4a.Build step by step a min-heap for 27, 13, 56, 35, 48, 8, 18, 67, 5, 62, 7 by inserting those elements (in order) into an initially empty heap. Show how the heap looks like (in a tree form and in an array form) after each insertion of the first 10 elements, then show the insertion of the last element (7) step by step in tree form and array form.

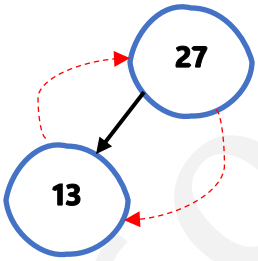
Insert 27

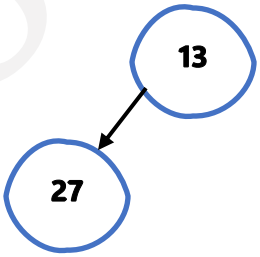
27									

Insert 13

27 is greater than 13.

Lets swap

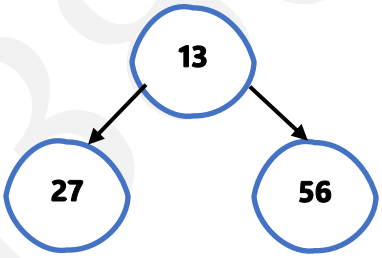




27	13								

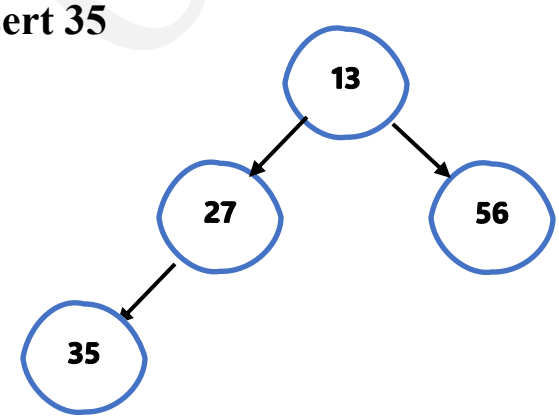
13	27								

Insert 56



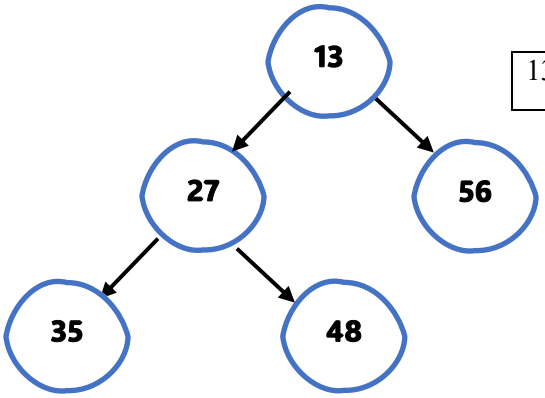
13	27	56							

Insert 35



13	27	56	35						

Insert 48

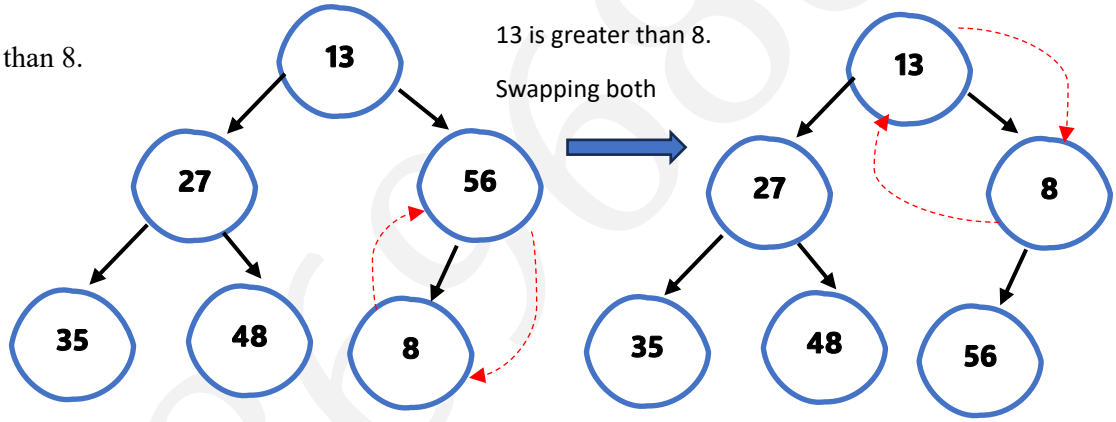


13	27	56	35	48						

Insert 8

56 is greater than 8.

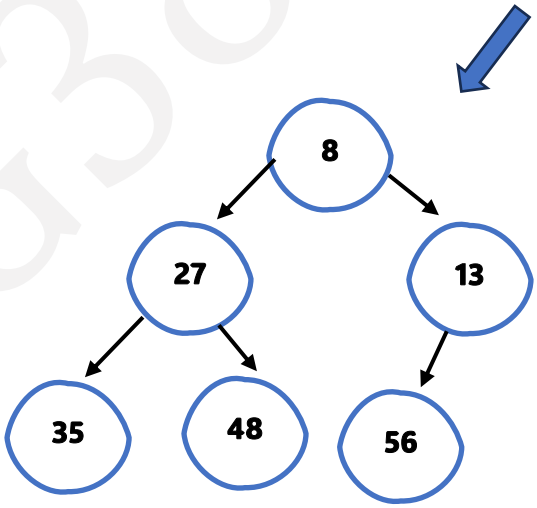
Lets swap.



13 is greater than 8.

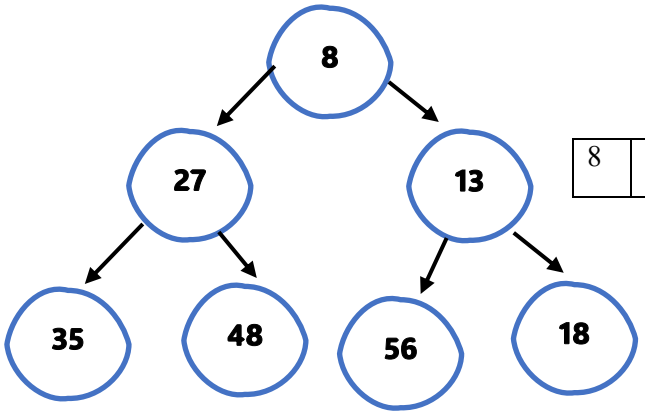
Swapping both

13	27	8	35	48	56					



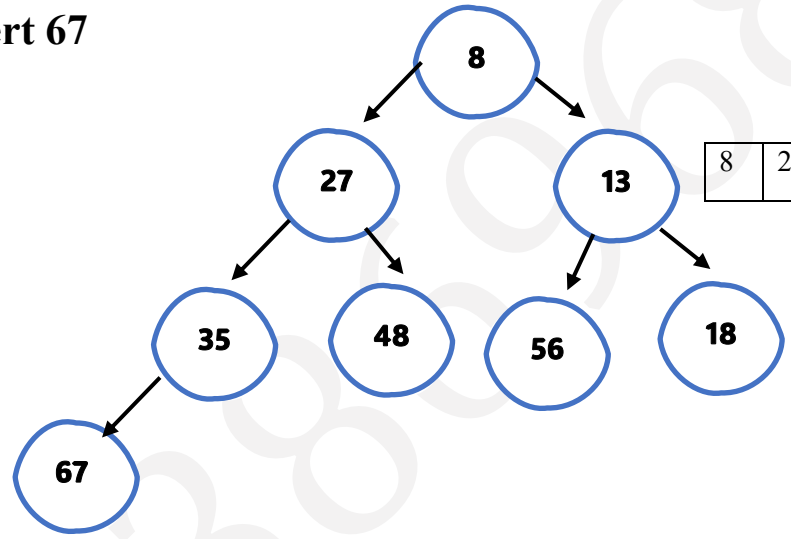
8	27	13	35	48	56					

Insert 18



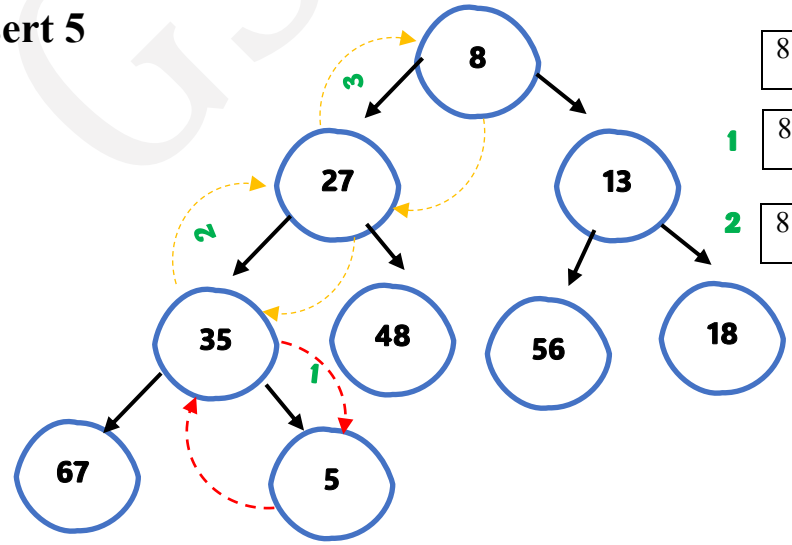
8	27	13	35	48	56	18				

Insert 67



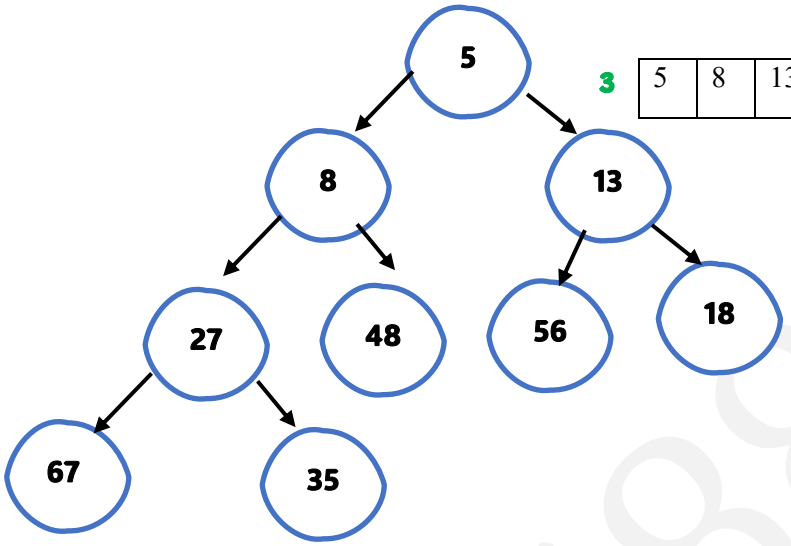
8	27	13	35	48	56	18	67			

Insert 5



	8	27	13	35	48	56	18	67	5		
1	8	27	13	5	48	56	18	67	35		
2	8	5	13	27	48	56	18	67	35		

Since 35 is greater than 5, we swapped it. Yet 27 is now greater than 5, we swap it. Now, 8 is greater than 5. We swap it. This step takes 3 swaps, making 5 to be the root element.

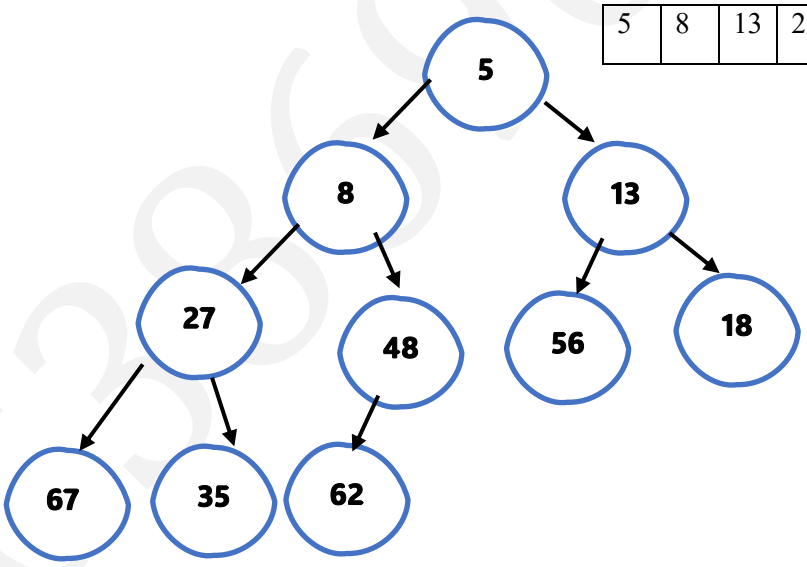


3

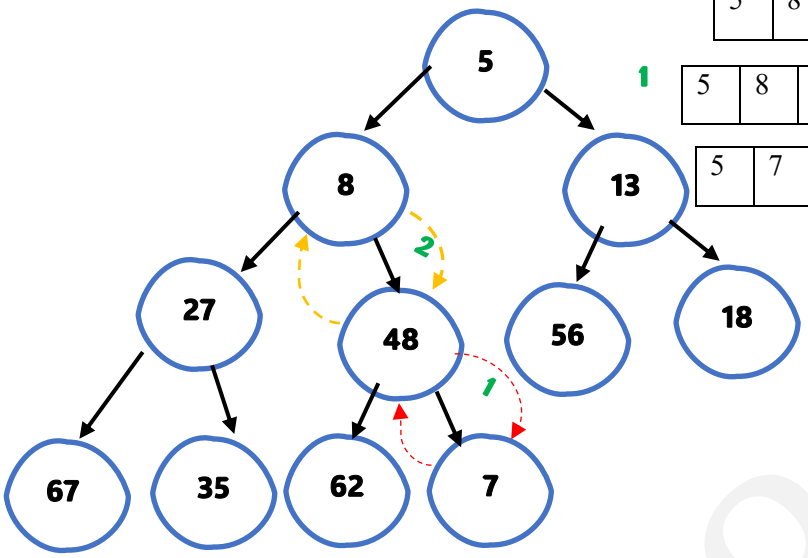
5	8	13	27	48	56	18	67	35		

Insert 62

5	8	13	27	48	56	18	67	35	62	



Insert 7



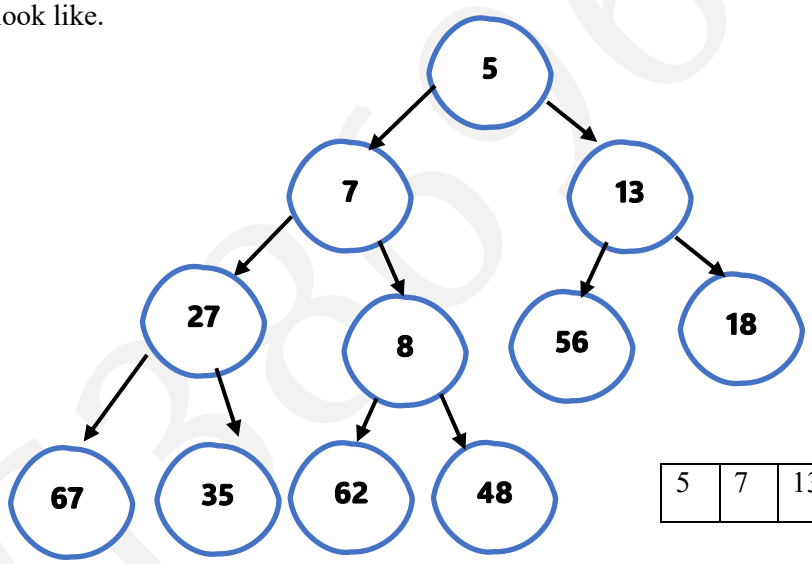
5	8	13	27	48	56	18	67	35	62	7

5	8	13	27	7	56	18	67	35	62	48

5	7	13	27	8	56	18	67	35	62	48

2

Since 48 is greater than 7, we swap both of them. Now 8 is greater than 7, hence we swap them making the tree look like.

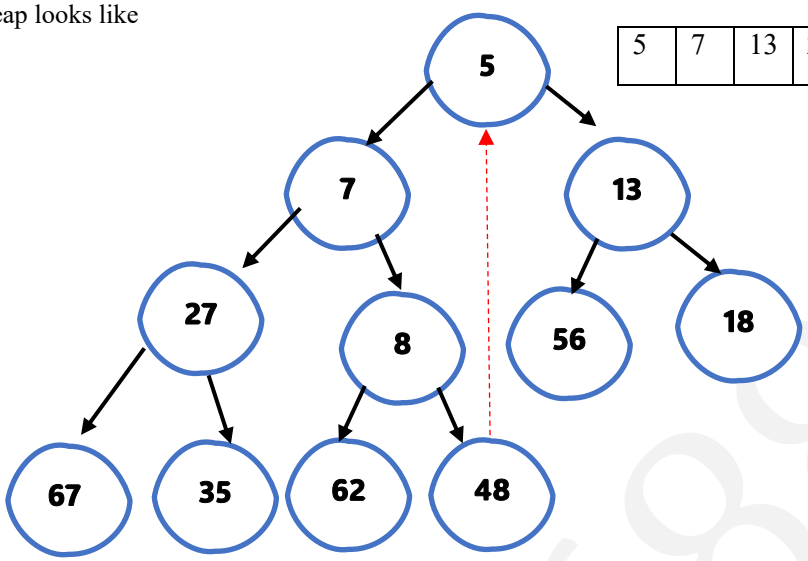


5	7	13	27	8	56	18	67	35	62	48

Final Min heap and final array

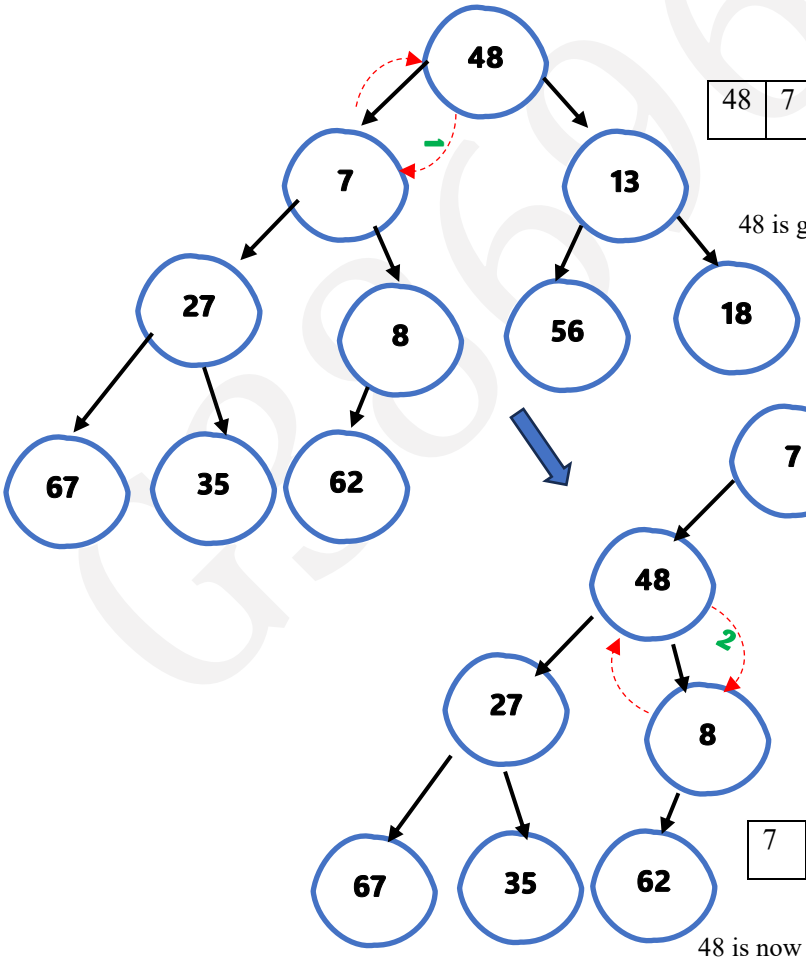
4. b. Perform delete-min() on the min-heap you built in part a. Show how the heap looks like (in a tree form and array form) after each step.

Our min heap looks like



5	7	13	27	48	56	18	67	35	62	48

Deleting 5 from our min heap tree and transferring the last node “48” to root.

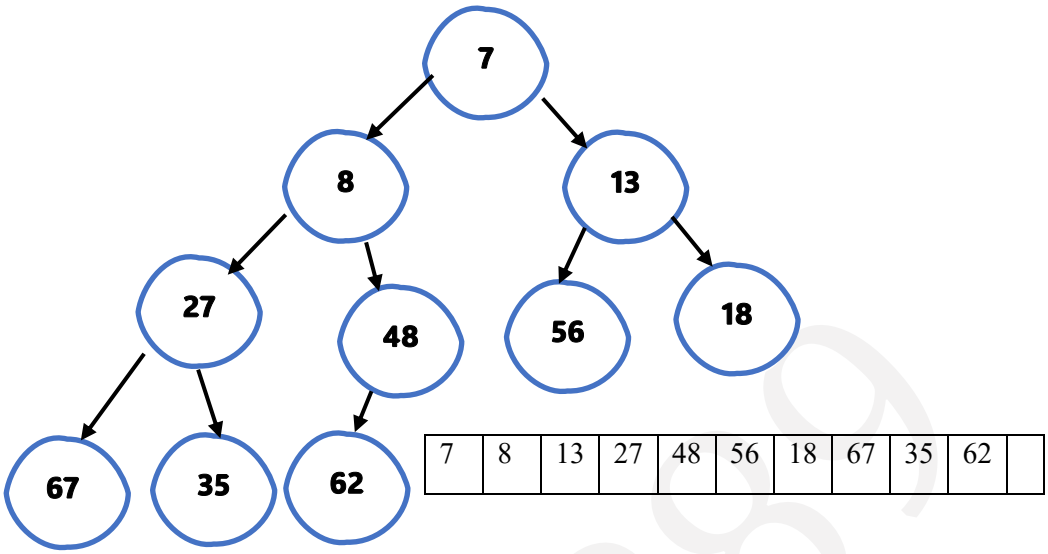


48	7	13	27	8	56	18	67	35	62	

48 is greater than 7. So we swap them.

7	48	13	27	8	56	18	67	35	62	

48 is now greater than 8. Now we swap them



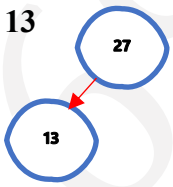
7	8	13	27	48	56	18	67	35	62	

Our final heap and array looks like above figures

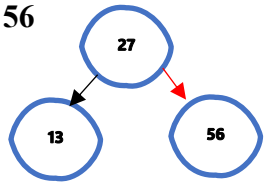
4c. Insert 27, 13, 56, 35, 48, 8, 18, 67, 5, 62, 3, 12, 14, 7, 10, 16, 17, 15 (as you read them) into an originally empty binary search tree. Show how the tree looks like after each insertion. Perform delete (13) on this tree step by step, showing how the tree looks like after each step

Insert 27

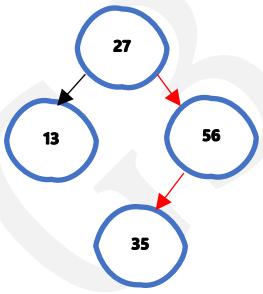
Insert 13



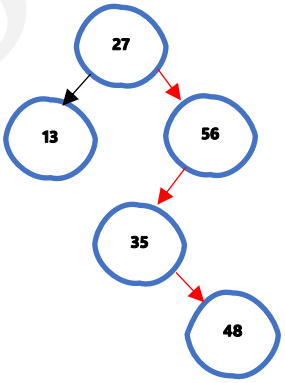
Insert 56



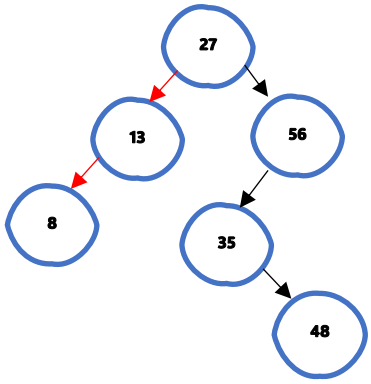
Insert 35



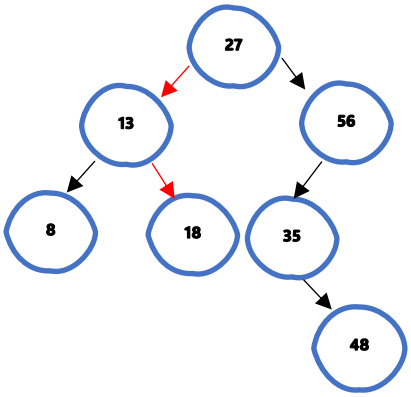
Insert 48



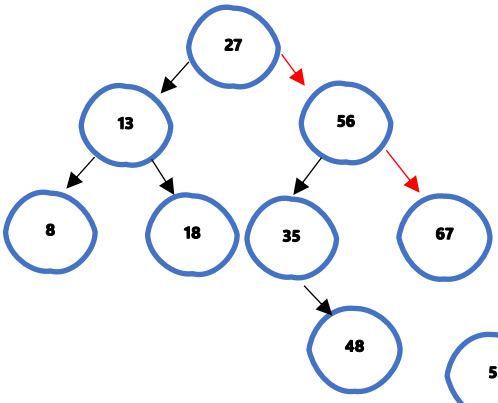
Insert 8



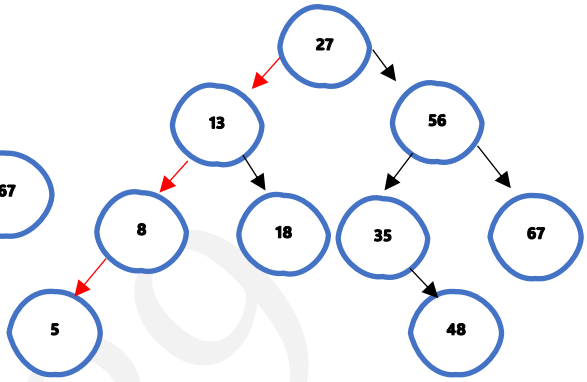
Insert 18



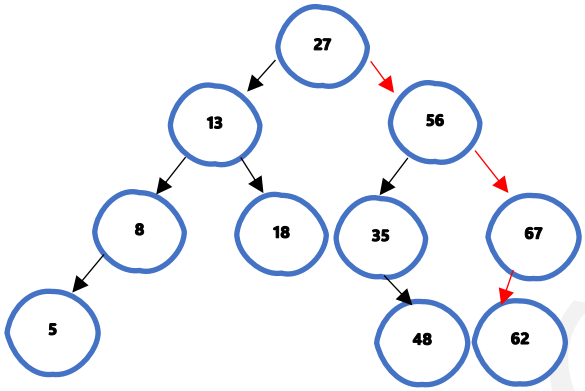
Insert 67



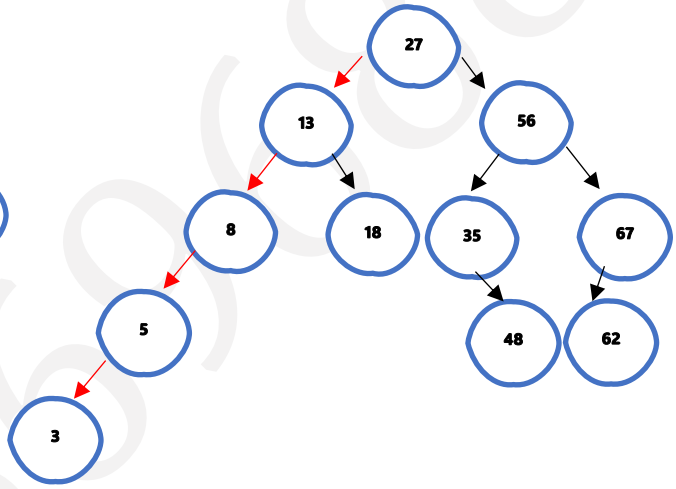
Insert 5



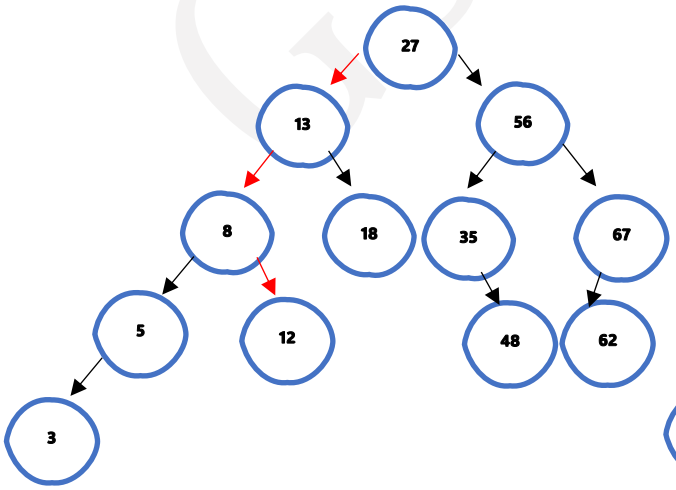
Insert 62



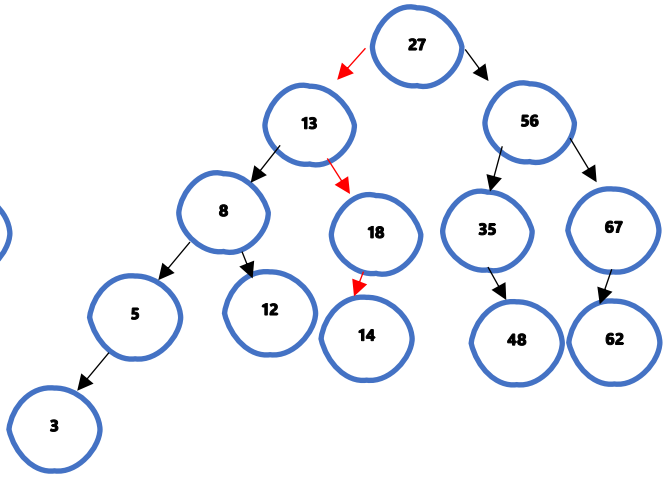
Insert 3



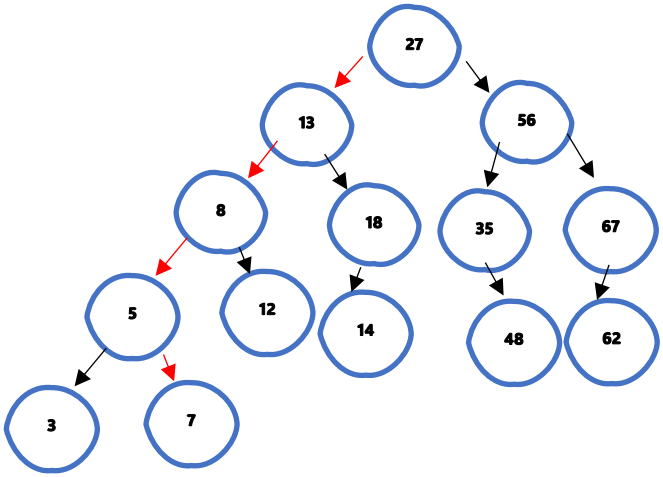
Insert 12



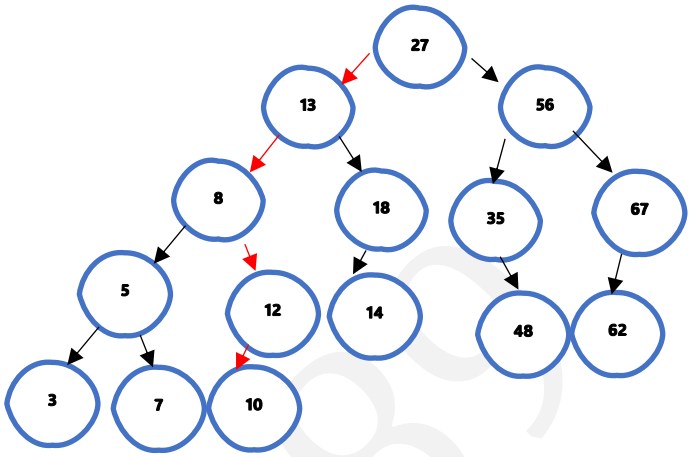
Insert 14



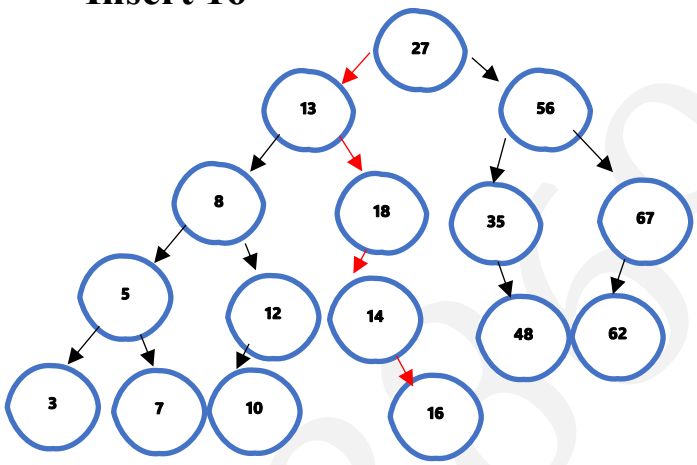
Insert 7



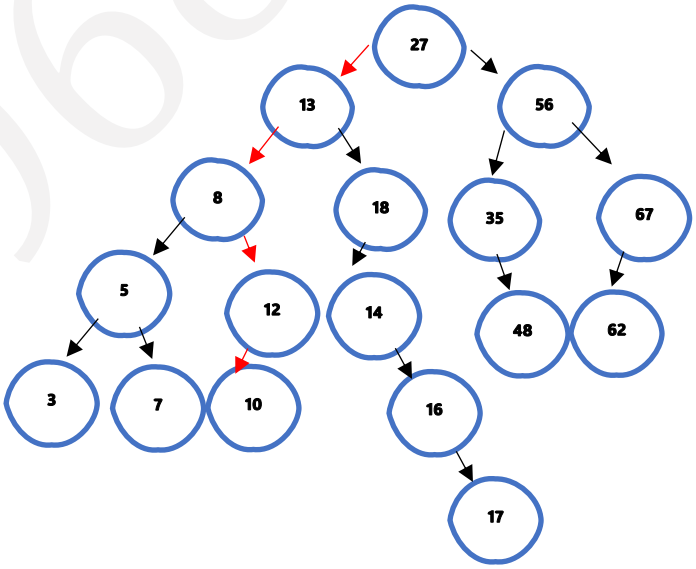
Insert 10



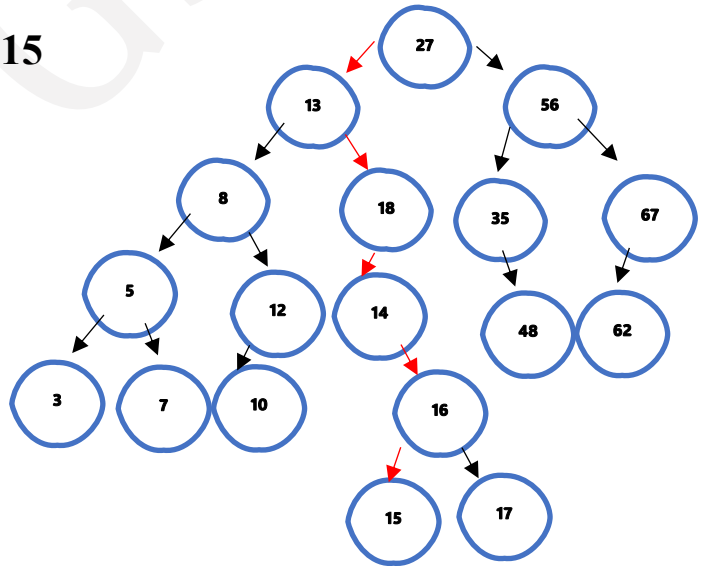
Insert 16



Insert 17



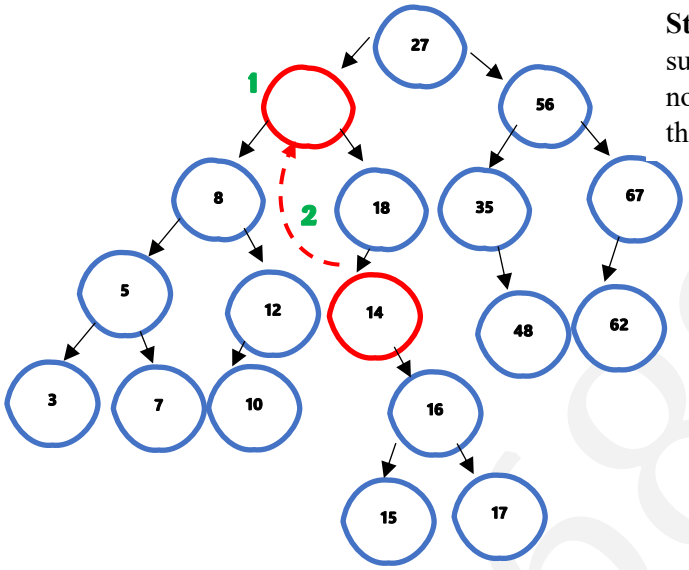
Insert 15

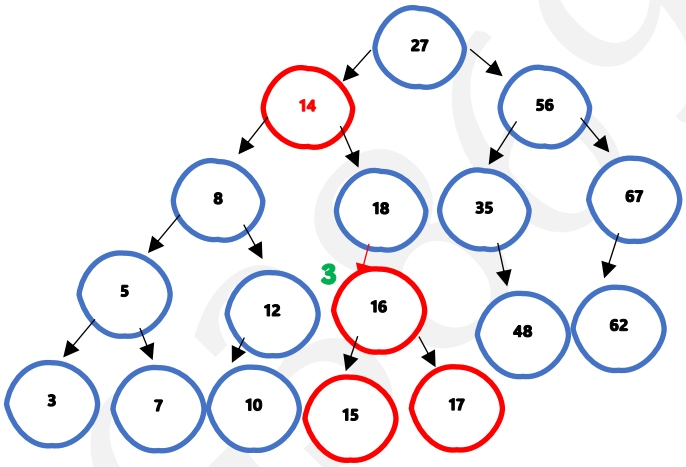


Deleting 13

Step 1 : Delete 13 from the node value.

Step 2 : 14 is the minimum node in the right subtree of node 13. So we move 14 to empty node. And the children of 14 will become the children of 18 in the left subtree.





Step 3: After inserting 14 in place of empty node. The right subtree of 14 becomes the left subtree of 18 now.

The BST after deleting the node 13

3.a Let T be a binary tree and assume that every node has one numerical data field (called"data") of type double. Write a recursive algorithm that takes as input the tree T and computes the following values: (1) number of nodes in T that have positive values in the data field, (2) number of nodes in T that have negative values in the data field, and (3) number of Nodes in T that have value 0 in the data field. Note: You should give one single algorithm (not three separate algorithms) that computes all three numbers together.

Answer

// driver code start
//GetNodeDetails is the function written just to call the NodeDeterminer procedure.
//driver code end

```
procedure GetNodeDetails(Tree T)
begin
     nodePtr p;
     p = T;
     int positiveNodes = 0,negativeNodes = 0, zeroNodes = 0;
     NodeDeterminer(T, positiveNodes,negativeNodes,zeroNodes)
end
```

//Actual procedure starts

```
procedure NodeDeterminer(Tree T, output positiveNodes,
negativeNodes, zeroNodes)
begin
    nodePtr p;
    p = T;
    while(p != null)
    begin
        if(p.data == 0)
            zeroNodes++;
        else if(p.data>= 1)
            positiveNodes++;
        else
            negativeNodes++;
        end if
    NodeDeterminer(p.Right);
    NodeDeterminer(p.Left);
    end while
end
```

The above algorithm computes the task in **O(n)** time. Because, we are using breadth first search and every node is visited exactly once.

3b. Write a recursive algorithm that takes as input a binary tree T and returns whether or not the tree is full. Give the time complexity of your algorithm.

Answer

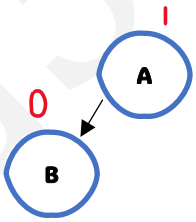
```
func DriverCode(Tree T)
begin
    bool isTreeFull = DetermineIfTreeIsFull(T)
end

func DetermineIfTreeIsFull(Tree T)
begin
    nodePtr = p;
    p = T;
    if(p==null)
        return true;
    while(p!=null)
    begin
        if((p.left == null && p.right != null) || (p.right == null && p.left != null))
            return false
        else
            return DetermineIfTreeIsFull(p.left) && DetermineIfTreeIsFull(p.right)
    end
end
```

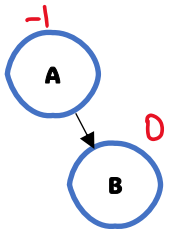
The above algorithm takes **O(n)** time to compute. As we visit each node exactly once, the code takes O(n) time complexity. In the full binary tree, each internal node must contain two nodes. So, here we are checking the failing conditions in the if block. If any node contains a single child i.e either only right or only left child, then we can safely discard tree as not full. Else we can continue to check recursively each subsequent child nodes. The "&&" operation is to check if all the nodes satisfies the condition. If any one of the node doesn't satisfy the case, we can discard tree as full.

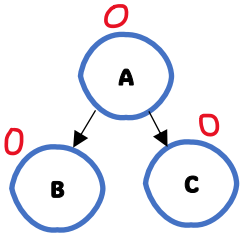
2a. Show all AVL trees of height 0 and height 1, and show 8 AVL trees of height 2.

AVL Trees Of Height 0

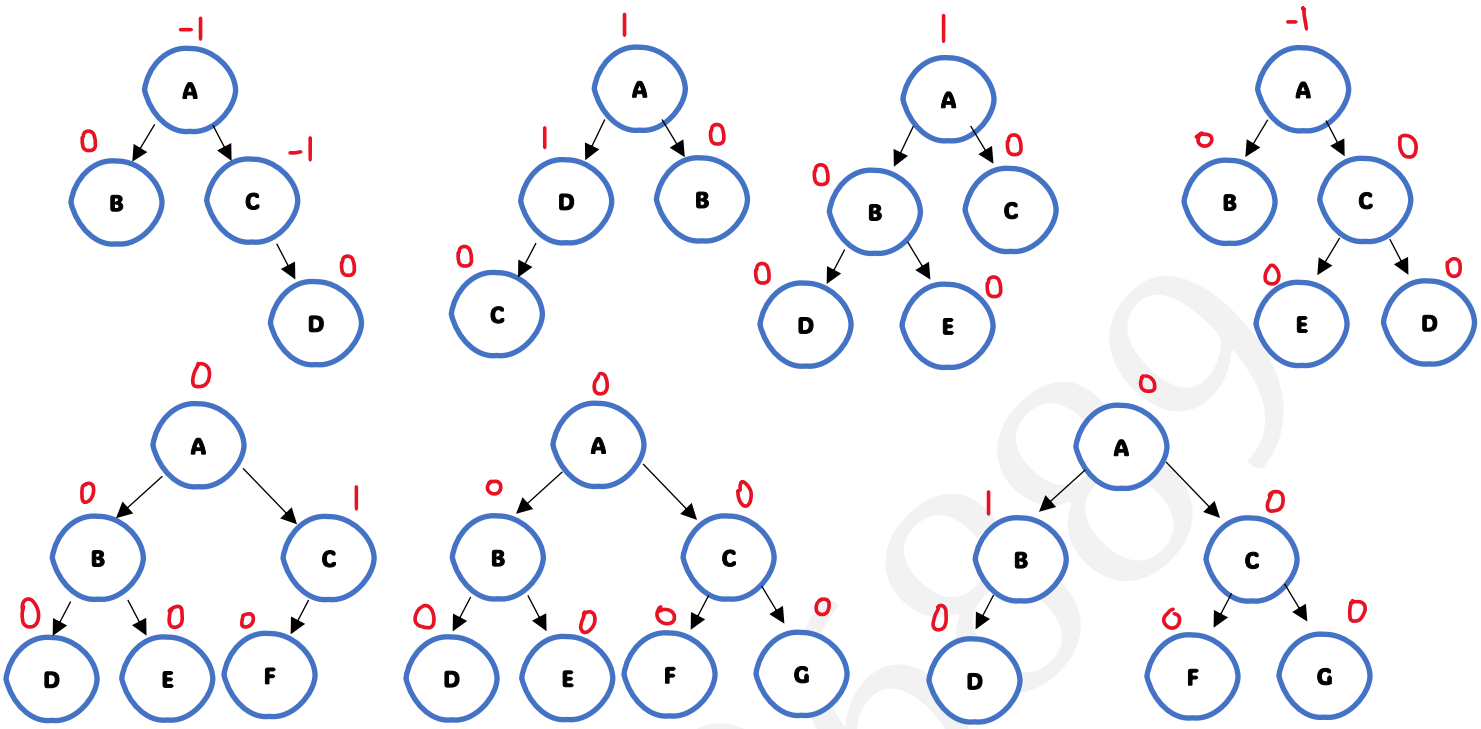
AVL Trees Of Height 1

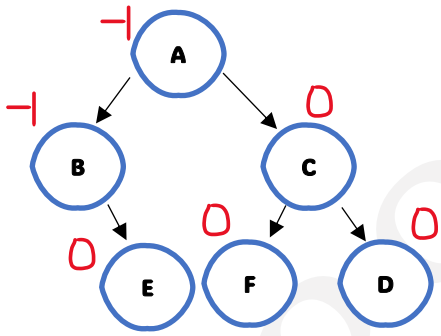






AVL Trees Of Height 2

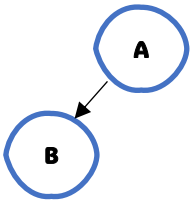




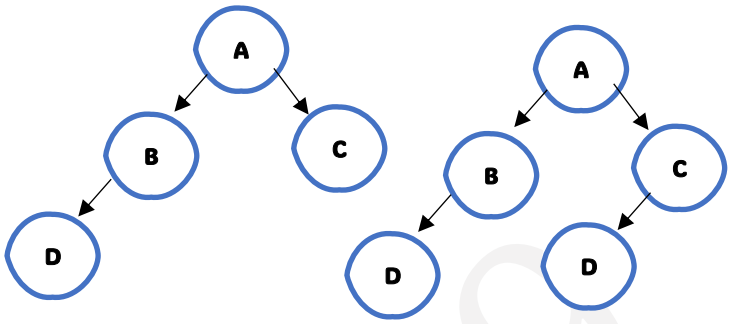
2b. A *left-heavy* AVL is an AVL where for every internal node, the height of its left subtree is one more than the height of its right subtree. Show all left-heavy AVL trees of height 0, height 1, height 2, and height 3.

Left Heavy AVL trees of height 0 (Though there is no left and right subtrees)

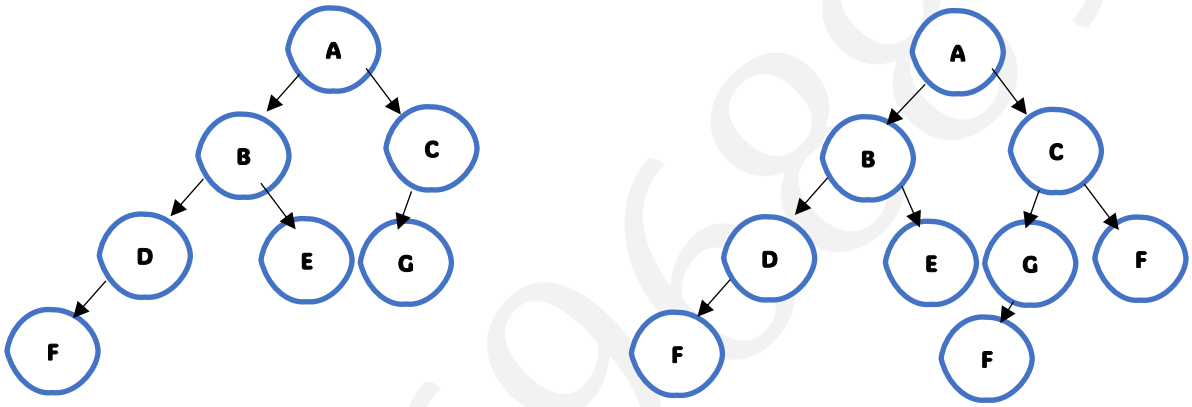
Left Heavy AVL trees of height 1



Left Heavy AVL trees of height 2



Left Heavy AVL trees of height 3



2 c. Let $N(h)$ be the number of nodes of a left-heavy AVL tree of height h. Express $N(h)$ in terms of $N(h - 1)$ and $N(h - 2)$.

Answer :

Given to represent N(h) in terms of N(h-1) and N(h-2). N(h) is defined as the minimum number of nodes at a given height "h" of an AVL tree.

In a left heavy AVL tree, each internal node is left heavy. That is left subtree is at most differ by 1 in height than right subtree.

Let h=0, then at a height h=0, number of nodes in a left heavy AVL tree = 1, i.e N(0) = 1, which is a root node.

For height h=1, number of nodes in a left heavy AVL tree = 2. Root node + 1 node on left side +0 nodes in right side.

N(1) = 1+1+0

For height h=2, number of nodes in a left heavy AVL tree = root node + nodes on left subtree + node in right subtree

Nodes in left subtree is N(t) where t is the height from the root node. If root node is at height h, then the height of left subtree is at height h-1 (since we exclude root node), ie. N(h-1)

Nodes in right subtree is N(t) where t is the height from the root node. If root node is at height h, then the height of right subtree is height h-2 (since we exclude root node and the height difference between it and left subtree is at most 1 and given it is a left heavy AVL tree), ie. N(h-2)

We can deduce that
N(2) = 1 + N(2-1) + N(2-2)

This pattern follows for every h>=2 and base case is N(0) =1 and N(1) =2

Hence we can conclude that

N(h) = 1+N(h-1)+N(h-2)

2d. It can be shown that $N(h) = a\left(\frac{1+\sqrt{5}}{2}\right)^h + b\left(\frac{1-\sqrt{5}}{2}\right)^h - 1$ for some constants *a* and *b* (you do not have to prove this formula). Find the values of *a* and *b*.

Answer :

Given $N(h) = a\left(\frac{1+\sqrt{5}}{2}\right)^h + b\left(\frac{1-\sqrt{5}}{2}\right)^h - 1$

N(h) is defined as the number of nodes at a given height "h" of an AVL tree.

We know that for N(0) = 1 and N(1) = 2

Substituting h=0 and h=1 in the given equation.

N(0) = 1 = $a\left(\frac{1+\sqrt{5}}{2}\right)^0 + b\left(\frac{1-\sqrt{5}}{2}\right)^0 - 1 \rightarrow a + b - 1$

$$a + b - 1 = 1$$

$$a + b = 2 \rightarrow \boldsymbol{eq1}$$

N(1) = 2 = $a\left(\frac{1+\sqrt{5}}{2}\right)^1 + b\left(\frac{1-\sqrt{5}}{2}\right)^1 - 1$

$$\frac{a}{2} + \frac{a\sqrt{5}}{2} + \frac{b}{2} - \frac{b\sqrt{5}}{2} - 1 = 2 \rightarrow \boldsymbol{eq2}$$

Solving both the equations. That is substituting a = 2-b

$$\frac{2-b}{2}+\frac{(2-b)\sqrt{5}}{2}+\frac{b}{2}-\frac{b\sqrt{5}}{2}=3$$

$$2-b+(2-b)\sqrt{5}+b-b\sqrt{5}=6$$

$$2-b+2\sqrt{5}-b\sqrt{5}+b-b\sqrt{5}=6$$

$$2+2\sqrt{5}-2b\sqrt{5}=6$$

$$1+\sqrt{5}-b\sqrt{5}=3$$

$$-b=\frac{2-\sqrt{5}}{\sqrt{5}}$$

$$b=\frac{\sqrt{5}-2}{\sqrt{5}}\rightarrow 1-\frac{2}{\sqrt{5}}$$

$$\boldsymbol{b=1-\frac{2}{\sqrt{5}}\rightarrow eq3}$$

Substituting eq3 in eq1

$$a+b=2$$

$$a+\mathbf{1}-\frac{\mathbf{2}}{\sqrt{\mathbf{5}}}=2$$

$$\boldsymbol{a=1+\frac{2}{\sqrt{5}}}$$

Hence the values of a and b are $\mathbf{1}+\frac{\mathbf{2}}{\sqrt{\mathbf{5}}}$ and $\mathbf{1}-\frac{\mathbf{2}}{\sqrt{\mathbf{5}}}$.

Problem 1

1a. Show by induction on *n* that $1^2+2^2+3^2+\cdots+n^2=n(n+1)(2n+1)/6$

Answer

Base Step :

Let us check if substituting 1 satisfies the equation; for n=1

1^2 = 1(1+1)(2(1)+1)/2 => 1*2*3/6 => 6/6 => 1

Base case satisfied.

Inductive Hypothesis :

Let us assume that for any number m-1 where m >=1, it satisfies the equation. That is

$1^2 + 2^2 + 3^2 + \cdots + (m-1)^2 = (m-1)(m)(2(m-1)+1)/6 => = \frac{(m-1)(m)(2m-1)}{6} \rightarrow eq1$

Inductive Step :

Prove the same for . That is

$$1^2 + 2^2 + 3^2 + \cdots + (m)^2 = \frac{(m)(m+1)(2m+1)}{6}$$

$$1^2 + 2^2 + 3^2 + \cdots + (m-1)^2 + m^2 = \frac{m(m+1)(2m+1)}{6} \rightarrow eq\ 2$$

We know that from $\boldsymbol{eq1}$

$$1^2 + 2^2 + 3^2 + \cdots + (m-1)^2 = \frac{(m-1)(m)(2m-1)}{6}$$

Substituting the above in eq 2

$$\frac{(m-1)m(2m-1)}{6} + m^2 = (m^2 - m)(2m-1)/6 + m^2$$
$$= (2m^3 - m^2 - 2m^2 + m + 6m^2)/6$$
$$\frac{\boldsymbol{2m^3 + 3m^2 + m}}{\boldsymbol{6}} = \frac{\boldsymbol{m(m+1)(2m+1)}}{\boldsymbol{6}}$$

Hence, by the mathematical induction, we proved that the given equation holds true for all n>=1

1b. Let $T(n)$ be defined recursively as follows: $T(1) = c$, and $T(n) = 5T\left(\frac{n}{5}\right) + c\ \forall\ n \geq 5$

where c is an arbitrary constant, and $\boldsymbol{n = 5^k}$ for some non-negative integer k. Prove by induction on k that $T(n) = \frac{5c}{4}n - \frac{c}{4}$

Answer :

Base step :

Let us substitute $\boldsymbol{n = 5^k}$, $\boldsymbol{T(5^k) = 5T\left(\frac{5^k}{5}\right) + c \rightarrow 5T(5^{k-1}) + c}$ and

$$\boldsymbol{T(5^k) = \frac{5c5^k}{4} - \frac{c}{4} = \frac{5^{k-1}c}{4} - \frac{c}{4}}$$

Let us check if the equation satisfies for the base case by substituting k=0

$$T(1) = \frac{5c(1)}{4} - \frac{c}{4} \rightarrow \frac{5c}{4} - \frac{c}{4} = c$$

The base case is satisfied.

Inductive Hypothesis :

Let us assume that the $T(5^k)$ holds true for any value k>=0. Then for some k=m the equation $T(5^m) = \frac{5c(5^m)}{4} - \frac{c}{4} = \frac{5^{m+1}}{4} - \frac{c}{4} \rightarrow (eq1)$ holds true.

Inductive Step :

Then prove the same for k = m+1 that $T(5^{m+1}) = \frac{5c5^{m+1}}{4} - \frac{c}{4} = \frac{5^{m+2}c}{4} - \frac{c}{4}$

From the above recurrence relation,

$$T(5^{m+1}) = 5T(5^{m+1-1}) + c = 5T(5^m) + c \rightarrow (eq\ 2)$$

Substituting eq 1 in eq 2.

$$5\left(\frac{5^{m+1}}{4} - \frac{c}{4}\right) + c = \frac{5^{m+2}}{4} - \frac{5c}{4} + c = \frac{5^{m+2}}{4} - \frac{c}{4}$$

Hence for k = m+1, the recurrence relation holds true. We can say that for $\forall\ n \geq 5$, where $n = 5^k$, $T(n) = \frac{5c}{4}n - \frac{c}{4}$ holds true and where $T(n) = 5T\left(\frac{n}{5}\right) + c$

1c. Let $T(n)$ be defined recursively as follows: $T(1) = c$, and $T(n) = 2T(\lfloor n/2 \rfloor) + cn$, for all integers $n \geq 2$, where c is an arbitrary positive constant and $\lfloor x \rfloor$ for any number x is the *floor* value of x. Prove by induction on n that $T(n) \leq cn\log n + cn$ for all integers $n \geq 1$.

Answer :

Base Step :

Let us substitute n=1 in T(n), then

$$T(1) \leq c(1)log(1) + c(1) \rightarrow c(0) + c = c$$

T(1) = c, hence satisfied.

Inductive Hypothesis :

Let us assume that for some n=k, the relation holds true. That is eq1 holds true for every n=k.

$$T(k) \leq cklogk + ck \rightarrow eq\ 1$$

Inductive Step :

Then prove that for n=k+1 for some n >= 1, T(n) also holds true. That is

$$T(k+1) \leq c(k+1)\log(k+1) + c(k+1)$$

From the recurrence relation,

$$T(k+1) = 2T(\lfloor(\mathrm{k}+1)/2\rfloor) + c(k+1)$$

We know that if $\left\lfloor\frac{n}{2}\right\rfloor$ *is even then* $\frac{n}{2}$ *and when* $\left\lfloor\frac{n}{2}\right\rfloor$ *is odd then* $\frac{n-1}{2}$. Then for $\left\lfloor\frac{k+1}{2}\right\rfloor$ it is $(\frac{k+1}{2})$

For the case where k+1 is even, then k+1 = 2q (since k+1 can be represented as multiple of 2 as it is an even number), and $\left\lfloor\frac{2q}{2}\right\rfloor = q$

$$T(2q) = 2T(q) + c(q) \rightarrow \boldsymbol{eq\ 2}$$

Substituting eq 1 in eq2, then

$$T(2q) \leq 2(cqlogq + cq) + c(q)$$

$$T(2q) \leq 2(cqlogq) + 2cq \rightarrow putting\ back\ k+1 = 2q, then$$

$$\boldsymbol{T(k+1) \leq c(k+1)log(k+1) + c(k+1)}$$

For the odd case, where k+1 is odd, then k+1 wholly can be written as 2q+1(We know that if "a" is an integer, then a =2p if a is even and a= 2p+1 if a is odd). It means k=2q

We know that if $\left\lfloor\frac{n}{2}\right\rfloor$ *is even then* $\frac{n}{2}$ *and when* $\left\lfloor\frac{n}{2}\right\rfloor$ *is odd then* $\frac{n-1}{2}$. Then for $\left\lfloor\frac{k+1}{2}\right\rfloor$ it is $(\frac{k}{2})$

Substituting 2q in k(since k is now even because of 1 is subtracted from k+1, which makes k even), then $\frac{2q}{2} = q$.

$$T(2q) = 2T(q) + c(q) \rightarrow \boldsymbol{eq\ 3}$$

Substituting eq 1 in eq3, then

$$T(2q) \leq 2(cqlogq + cq) + c(q)$$

$$T(2q) \leq 2(cqlogq) + 2cq \rightarrow putting\ back\ k = 2q, then$$

$$\boldsymbol{T(k) \leq c(k)log(k) + c(k)}$$

Which is also true for

$$\boldsymbol{T(k+1) \leq c(k+1)log(k+1) + c(k+1)}$$

Hence, $T(n) \leq c(n)log(n) + c(n)$ holds true for every n >=1

1.d Let $T(n)$ be defined recursively as follows: $T(1) = c$, and $T(n) = 4T\left(\frac{n}{2}\right) + n^5$. Use the Master Theorem to find the Θ value of $T(n)$,

Answer

Master Theorem states that for a recurrence relation of type $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, where a is number of subproblems and b is the size of each subproblem and f(n) is a function that is asymptotically positive for a>=1 and b>1,

Then if

$$f(n) = O(n^{\log_b a - \varepsilon}), Then\ T(n) = \theta(n^{\log_b a})$$

$$f(n) = \theta\left(n^{\log_b a}\right), Then\ T(n) = \theta(n^{\log_b a} * logn)$$

$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ $for\ some\ constant\ \varepsilon > 0\ if\ af\left(\frac{n}{b}\right) \leq cf(n) for\ some\ c < 1\ for\ all\ sufficiently\ larger\ n, then\ T(n) = \theta(f(n))$

Where for $\varepsilon > 0$

In our case,

a = 4, b=2, f(n) = n^5

$$\log_2 4 = 2$$

clearly $f(n) > n^{\log_b a + \varepsilon}$ for some ε

and $\boldsymbol{af\left(\frac{n}{b}\right) \leq cf(n)} \rightarrow 4\left(\frac{n}{2}\right)^5 \leq cn^5 \rightarrow \frac{4}{32}n^5 \leq cn^5 \rightarrow c \geq \frac{1}{8}$

for some $\boldsymbol{c \geq \frac{1}{8}}$ $and\ less\ than\ 1\ the\ equation\ holds\ true$

Hence, case 3 applies for our problem.

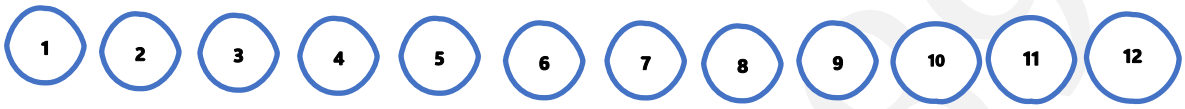
$$\boldsymbol{f(n) = \theta(n^5)}$$

Problem 5 :

Consider the elements 1, 2, ... , 12. Perform the following sequence of Unions (U) and Finds (F) using the *path compression* algorithm (i.e., 3rd implementation), show how the forest looks like after each operation, and display the PARENT array alongside each snapshot of the forest:
U(1,5) U(3,4) U(2,3) U(1,3) U(6,7) U(8,10) U(8,9) U(6,8) U(11,12) U(3,11) U(8,3) F(7)

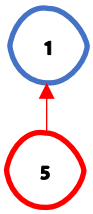
Answer :

i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



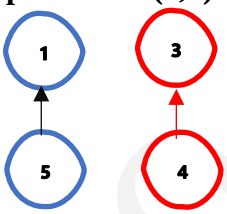
Initially, the every node is set to itself, i.e root node.

Operation U(1,5) :



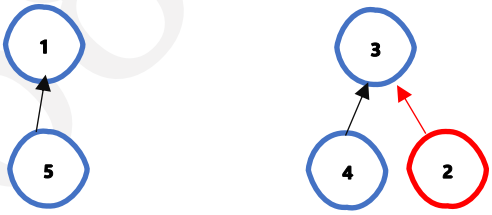
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	-2	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1

Operation U(3,4)



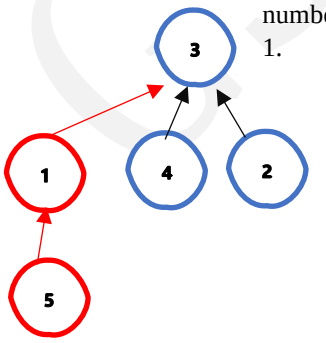
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**-2**	**-1**	**-2**	**3**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**

Operation U(2,3)



i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**-2**	**-3**	**-3**	**3**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**

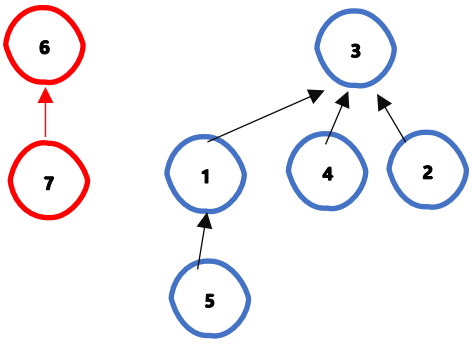
Operation U(1,3)



Making 3 as root of 1 because of more number of nodes that other set with root 1.

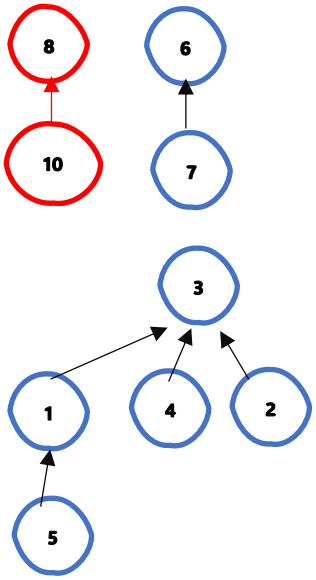
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-5**	**3**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**

Operation U(6,7)



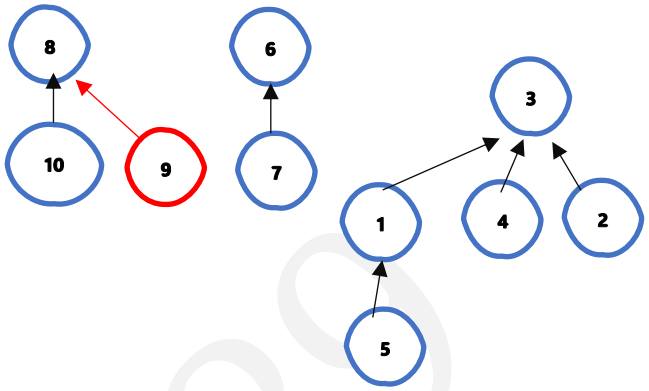
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-5**	**3**	**1**	**-2**	**6**	**-1**	**-1**	**-1**	**-1**	**-1**

Operation U(8,10)



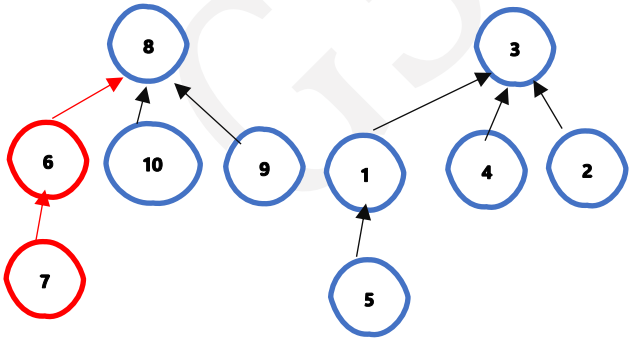
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-5**	**3**	**1**	**-2**	**6**	**-2**	**-1**	**8**	**-1**	**-1**

Operation U(8,9)



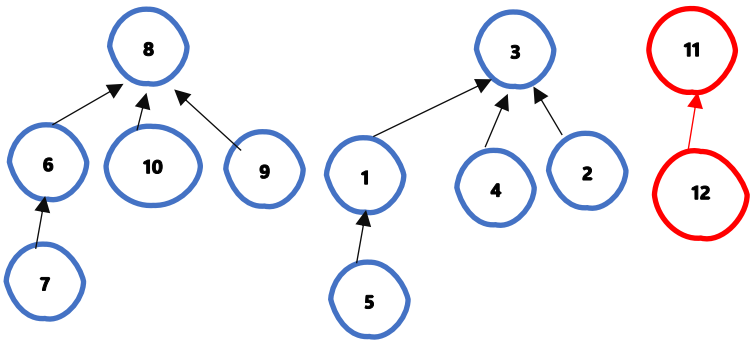
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-5**	**3**	**1**	**-2**	**6**	**-3**	**8**	**8**	**-1**	**-1**

Operation U(6,8)



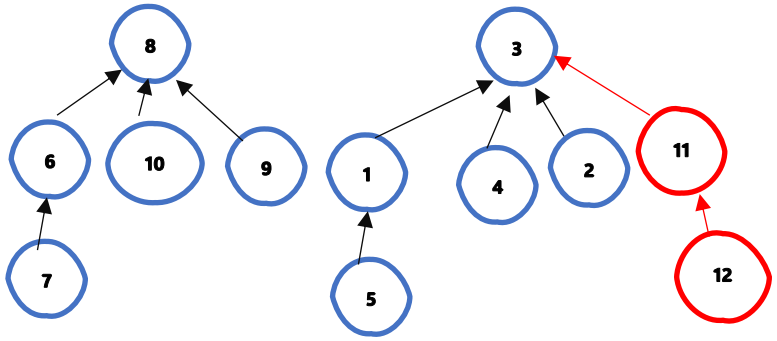
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-5**	**3**	**1**	**8**	**6**	**-5**	**8**	**8**	**-1**	**-1**

Operation U(11,12)



i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-5**	**3**	**1**	**8**	**6**	**-5**	**8**	**8**	**-2**	**11**

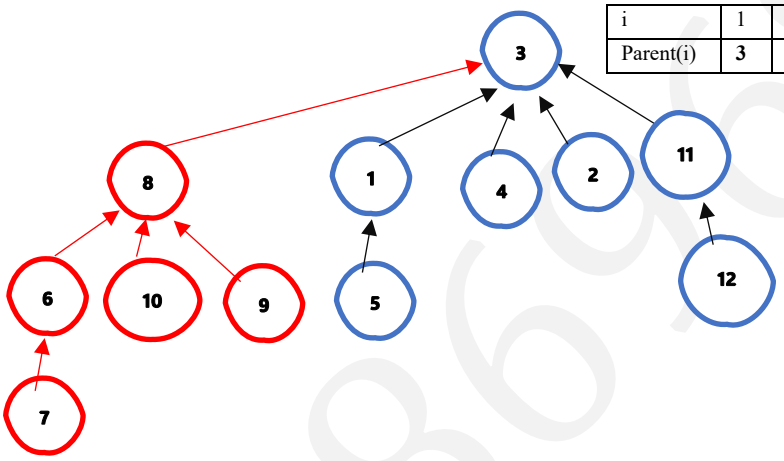
Operation U(3,11)



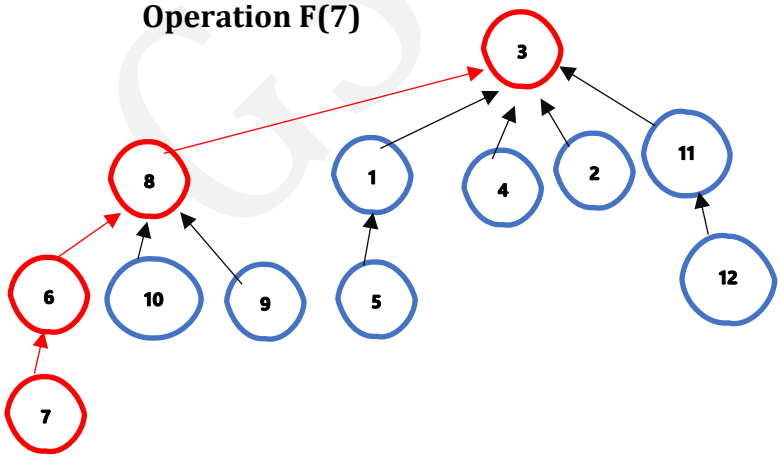
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-7**	**3**	**1**	**8**	**6**	**-5**	**8**	**8**	**3**	**11**

Operation U(8,3)

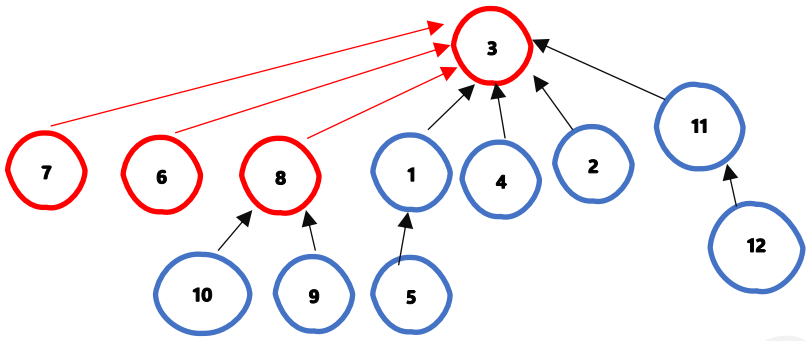
i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-12**	**3**	**1**	**8**	**6**	**3**	**8**	**8**	**3**	**11**



Operation F(7)



In the Path Compression Algorithm, we make each node in the path of "x" in find(x) ,the direct children to the root of the whole set/forest. That is 7, 6 (parent of 7), 8 (parent of 6) will be the direct children of 3.



i	1	2	3	4	5	6	7	8	9	10	11	12
Parent(i)	**3**	**3**	**-12**	**3**	**1**	**3**	**3**	**3**	**8**	**8**	**3**	**11**

7,6,8 are now direct /immediate children of 3.