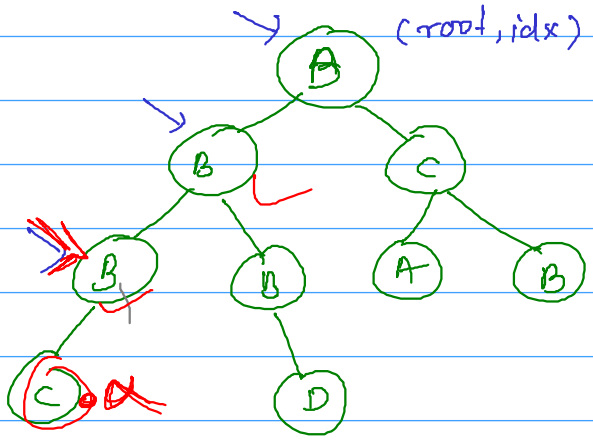
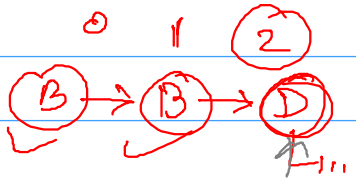
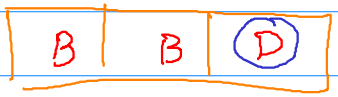
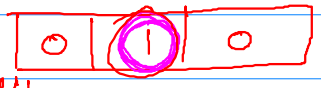
TREES



B B D

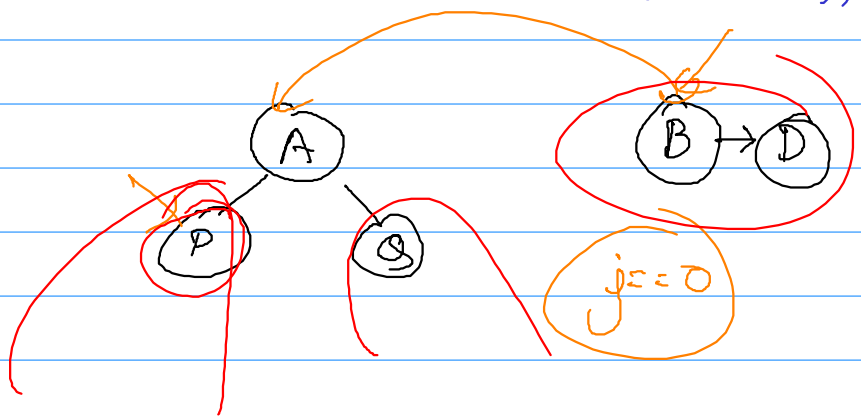




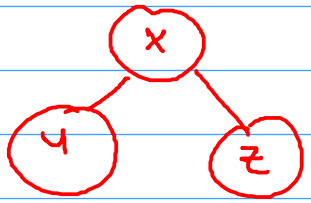



```
f(root, idx)
if(idx>=arr.size()) return true;
if(!root) return false;

if(root->val == arr[idx]  f(root->left, idx+1) ||
                          f(root->right, idx+1);

if(j==0)
    return f(root->left, 0) ||
           f(root->right, 0);

return f(root->left, lps[idx-1]) || f(root->left, lps[idx-1]);
}
```

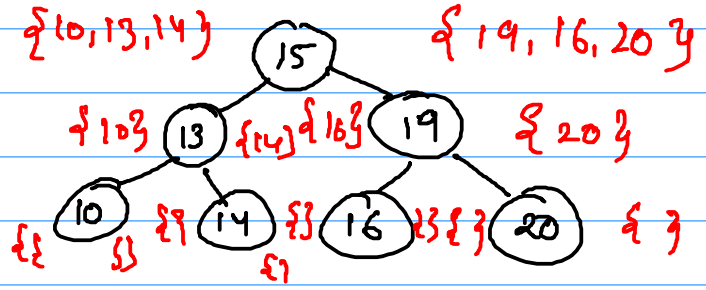
j lps[j-1]



Binary Search Tree:

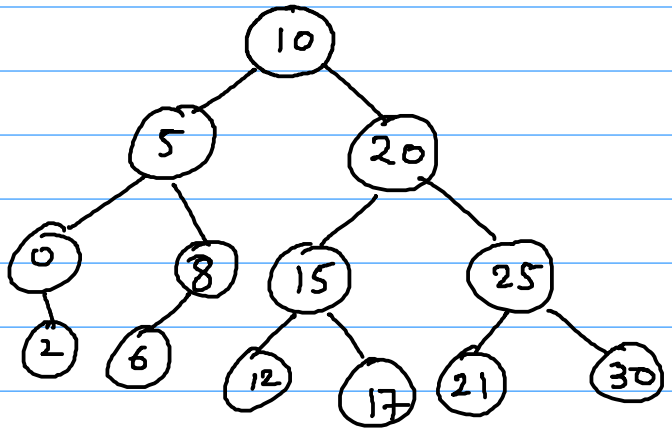


- all nodes left to the root will have values less than root→val
- all value which on right will be greater than root
- this property holds for every node.



valid BST.

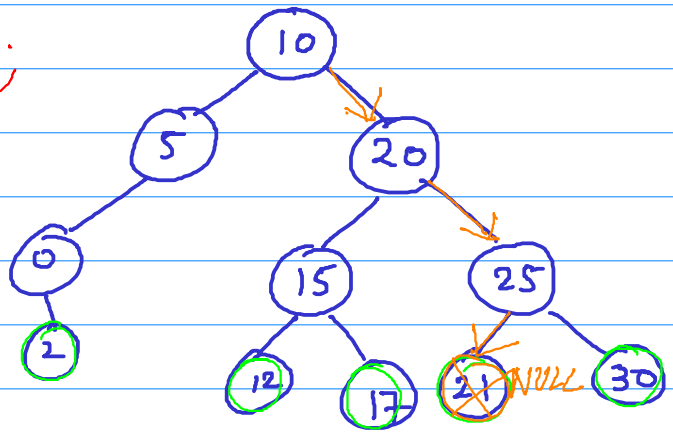
- Insert a node in BST.



```
Insert ( root, x ) {
if (!root) return  new Node(x);
if ( root->val < x)
        root->left = Insert(root->left, x);
else    root-> right : Insert( root->right,x);
return root;
}.
```

⊙ Delete a node from BST.

fun(root, x);



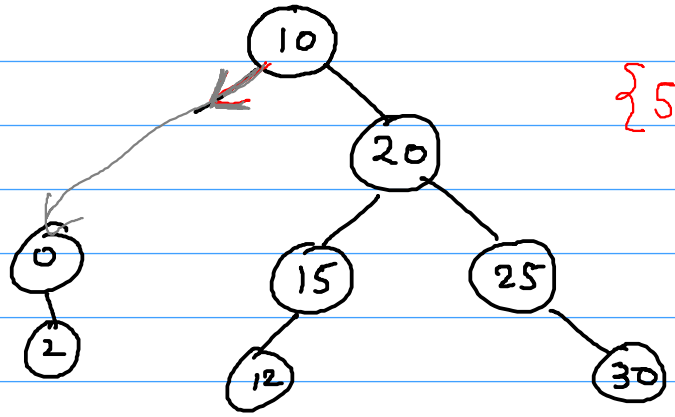
① Delete a leaf node.

21

```
Delete ( root, x )
 if(! root) return NULL;
 if (root-> val > x)
 root->left = Delel (root->left,x)
else if (root-> val < x){
 root-> right= delete (root->right,x)
}
```

```
//equal
else {
    if ( !root->left && !root->right)        ②
            return NULL;

⊙ else if ( root->left and root->right) //

⊙ else if ( root->left)
        root-> left;
⊙ else    root-> right;
}
    return root;
}
```

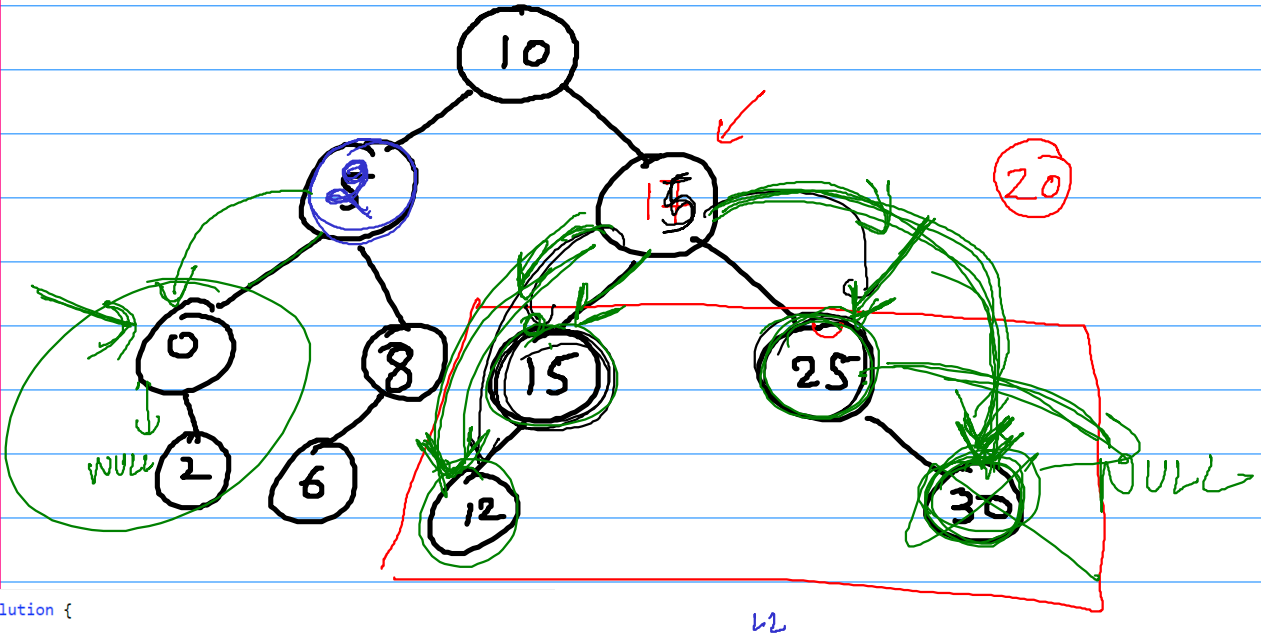
② Node having single child.



{5, 0, 15, 25}.

if(root→left

① Node having both the children.




```
class Solution {
public:
    int maxinLeft(TreeNode* root){
        while(root -> right){
            root = root -> right;
        }
        return root -> val;
    }
    TreeNode* deleteNode(TreeNode* root, int key) {
       if(!root)
           return NULL;
        if(key < root -> val){
            root -> left = deleteNode(root -> left, key);
        }else if(key > root -> val){
            root -> right = deleteNode(root -> right, key);
        }else{
            if(root -> left and root -> right){
                int val = maxinLeft(root -> left);
                root -> val = val;
                root -> left = deleteNode(root -> left, val);
                return root;
            }else if(root -> left){
                return root -> left;
            }else if(root -> right){
                return root -> right;
            }else{
                return nullptr;
            }
        }
        return root;
    }
```

root→right ≡ Rule

Time: O(H) ≈ (H × N)

Q. Validate BST:

Description | Solution | Discuss (999+) | Submissions

98. Validate Binary Search Tree

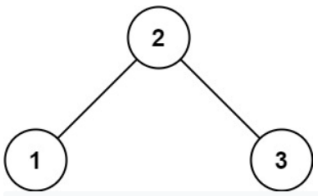
Medium 10154 903 Add to List Share

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:




```
Input: root = [2,1,3]
Output: true
```


P1 (min1, max1) root->val ;
P2 (min2, max2)

return (min(min1, min2, root->val),
max(max1, max2, root->val)};

```
12  class Solution {
13  public:
14      typedef long long ll;
15      bool ans;
16      pair<ll, ll> dfs(TreeNode* root){
17          if(!root){
18              return {LLONG_MIN, LLONG_MAX};
19          }
20          if(ans == false)return {-1, -1};
21          auto p1 = dfs(root -> left); //{max, min}
22          auto p2 = dfs(root -> right);
23          if(p1.first < root -> val and p2.second > root -> val){
24              return {max(p2.first, 1ll*root -> val), min(p1.second, 1ll*root ->val)};
25          }
26          ans = false;
27          return {-1, -1};
28
29      }
30      bool isValidBST(TreeNode* root) {
31          ans =  true;
32          dfs(root);
33          return ans;
34      }
35  };
```

O(N)

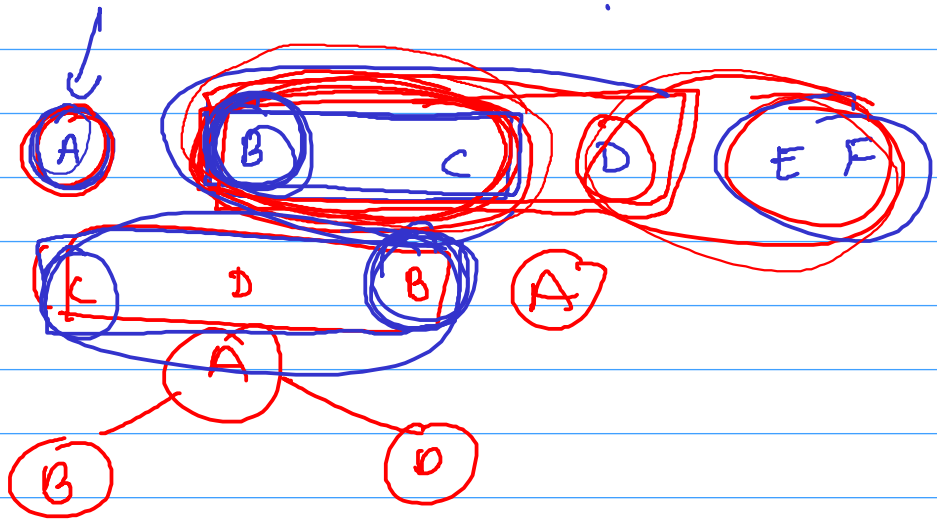
```cpp
class Solution {
public:
    typedef long long ll;

    bool isValidBST(TreeNode* root, ll mn = LLONG_MIN, ll mx = LLONG_MAX) {
        if(!root)return true;
        return (root -> val > mn and root -> val < mx and isValidBST(root -> left,mn, root -> val) and isValidBST(root ->  right, root -> val , mx));
    }
};
```

O(N)

⊙ Maxim

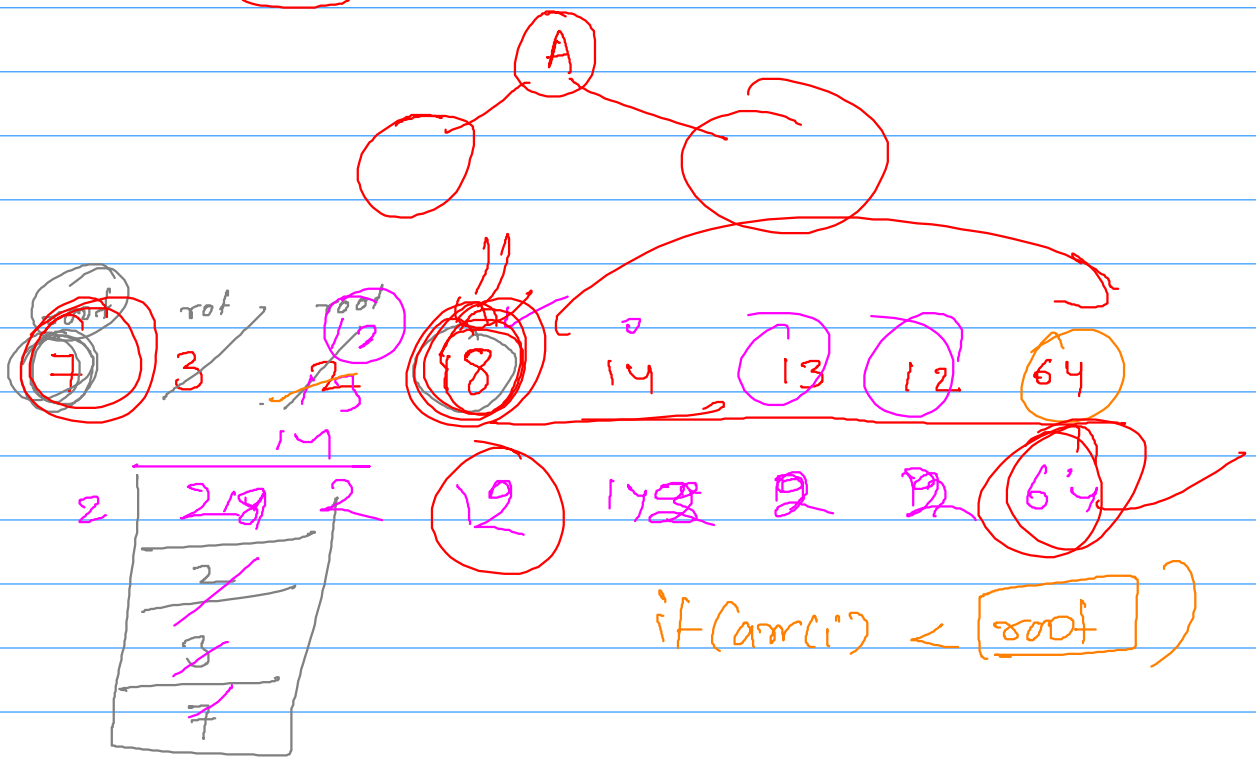
Preorder

Inorder.



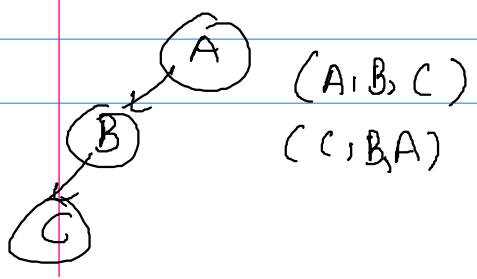
A B C

A B

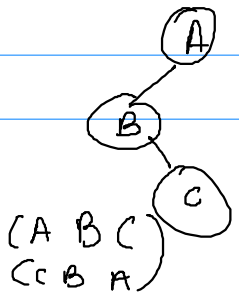


⊙ Create Tree from Given Traversals.

① Pre-order & Post-order.



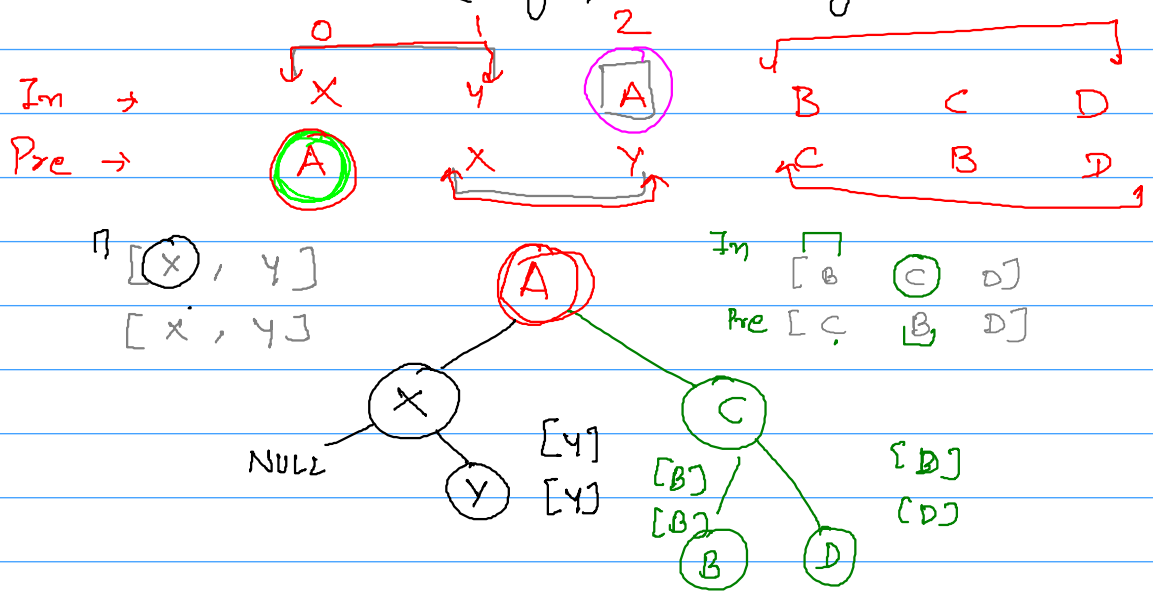
(A, B, C)
(C, B, A)



① Inorder & Preorder.

Pre order (root, left, right)

Inorder (left, root, right)



help(in, pre, instart, inend, pstart, pend) //O(N)

if (pstart > pend || instart > inend)

return NULL;

Tree *Node = new Node (pre(pstart));

int idx = findIn Inorder (in, pre[pstart]);

int in_left = idx - instart;

root->left : help(in, pre, instart, idx-1, pstart+1, pstart + in_left);

root->right = help (in, pre, idx+1, inend, pstart + inleft+1, pend) ;

};

O(N)

O(N²)

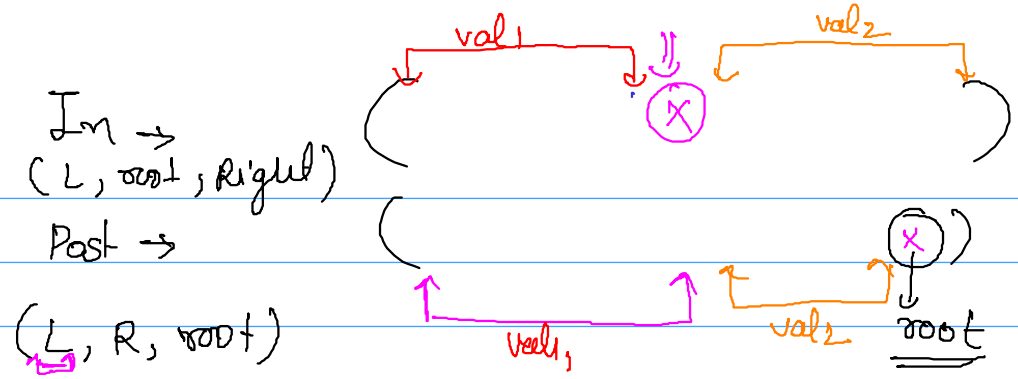
NlogN

N

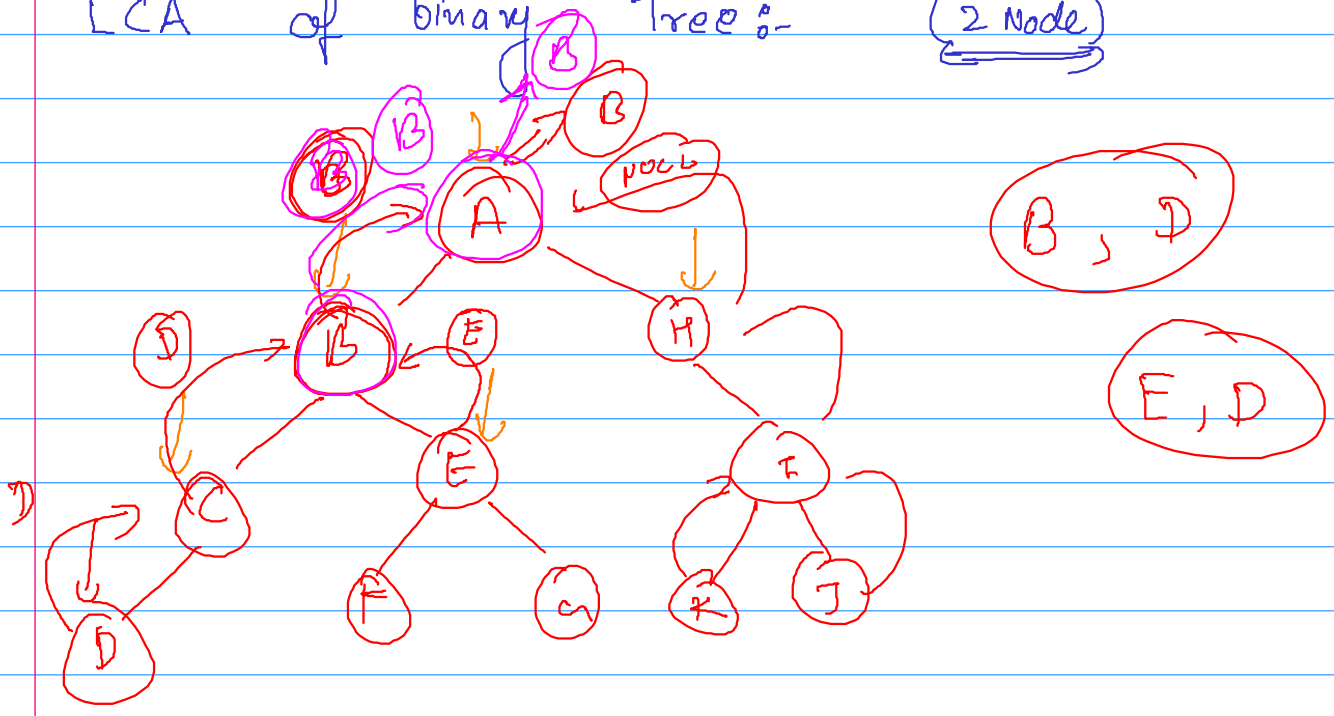
O(N)

logN

O(1)

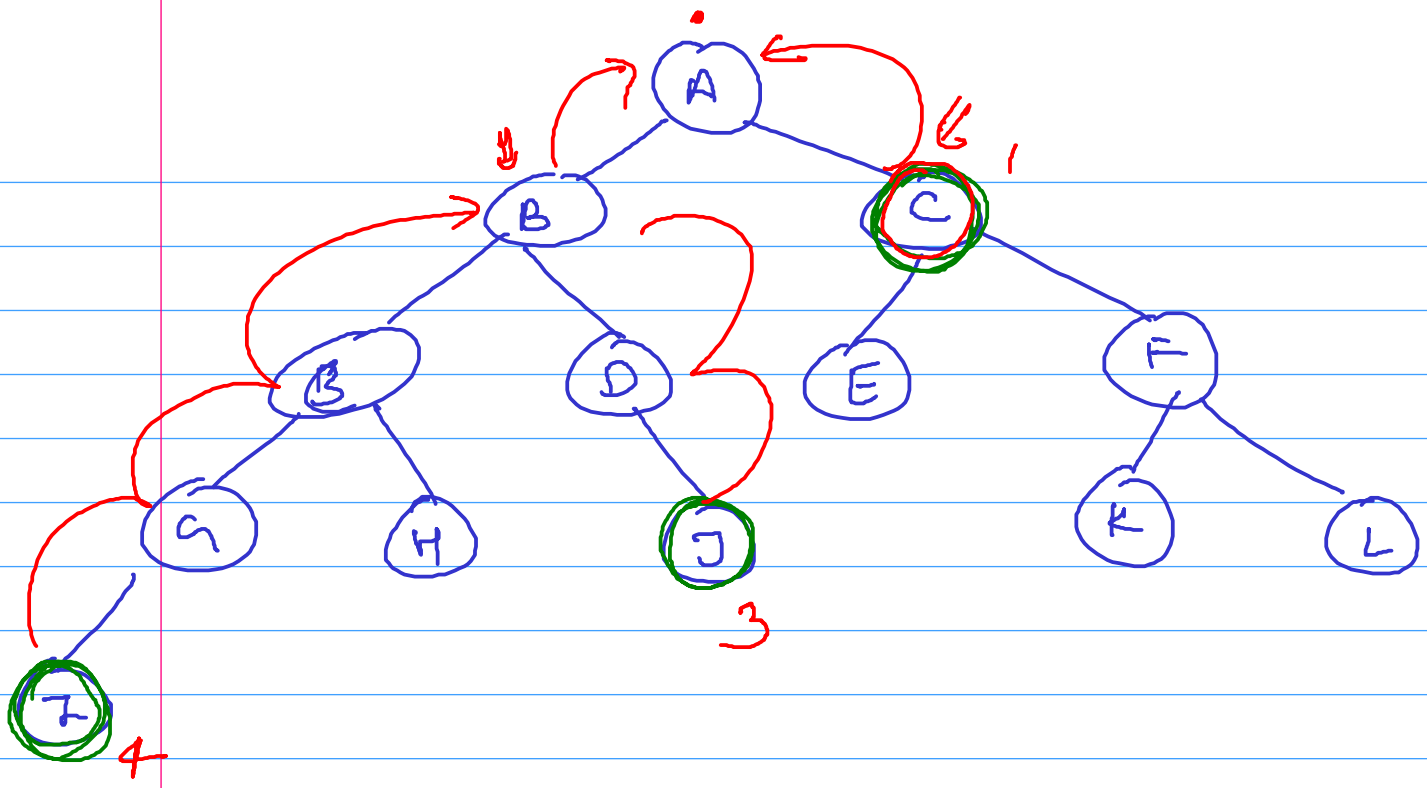


① LCA of binary Tree :- (2 Node)

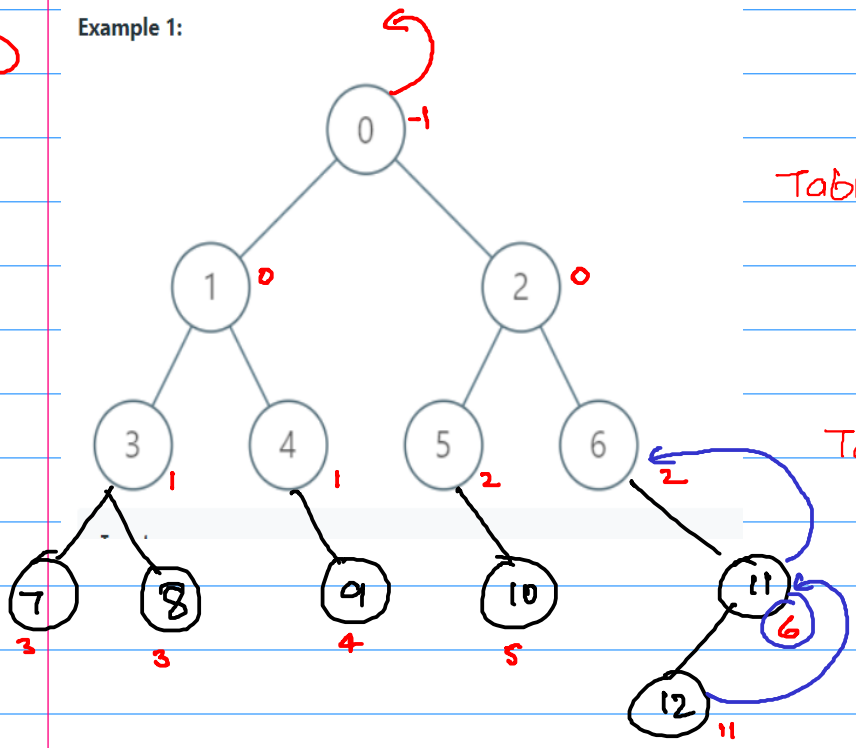



```
LCA ( r , v1 , v2 )
if(!r) ret NULL;
if( r->val == v1 || r->val == v2)
      retn r;
left  LCA(r->left, v1, v2)
right LCA (r->right, v1, v2)
if (left, right) retn r;
if(left) retn left,
retn right;
```

O(N)



Example 1:



Table[i][j] =

i^{th} 2^j

Table[i][0] = par[i];

Table[i][i] =

Table[i][

$2^1 = 2^0$

Table[i][i] = Tab [Tab[i][0]][0];

Table[i][j] = Table[table[i][j-1]][j-1];

2^j

2^2 = Table[Table[i][1]][1]

```cpp
class TreeAncestor {
public:
    vector<vector<int>>table;
    TreeAncestor(int n, vector<int>& parent) {
        table.resize(n, vector<int>(20));
        for(int i = 0; i < n ; i++){
            table[i][0] = parent[i];
        }

        for(int j = 1; j < 20 ; j++){
            for(int i = 0; i < n ; i++){
                if(table[i][j - 1] == -1)
                    table[i][j] = -1;
                else
                    table[i][j] = table[table[i][j - 1]][j - 1];
            }
        }
    }

    int getKthAncestor(int node, int k) {

        for(int i = 19 ; i >= 0 ; i--){
            if((k >> i)&1){
                node = table[node][i];
                if(node == -1)
                    return node;
            }
        }
        return node;
    }
};
```

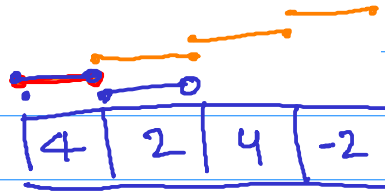
O(N log N)

O(log N)

table[i][0] = Arr[i];

table[i][j] =

min (table[i][j-1], table[i + 2^(j-1)][j-1])



table[j][j]

i + 2^0

0 + 1 =

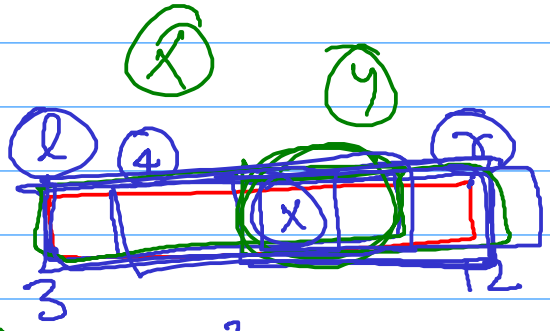
starting → i consider len 2^j

what is min of this range.

gcd

(2, 2, 3)

(2, 3)



12 + 1 − 3 = 10

f(a, b, c,) = a^(2^3)

(3)(8)

8 + 3 = 11 ⇒ 8

f(a, b, b, c, c) = a

(3)(3), ((3 + 2^3), (2))

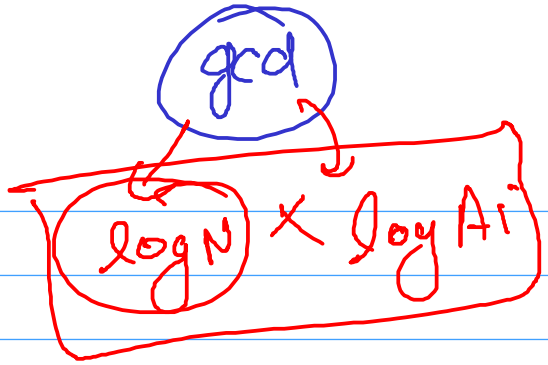
i[2^x]

[r − 2^x][x]

2, 3, 4 = 2

(2, 2, 3, 3, 4, 4) = 2

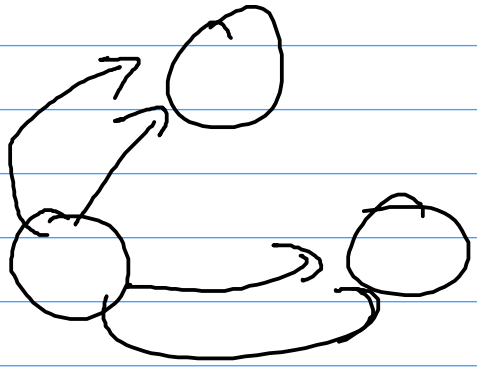
Updation Segnet



Sparse table

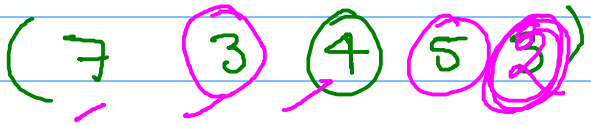
O(1)

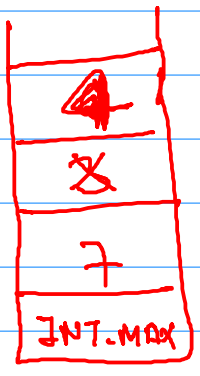
O(logA[])



7 1 4 2 9

7 1 4

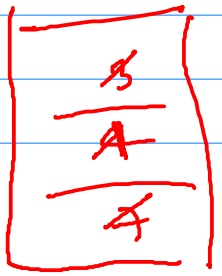


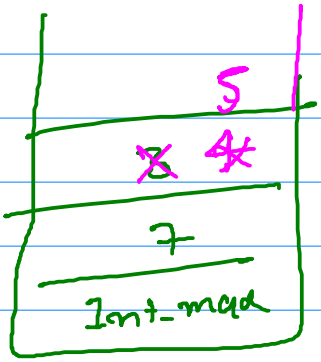


Mn=3

Mn=4

10





Mn=3

Mn=4

7 1 4 3 10 6

Heap.

0 Get Min / Max O(1)

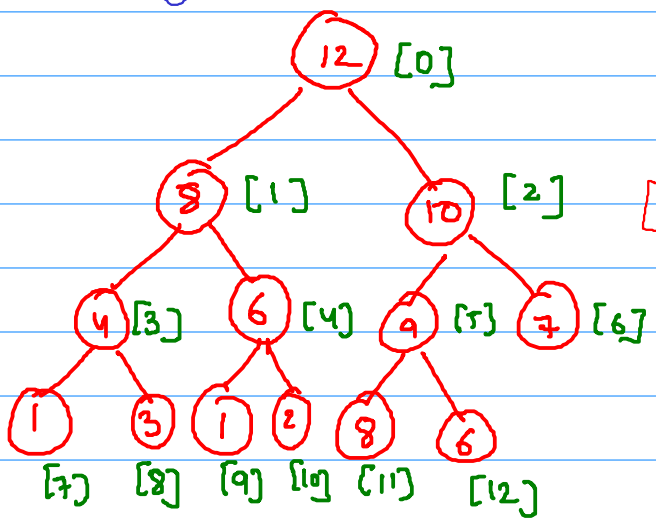
0 Remove Min / Max O(log N) N: is no. of elemet in Heap.

0 Insert a element O(logN).

0 Trees → Binary Trees → Complete binary Trees.

Binary Heap. {MAX}.

root Node value will be maxim. in its sub trees. every node will follow this property.



Complete Tree.
{Array notation}.

[valid Heap].

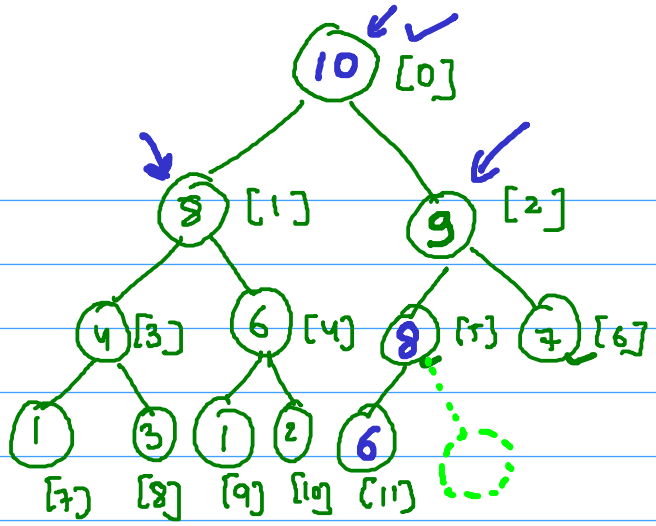
idx → L.C: 2*idx+1

R.C. 2*idx+2

idx → parent: $\frac{(idx-1)}{2}$.

1 Get ~~Min~~ MAX. O(1) heap[0] // will be max. demod.

0 Remove MAX Complete Tree $\lceil \log_2 N \rceil$

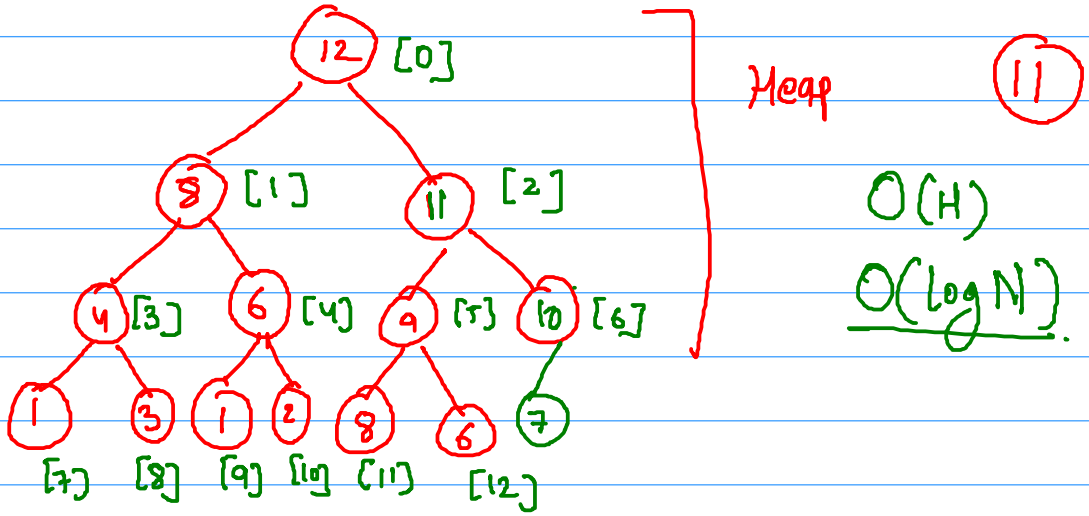


Heap ✓

O(H)

O(log N)

① Insert an Element.



Heap

11

O(H)

O(log N).

```
18  class HEAP{
19    int maxsz, *heap, n;
20  public:
21    HEAP(int cap){
22      n = 0;
23      maxsz = cap;
24      heap = new int[cap];
25    }
26    int getMax(){
27      if(n == 0)return -1;
28      return heap[0];
29    }
30
31    void downHeapify(int i){
32      int idx = i;
33      if(2*i + 1 < n and heap[idx] < heap[2*i + 1]){
34        idx = 2 * i + 1;
35      }
36      if(2 * i + 2 < n and heap[idx] < heap[2 * i + 2]){
37        idx = 2 * i + 2;
38      }
39      if(idx != i){
40        swap( & heap[idx] , & heap[i]);
41        downHeapify( i idx);
```

// O(1)

// O(log N)

```
44    void removeMax(){
45      if(n == 0){
46        return ;
47      }
48      swap( & heap[0], & heap[n-1]);
49      n--;
50      downHeapify( i 0);
51    }
52
53    void insert(int x){
54      if(n == maxsz){
55        return ;
56      }
57      heap[n] = x;
58      n++;
59      int idx = n - 1;
60      while (idx > 0 and heap[(idx - 1) / 2] < heap[idx]){
61        int p = (idx - 1) / 2;
62        swap( & heap[p], & heap[idx]);
63        idx = p;
64      }
65    }
```

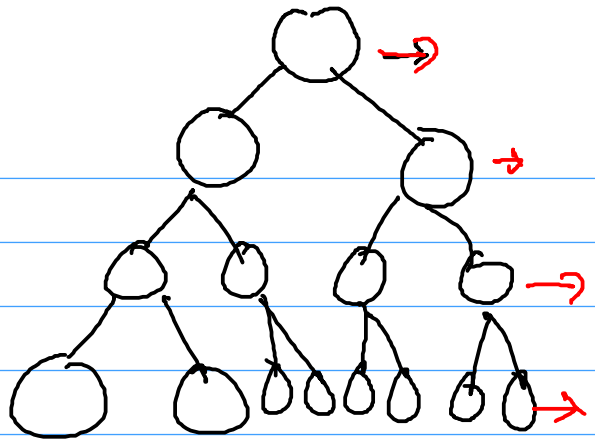
// O(log N)

solve

N

Build Heap O(N log N)

O(N)



log 1

2 x 2

4 x 3

8 x 4

$O(2N) = O(N)$

N. log N

GFG

= x = x = x = x = x =

- Heap Sort :-

  ↳ selection sort with finding Min in log N.

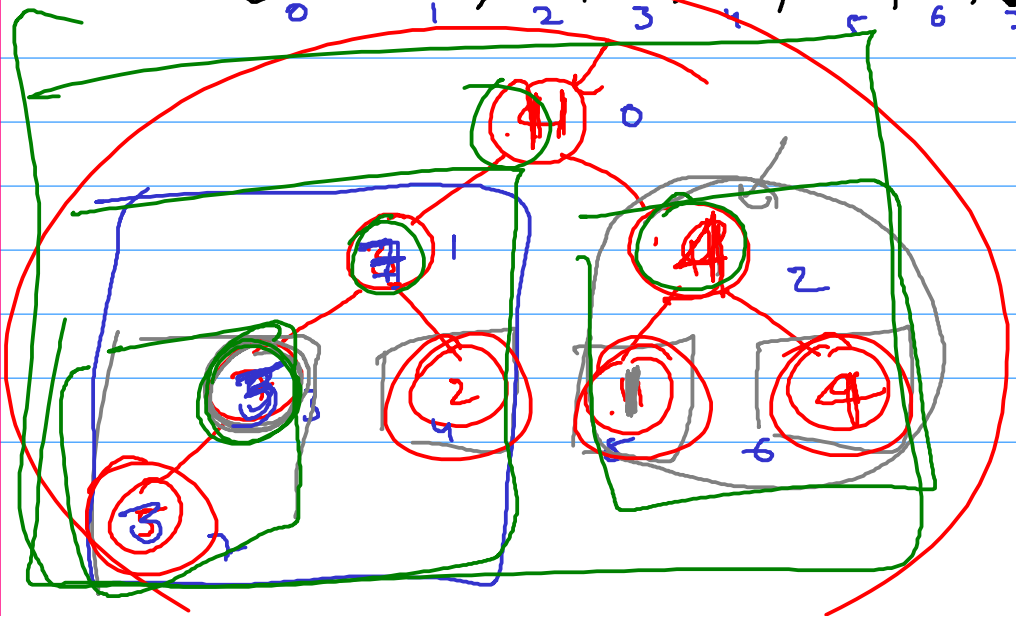
$O(N)$

$N \cdot N$ N^2

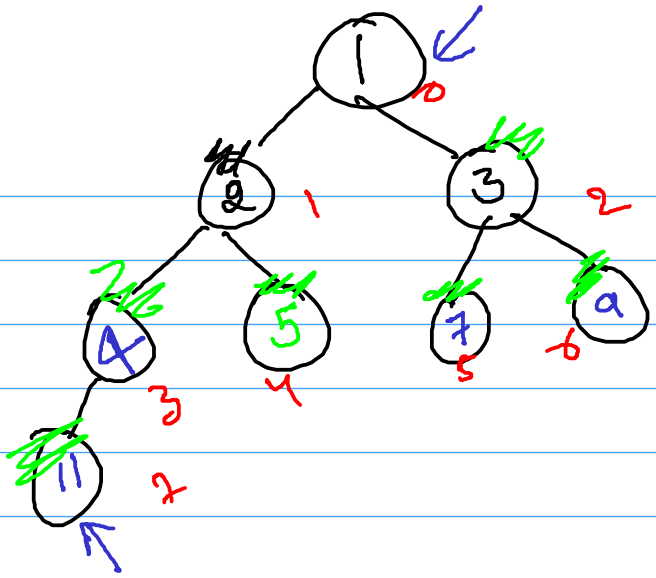
N log N

[4, 3, 1, 7, 2, 11, 9, 5]

0 1 2 3 4 5 6 7



$\frac{(7-1)}{2} = 3$



[1, 2, 3, 4, 5, 7, 9, 11]

```
22
23
24  void heapify(int heap[], int n, int i) {
25      // printf("%d\n", i);
26      // Find largest among root, left child and right child
27      int idx = i;
28      if(2*i + 1 < n && heap[idx] < heap[2*i + 1]){
29        idx = 2 * i + 1;
30      }
31      if(2 * i + 2 < n && heap[idx] < heap[2 * i + 2]){
32        idx = 2 * i + 2;
33      }
34      if(idx != i){
35        swap(&heap[idx] , &heap[i]);
36        heapify(heap, n, idx);
37      }
38
39  }
40
41  // Main function to do heap sort
42  void buildHeap(int arr[], int n) {
43      int idx = n - 1;
44      int p = (idx - 1) / 2;
45      for(int i = ((n - 1) - 1) / 2; i >= 0 ; i--){
46          heapify(arr, n, i);
47      }
48      for(int i = 0; i < n ; i++){
49          swap(&arr[0], &arr[n - 1 - i]);
50          heapify(arr, n - 1 - i, 0);
51      }
52  }
53      } // } Driver Code Ends
```

Time: O(NlogN)

Space: O(1)

0 Median of Data stream :-

295. Find Median from Data Stream

Hard 7140 131 Add to List Share

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value and the median is the mean of the two middle values.

- For example, for `arr = [2,3,4]`, the median is `3`.
- For example, for `arr = [2,3]`, the median is `(2 + 3) / 2 = 2.5`.

Implement the MedianFinder class:

- `MedianFinder()` initializes the `MedianFinder` object.
- `void addNum(int num)` adds the integer `num` from the data stream to the data structure.
- `double findMedian()` returns the median of all elements so far. Answers within 10^{-5} of the actual answer will be accepted.

Example 1:

```
Input
["MedianFinder", "addNum", "addNum", "findMedian", "addNum",
"findMedian"]
[[], [1], [2], [], [3], []]
Output
[null, null, null, 1.5, null, 2.0]

Explanation
```

Brute force: vec

vec.push_back();

sort(vec.begin() vec.end())

if (n&1)

vec.[vec.size()/2];

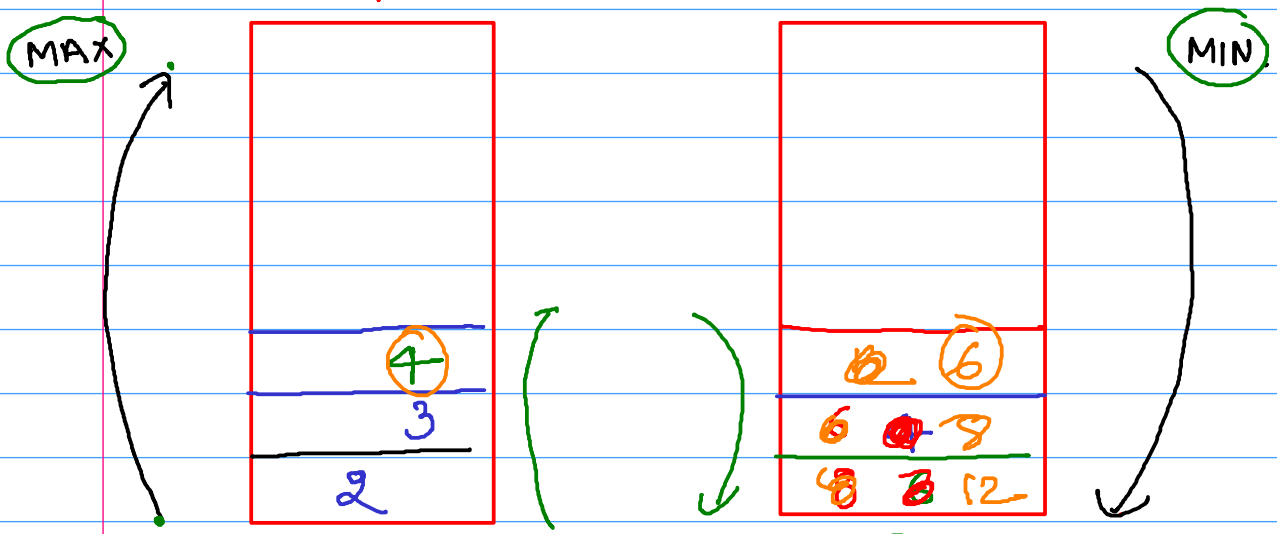
MAX

D_1.size() == D_2.size() == D_3.size() == D_4.size()

4, 6, 2, 3, 8, 12

4, 5, 4, 3.5, 4, 10

MAX < MIN



D_1

median = D_1 (MAX + MIN) / 2.0

- Sliding window maximum :-
- Sliding window Median.

480. Sliding Window Median

Hard 2221 135 Add to List Share

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle values.

- For examples, if `arr = [2,3,4]`, the median is `3`.
- For examples, if `arr = [1,2,3,4]`, the median is `(2 + 3) / 2 = 2.5`.

You are given an integer array `nums` and an integer `k`. There is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position.

Return *the median array for each window in the original array*. Answers within 10^{-5} of the actual value will be accepted.

Example 1:

```
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [1.00000,-1.00000,-1.00000,3.00000,5.00000,6.00000]
Explanation:
Window position                Median
---------------                -----
[1  3  -1] -3  5  3  6  7       1
 1 [3  -1  -3] 5  3  6  7      -1
 1  3 [-1  -3  5] 3  6  7      -1
 1  3  -1 [-3  5  3] 6  7       3
 1  3  -1  -3 [5  3  6] 7       5
 1  3  -1  -3  5 [3  6  7]      6
```

a_0 a_1 a_2 a_3 a_4

+1, -1 coeff = 1

arr(i) = median

a_0 a_1 a_2 a_3

$\sum |a_i - x|$ minimum

$\sum (a_i - x)^2$ Min :

x = median of array

$x = \frac{\sum a_i}{n}$

$a_i - x$

$$f(x) = (a_1 - x)^2 + (a_2 - x)^2 + (a_3 - x)^2 + \cdots$$

$$= x^2 - 2a_1x + a_1^2 + a_2^2 + x^2 - 2a_1x + a_3^2 - 2xa_3 + x^2$$

$$f(x) = nx^2 - 2x(a_1 + a_2 + a_3 + \cdots a_n) + \sum a_i^2$$

$$f'(x) = 2nx - 2(\text{Sum}) + 0 \to 0$$

$$2nx = 2(\text{Sum}) = 0$$

$$nx = \text{Sum}$$

$$x = \left[\frac{\text{Sum}}{n}\right]$$

$f''(x) = 2n > 0$

(pt of Minima)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. Your task is to calculate for each window of k elements, from left to right, the minimum total cost of making all elements equal.

You can increase or decrease each element with cost x where x is the difference between the new and the original value. The total cost is the sum of such costs.

Input

The first input line contains two integers n and k: the number of elements and the size of the window.

Then there are n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Output $n - k + 1$ values: the costs.

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:
```
8 3
2 4 3 5 8 1 2 1
```

Output:
```
2 2 5 7 7 1
```

3

MAX

Set

MIN

Set

O(1)

Example 1:

```
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [1.00000,-1.00000,-1.00000,3.00000,5.00000,6.00000]
Explanation:
Window position                Median
---------------                -----
[1  3  -1] -3  5  3  6  7       1
 1 [3  -1  -3] 5  3  6  7      -1
 1  3 [-1  -3  5] 3  6  7      -1
 1  3  -1 [-3  5  3] 6  7       3
 1  3  -1  -3 [5  3  6] 7       5
 1  3  -1  -3  5 [3  6  7]      6
```

MAX - MIN ≤ 1

Example 2:

```cpp
class Solution {
public:
    set<pair<double, int>> MIN;
    set<pair<double, int>, greater<>> MAX;
    void balance(){
        if((int)MAX.size() - (int)MIN.size() == 2){
            auto pr = *MAX.begin();
            MAX.erase(MAX.begin());
            MIN.insert(pr);
        }
        if(!MIN.empty() and MIN.begin() -> first < MAX.begin() -
            auto pr1 = *MAX.begin();
            auto pr2 = *MIN.begin();
            MAX.erase(MAX.begin());
            MIN.erase(MIN.begin());
            MAX.insert(pr2);
            MIN.insert(pr1);
        }
    }
    void insert(double val, int idx){
        MAX.insert({val, idx});
        balance();
    }
    void erase(double val, int idx){
        if(MAX.count({val, idx})){
            MAX.erase({val, idx});
            if(MAX.size() < MIN.size()){
                auto pr2 = *MIN.begin();
                MIN.erase(MIN.begin());
                MAX.insert(pr2);
            }
        }else{
            MIN.erase({val, idx});
            balance();
        }
    }
    double med(int o){
        double med = MAX.begin() -> first;
        if( o == false )
            med = (med + MIN.begin() -> first) / 2.0;
        return med;
    }
    vector<double> medianSlidingWindow(vector<int>& nums, int k) {

        for(int i = 0; i < k ; i++){
            insert(nums[i]*1.0, i);
        }
        vector<double> ans;
        int o = (k&1);
        ans.push_back(med(o));
        for(int i = k; i < nums.size(); i++){
            erase(nums[i - k], i - k);
            insert(nums[i], i);
            ans.push_back(med(o));
        }
        return ans;
    }
};
```

M

$|a_i - x|$ min x = med.

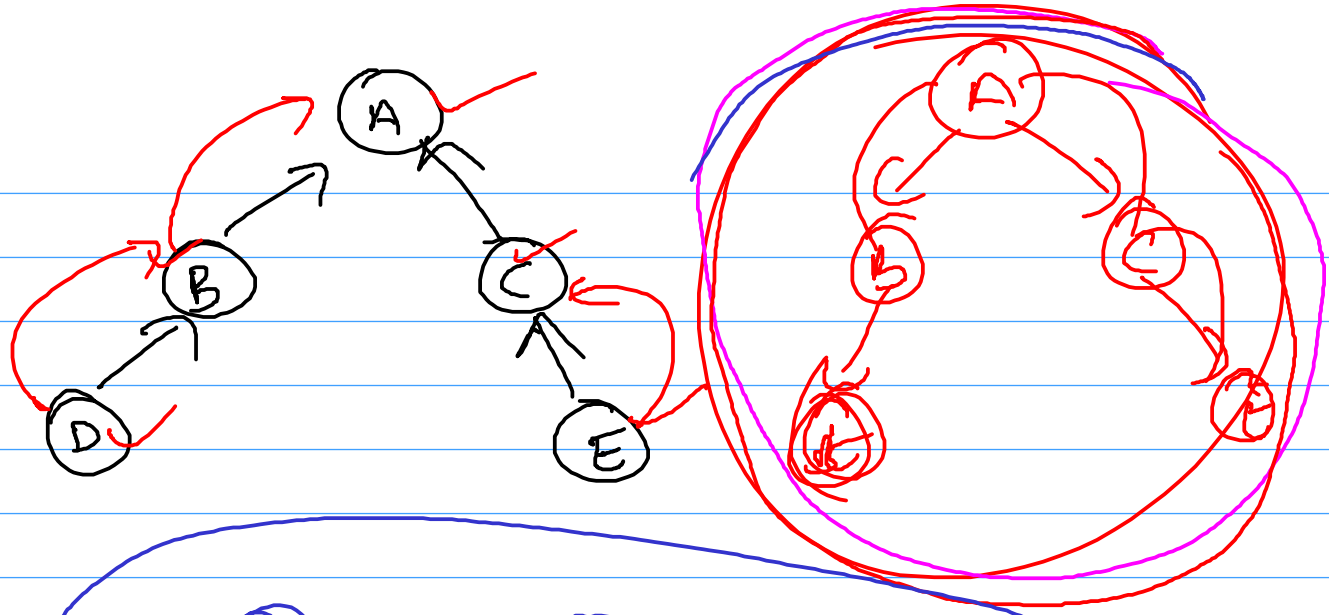
$|a_0 - x| + |a_1 - x| + |a_2 - x| + \dots |a_{n-1} - x|$

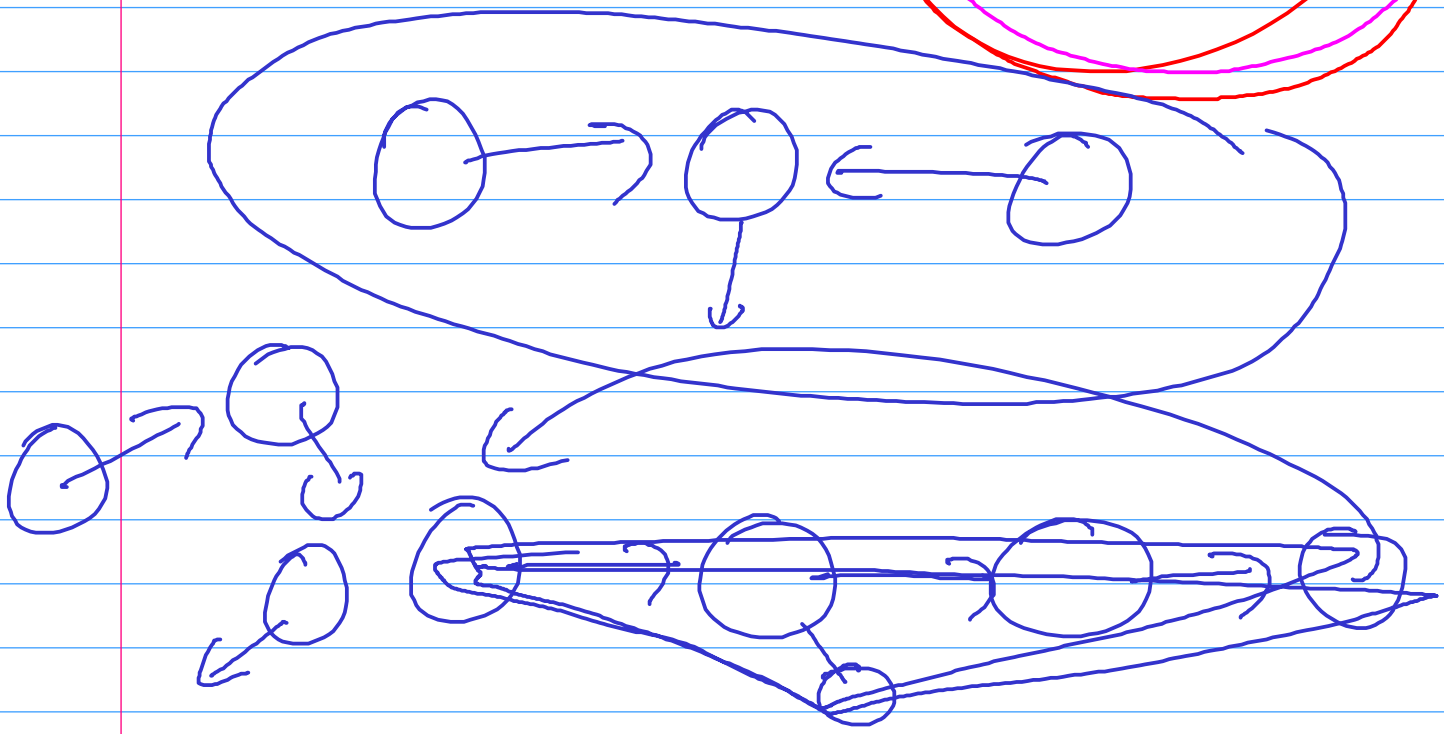
for all $a_i \le x$ for all value $a_j > x$

$\sum (x - a_i)$ $\sum (a_i - x)$

MAX.begin()->first = med. Sum: Min of heap

(med * No. of ele in Max heap) − (Sum of Max) − x.size of MinHp





$q_3 < q_0 + q_1$

$q_0 + q_1$ — ✗

q_0

$x < 2x$

↑ ↑

$q_0 + q_1$

for (int i=0; i<n; i++)

int fix = a(i);

fix + a(i)

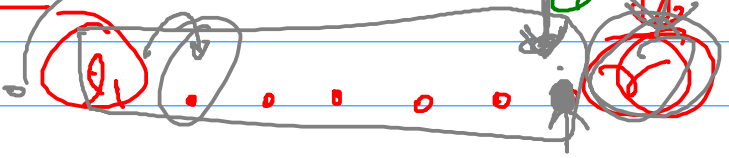
fix+a(P1) < P2 int P1 = i+1, P2 = i+2;

while (P1

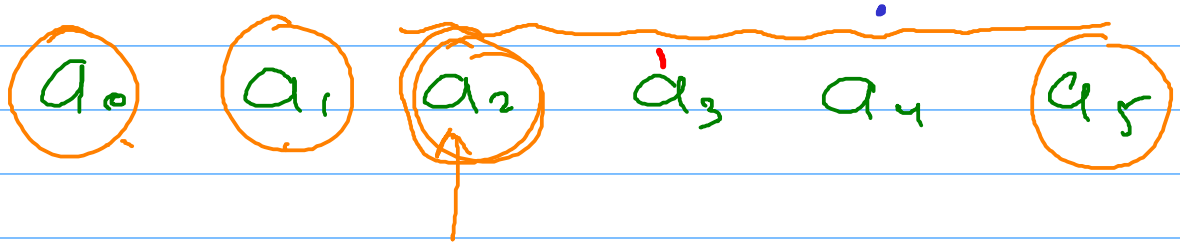
if (fix + a(P1) < a(P2))

a(P1) + a(i) > a(r) P1++.

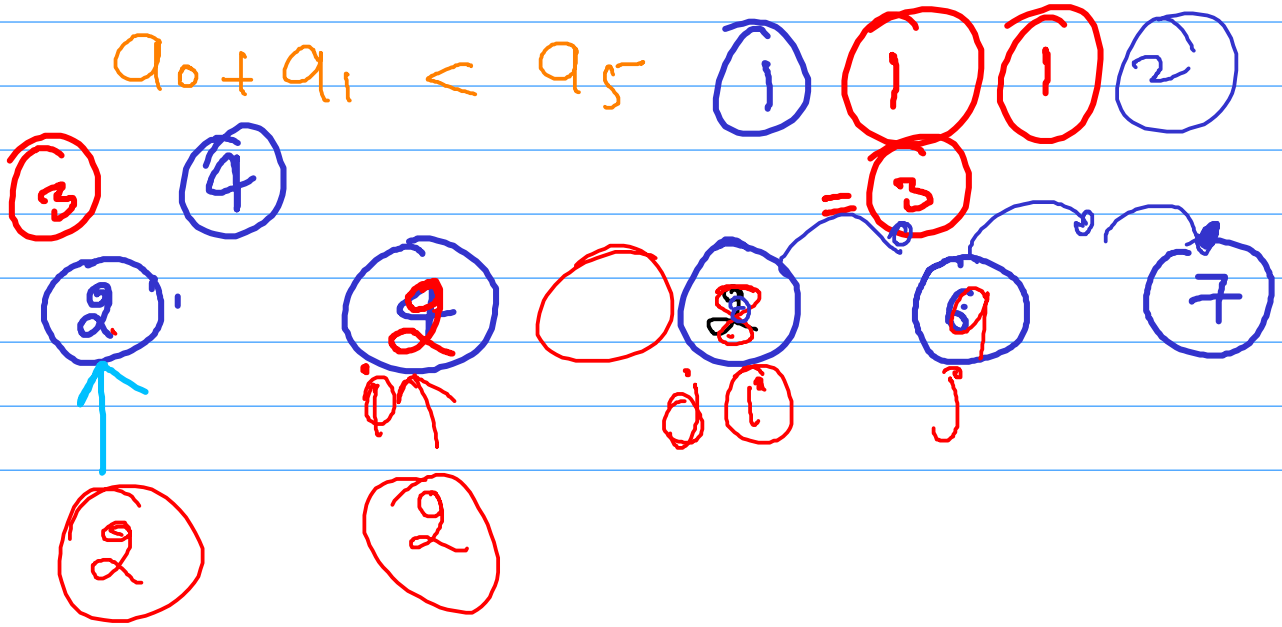
ans += (r - l1)



a0 + a1 −1 a2 + a(i)



a0 + a1 < a5



[4, 6, 13, 16, 20, 3, 1, 12]