after that we should visit its right child which is D, then we visit the mother of B which is A. It remains for us to traverse right subtree A. So we start with the leftmost node which is G then we visit its mother E and after that we visit C. It remains for us to traverse right subtree C, so we start with the leftmost node which is H then we visit its mother F and finally we visit its right child I. Therefore the final answer is: B,D,A,G,E,C,H,F,I.**

***Problem 3 (Binary Trees):***

**Traverse the following tree in Pre-order and Inorder.**



***Solution 3:***

**In Pre-order, we visit the root itself then we visit its left subtree and after that its right subtree. Therefore in this case we start by visiting 60, then we must visit its left subtree so we visit 20, after that we visit the left subtree of 20 so we reach 10, then we visit 40 which is in the right subtree of 20, and we continue to visit 30 and 50. Finally, we still have to visit the right subtree of 60 so we visit 70.**

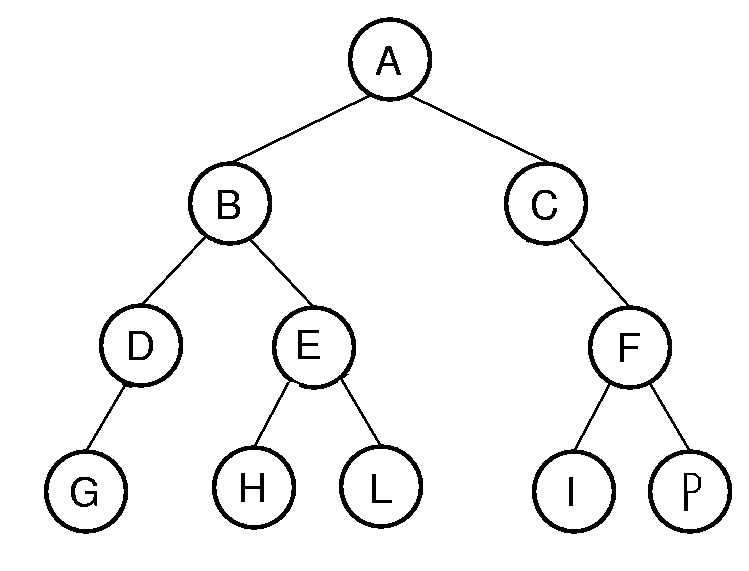
**Therefore the final answer is: 60, 20, 10, 40, 30, 50, 70.**

**In inorder, we visit left-root-right. Therefore we should start with the leftmost node which is 10, after that we should visit its mother which is 20, then we visit the leftmost node in the right subtree of 20 which is 30. Then we visit its mother 40 and the right child of 40 which is 50. It remains to visit the the main node 60 and its right child 70.**

**Therefore the final answer is: 10, 20, 30, 40, 50, 60, 70.**

***Problem 4 (Binary Trees):***

**Traverse the following tree in Pre-order and Inorder.**

****

***Solution 4:***

**In Pre-order, we visit the root itself then we visit its left subtree and after that its right subtree. Therefore in this case we start by visiting A, then we must visit its left subtree so we visit B, after that we visit the left subtree of B so we reach D, then we visit G. After that we visit the node E, then we visit its left child H and E’s right child L. After that we visit the right child of A which is C and after C we visit F. Finally, we visit the nodes I and P.**

**Therefore the final answer is: A, B, D, G, E, H, L, C, F, I, P.**

**In inorder, we visit left-root-right. Therefore we should start with the leftmost node which is G, after that we should visit its mother which is D, then we visit its mother B. Now we move to the right subtree of B and visit the leftmost node which is H. After that we visit H’s mother E and then its right child L. Now we reach the node A, then we visit the left most node in the right subtree of A which is I. After that we visit the node F and its right subroot P. It remains to visit the mother of F which is C.**

**Therefore the final answer is: G, D, B, H, E, L, A, I, F, P, C.**

***Problem 5 (Binary Trees):***

template <class Elem>

void order(BinNode<Elem>\* subroot)

{ if (subroot==NULL) return;

order(subroot->right());

cout<<subroot->val()<<endl;

order(subroot->left());

}

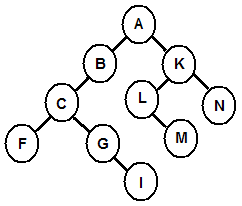
**Consider the function above.What type of tree traversal is this?**

***Solution 5:***

**If we follow the code closely, we notice that this functions visits the right subtree of root, then visits the root itself, and after that it visits the left subtree. Therefore this is neither inorder nor postorder nor pre-order, this is another different way of traversing a tree which is by visiting the right subtree then the root and after that visit the left subtree.**

***Problem 6 (Binary Trees):***

**Given the traversal in the previous question (problem 5), traverse the below tree.**



1. **FIGCBMLNKA**
2. **NMLKIGFCBA**
3. **AKNLMBCGIF**
4. **NKMLAIGCFB**
5. **None of the above**

***Solution 6:***

**In the previous question, the traverse is right-root-left. Therefore we must first visit the rightmost subtree so we start with node N. then we visit its root which is K. After that we visit M which is the rightmost root in the left subtree of K, and after that we visit its mother which is L. Now we have to traverse the main root A. After that we go to the right most root in the left subtree of A which is I, then we traverse its mother G, and then G’s mother C. After that we have to visit the left child of C which is F and it remains to visit the mother of C which is B.**

**Therefore the traversal is NKMLAIGCFB and the right answer is d).**

***Problem 7 (Binary Trees):***

**Draw the BST that results from inserting the following values in the shown order**

**40, 38, 35, 16, 37, 59, 50, 39, 60, 39, 38.**

***Solution 7:***

**The rule of inserting in a binary tree is that if the new node is larger than the existing node it is inserted on the right, if it is larger we insert it on the left. Since the first number is 40 then it will be out main node. Then we have to insert 38, since 38 is less than 40 then it is inserted to the left. Then we have to insert 35, since 35 is less than 40 then it has to be inserted to the left but on the left we have 38, so we compare it with 38 and since 35 is less than 38 then it is inserted on the left of 38. We continue in this way and finally we get the following tree:**

39

38

39

50

60

38

59

40

35

16

37

***Problem 8 (Binary Trees):***

**Draw the BST tree that results after deleting the record whose key is equal to 2 in the BST below.**

2

12

9

6

9

11

15

13

10

***Solution 8:***

**When deleting a node in a binary tree, it is swapped with the minimum node in its right subtree and after that we delete the node which contains the record we want to delete. If the node to be deleted doesn’t have a right subtree, then it is simply just removed.**

**In this case, we want to delete the record 2. The minimum node in the right subtree is 6, therefore we swap the records 2 and 6 and we delete 2 from the tree.**

**Therefore the tree becomes:**

6

12

9

9

11

15

13

10

***Problem 9 (Binary Trees):***

**Given a plain binary tree, examine the tree to determine if it meets the requirement to be a binary search tree. Which trees are BSTs?**

**a.  5     
   / \   
  2   7**

**b.  5     
   / \   
  6   7**

**c.   5    
    / \   
   2   7   
  /   
 1**

**d.   5    
    / \   
   2   7   
  / \   
 1   6**

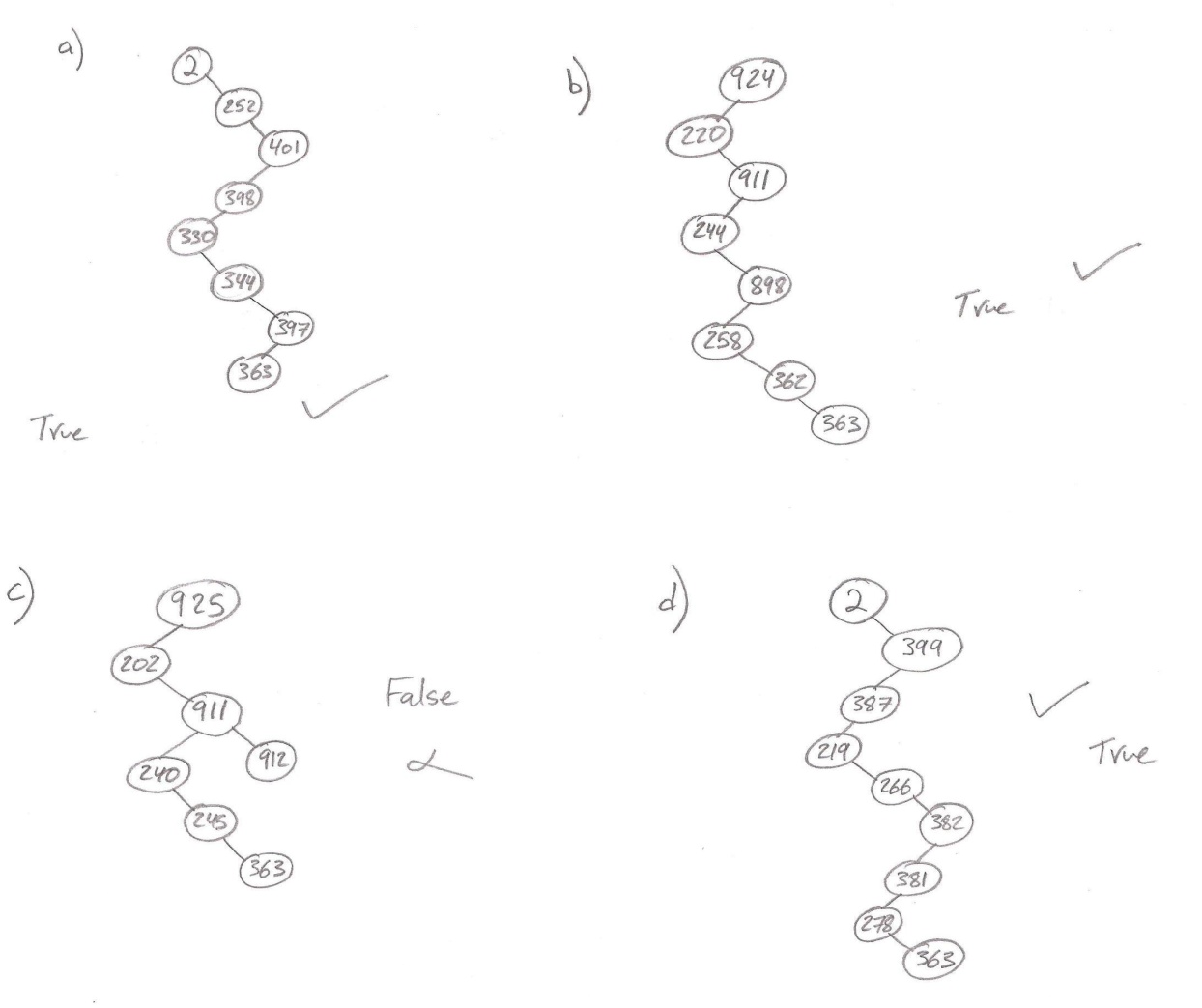
***Solution 9:***

1. **True, because 2 is less than 5 and 7 is greater than 5.**
2. **False, because the 6 is greater than 5 and it is placed on its left.**
3. **True, because 1 is less than 5, 7 is greater than 5, and in the left subtree 1 is less than 2.**
4. **False, because although 6 is less than 2, 6 is greater than 5 and shouldn’t be placed in its left subtree.**

***Problem 10 (Binary Trees):***

**Suppose that we have numbers between 1 and 1000 in a binary search tree and want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined?**

1. **2, 252, 401, 398, 330, 344, 397, 363.**
2. **924, 220, 911, 244, 898, 258, 362, 363.**
3. **925, 202, 911, 240, 912, 245, 363.**
4. **2, 399, 387, 219, 266, 382, 381, 278, 363.**
5. **935, 278, 347, 621, 299, 392, 358, 363.**

***Solution 10:***

**As we notice from the drawings c) is the wrong answer because we cannot check 911, 912 and then 240. Therefore c) is the wrong answer.**

***Problem 11 (Binary Trees):***

void main()

{Employee e;

BST <int,Employee,IDEmpCompare, IDCompare> EmpDict1;

EmpDict1.insert(Employee(44,"Ali"));

EmpDict1.insert(Employee(82,"Ahmad"));

EmpDict1.insert(Employee(33,"Michel"));

EmpDict1.insert(Employee(24,"Nader"));

EmpDict1.insert(Employee(14,"Joe"));

EmpDict1.insert(Employee(35,"Mohd"));

EmpDict1.insert(Employee(102,"Kobe"));

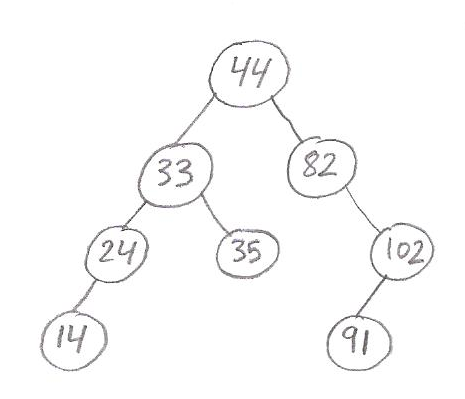
EmpDict1.insert(Employee(91,"Samer"));

EmpDict1.remove(33,e);

EmpDict1.removeAny(e);

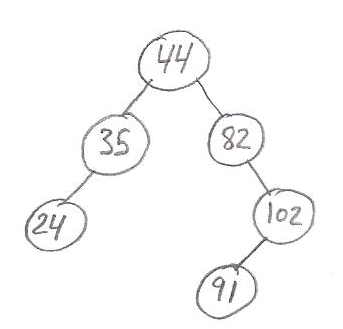
}

**What will the final tree look like?**

******

***Solution 11:***

**Since we are using the IDCompare as an element-element compare, therefore they are inserted in the tree according to ID. Therefore the tree becomes:**



**EmpDict1.remove(33,e); means that we have to remove the record with ID 33, when removing it we must replace it with the minimum from its right subtree which is 35. EmpDict1.removeAny(e); removes the the minimum record in the tree which is 14. Finally the tree becomes:**

***Problem 12 (Binary Trees):***

void main()

{Employee e;

BST <int,Employee,IDEmpCompare, NameCompare> EmpDict1;

EmpDict1.insert(Employee(44,"Ali"));

EmpDict1.insert(Employee(82,"Ahmad"));

EmpDict1.insert(Employee(33,"Michel"));

EmpDict1.insert(Employee(24,"Nader"));

EmpDict1.insert(Employee(14,"Joe"));

EmpDict1.insert(Employee(35,"Mohd"));

EmpDict1.insert(Employee(63,"Hisham"));

EmpDict1.insert(Employee(48,"Samer"));

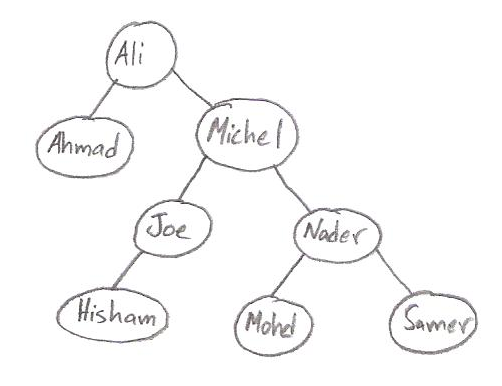
EmpDict1.remove(33,e);

EmpDict1.removeAny(e);

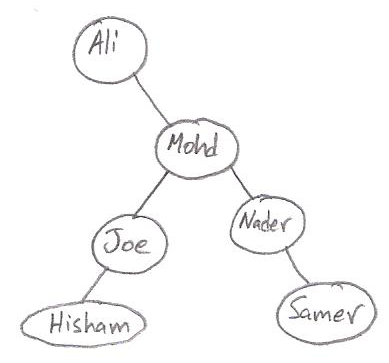
EmpDict1.print();

}

**What will the final tree look like?**

***Solution 12:***

**Since we are using the NameCompare as an element-element compare, therefore they are inserted in the tree according to Name. Therefore the tree becomes:**

**EmpDict1.remove(33,e); means that we have to remove the record with ID 33, which is “Michel” and when removing it we must replace it with the minimum from its right subtree which is “Mohd”. EmpDict1.removeAny(e); removes the the minimum record in the tree which is “Ahmad”. Finally the tree becomes:**

***Problem 13 (Binary Trees):***

**Draw the BST tree that results after deleting the record whose key is equal to 6 and 12 in the BST below.**

2

12

9

6

9

11

15

13

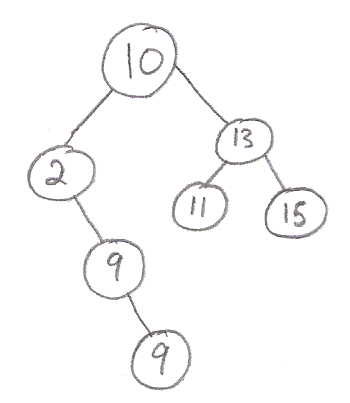
10

***Solution 13:***

**When deleting a node in a binary tree, it is swapped with the minimum node in its right subtree and after that we delete the node which contains the record we want to delete. If the node to be deleted doesn’t have a right subtree, then it is simply just removed.**

**In this case, we want to delete the record 6. The since 6 is the minimum node it can be deleted directly. To delete the node 12, we must replace it with the minimum of its right subtree which is 13, therefore we swap the records 13 and 12 and we delete 12 from the tree.**

**Therefore the tree becomes:**

****

***Problem 14 (Max Heaps):***

**What will become the following array of integers after using build heap for a max-heap? A ={ 2,0,7,9,6,8,1,5,3,4,7 }**

1. **A = { 9,8,7,6,5,4,3,2,1,0,7 }**
2. **A = { 4,2,0,1,7,6,3,8,5,9,7}**
3. **A = { 9,5,8,3,6,7,1,0,2,4,7 }**
4. **A = { 9,6,8,5,4,7,1,0,3,2,7 }**

**None of the above**

***Solution 14:***

**First we build the heap:**

6

4

7

8

1

0

7

2

9

5

3

**After that we have to siftdown all the non-leaf nodes of this Heap. In sifting down if the largest of a node’s children is greater than the mother node, we swap it with the node. For example, we should start by sifting 6, we should swap it with 7. Then we should siftdown 9, however 5 and 3 are both less than 9 so we do nothing. After that we should siftdown 7, so we swap it with 8. And now we reach 0, first we should swap it with 9, and then after that we continue on sifting it down and swap it with 5. Finally we reach 2, first we should swap it with 9, then we continue on sifting it down and swap it with 7. Now 2 has 4 and 6 as children so we have to also sift it one more time and swap it with 6.**

**Therefore the heap becomes:**

6

4

2

7

1

7

8

9

5

0

3

**Therefore the right answer is: {9,7,8,5,6,7,1,0,3,4,2}. As a result it is None of the above.**

***Problem 15 (Max Heaps):***

**Redraw the heap on the right after calling RemoveMax 2 times.**

30

26

7

15

11

38

29

40

35

16

9

**Assume that siftdown is called after each call to**

**RemoveMax.**

***Solution 15:***

**Calling RemoveMax the first time:**

**First we replace the maximum 40 with the last element and delete the last element. So the heap becomes:**

30

26

15

11

38

29

7

35

16

9

**Then we siftdown our new maximum which is 7. In sifting down if the largest of a node’s children is greater than the mother node, we swap it with the node. As a result 7 will be swapped with 38, then it will be swapped with 35 and after that it will be swapped with 16. So, the heap becomes:**

30

26

15

11

35

29

38

16

7

9

**Similarly we call RemoveMax for the second time and after fully executing it the heap becomes:**

26

15

11

30

29

35

16

7

9

***Problem 16 (Max Heaps):***

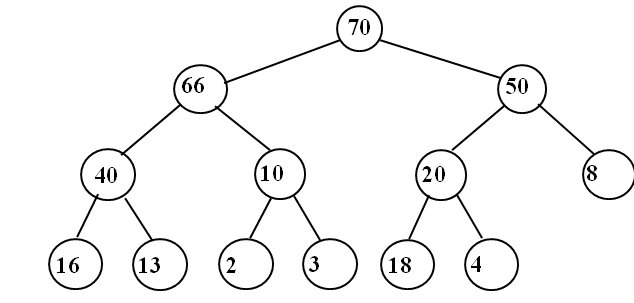
**Apply Build Heap to the following Maxheap:**



***Solution 16:***

**To build the heap we have to siftdown all the non-leaf nodes of this Heap. In sifting down if the largest of a node’s children is greater than the mother node, we swap it with the node Therefore we start with the node 4, we must swap it with the largest of it children which is 20. We move now to the node 10, we must swap it with 66. 70 is greater than both its children so we leave it as it is. Consider the node 8, it should be swapped with 50. 16 is first swapped with 70, now 16 has 40 and 13 as its children, so it should be swapped with 40. Its remains that we siftdown 3, first we swap it with 70, then we continue and swap it with 66. Finally 3 now has 10 and 2 as its children so we have to swap it with 10.**

**Therefore the MaxHeap becomes:**



***Problem 17 (Max Heaps):***

**What will become the following array of integers after using build heap for a max-heap**

**A= [10,11,65,9,28,12,44,14,51,63,62,]**

***Solution 17:***

**First we build the heap:**

28

6363

62

12

44

110

65

10

9

14

51

**After that we have to siftdown all the non-leaf nodes of this Heap. In sifting down if the largest of a node’s children is greater than the mother node, we swap it with the node. For example, we should start by sifting down 28, so we should swap it with 63. After that we should siftdown 9, so we swap it with 51. Then we should sift 65, but it is greater than both its children so we leave it as it is. And now we reach 11, first we should swap it with 63, and then after that we continue on sifting it down and swap it with 62. Finally we reach 10, first we should swap it with 65, now 10 has 12 and 44 as children so we have to also sift it one more time and swap it with 44.**

**Therefore the heap becomes:**

62

28

11

12

10

63

44

65

51

14

9

**Therefore the right answer is: {65,63,44,51,62,12,10,14,9,28,11}.**