LEARN DSA WITH C++

WEEK :: 11

LEARN DSA
WITH C++

PDF

COURSE INSTRUCTOR BY
ROHIT NEGI

Check Profile

MADE BY-
PRADUM SINGHA

My profile

LEARN DSA WITH C++

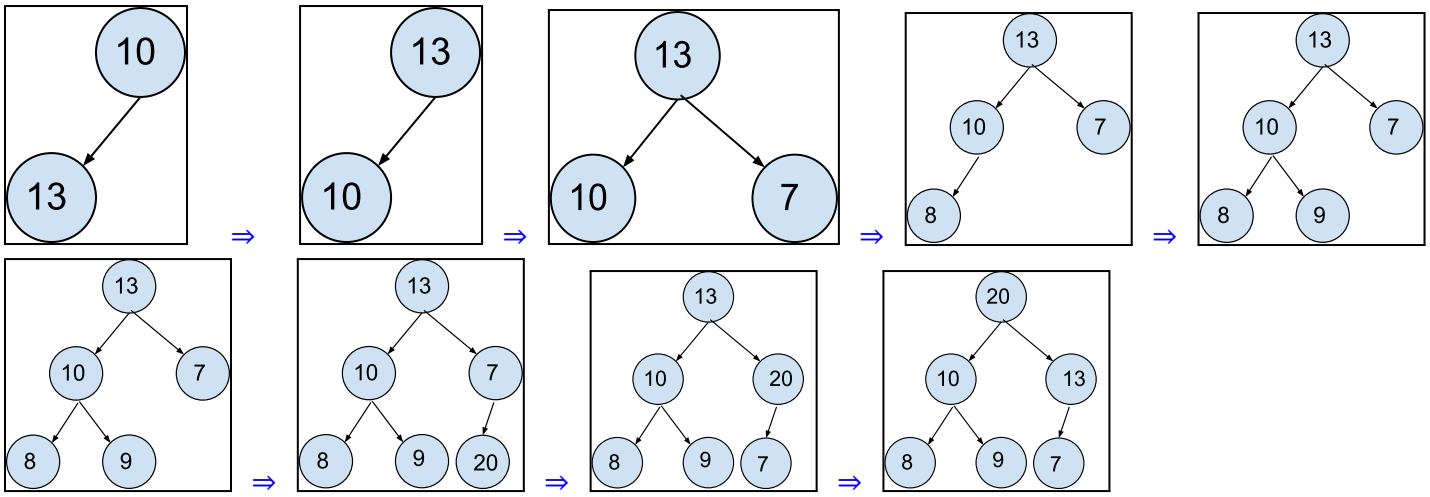
WEEK :: 11 **DAY: 01** **DATE: 28-06-2023**

HEAP BASIC

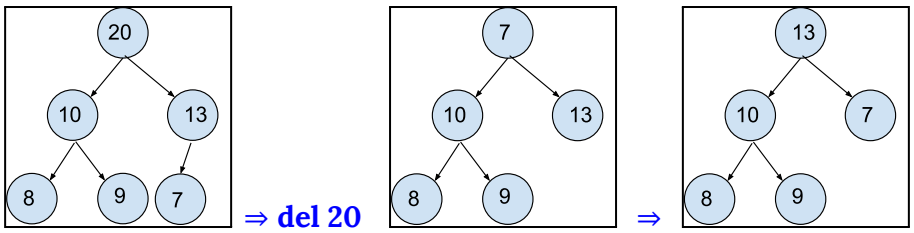
A Heap is a special Tree-based data structure in which the tree is a complete binary tree.
There are more priority nodes to execute first.

How to Create HEAP:-

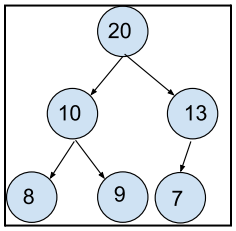
Input :	**1**	**2**	**3**	**4**	**5**	**6**
Emergency :	**10**	**13**	**7**	**8**	**9**	**20**



Delete Operation :-



Insertion Operation :-

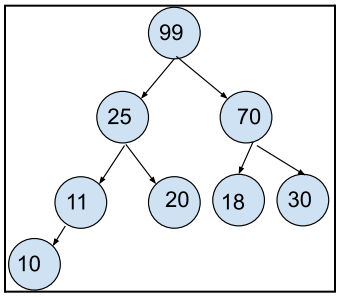


Convert Array : {20, 10, 8, 9, 13, 7};
i = parent node
Left Child = 2 * i + 1;
Right Child =2 * i + 2;
Find Parent Node = (i+1)/2 i=child Node

Create HEAP using Array:-

{30, 10, 70, 20, 25, 18, 9, 11}

Inserte elem ⇒ Parent Node >= Child Node



Index =	0	1	2	3	4	5	6	7
Array =	99	25	70	10	20	18	30	10

Time Complexity :- N(logN)

Insertion + Height;

LEARN DSA WITH C++

WEEK :: 11 **DAY: 02** **DATE: 29-06-2023**

HEAP

Priority Queue

Heap

Max - Heap Min Heap

Create HEAP using Array:-

Input : 30 20 ; Array:- 30 20 NO change child =i; Parent =(i-1)/2
Idx : 0 1 Idx : 0 1
Input : 40 ; Array:- 30 20 40 => 40 20 30 {compare between child(40) and parent(30)}
Idx : 0 1 2 Idx : 0 1 2
Input : 15; Array:- 40 20 30 15 => Array:- 40 20 30 15
Idx : 0 1 2 3 Idx : 0 1 2 3
Input : 32; Array:- 40 20 30 15 32 => Array:- 40 32 30 15 20
Idx : 0 1 2 3 4 Idx : 0 1 2 3 4
Input : 70; Array:- 40 32 30 15 20 70 => Array:- 40 32 70 15 20 30 =>Array:- 70 32 40 15 20 30
Idx : 0 1 2 3 4 5 Idx : 0 1 2 3 4 5 Idx : 0 1 2 3 4 5
Input : 25; Ary:- 70 32 40 15 20 30 25 =>Ary:- 70 32 40 15 20 30 25
Idx : 0 1 2 3 4 5 6 Idx : 0 1 2 3 4 5 6
Input : 18; Ary:- 70 32 40 15 20 30 25 18 =>Ary:- 70 32 40 25 20 30 25 15
Idx : 0 1 2 3 4 5 6 7 Idx : 0 1 2 3 4 5 6 7

Ary:- 70 32 40 25 20 30 25 15
Idx : 0 1 2 3 4 5 6 7

Create HEAP by Code :-

```cpp
#include<iostream>
#include<vector>
using namespace std;

void insertHeap(vector<int> &maxheap)
{
    int index = maxheap.size() -1;
    int parent;
    while(index >0)
    {
        parent = (index -1)/2;
        // parent is small
        if(maxheap[parent]< maxheap[index])
        {
```

```
            swap(maxheap[parent], maxheap[index]);
            index = parent;
        }
        // parent is big
        else
        break;
    }
}
int main()
{
    vector<int>maxHeap;
    int n, element;

    // size of heap
    cin>>n;
    for(int i=0; i<n; i++)
    {
        cin>>element;
        maxHeap.push_back(element);
        insertHeap(maxHeap);
    };

    for(int i=0; i<maxHeap.size(); i++)
    cout<<maxHeap[i]<<" ";

return 0;
};
```

Time Complexity :-	**O(nlogn)**
Space Complexity :-	**O(1)**

Delete Operation Code :-

```
#include<iostream>
#include<vector>
using namespace std;

void insertHeap(vector<int> &maxheap)
{
    int index = maxheap.size() -1;
    int parent;
    while(index >0)
    {
        parent = (index -1)/2;
        // parent is small
        if(maxheap[parent]< maxheap[index])
        {
            swap(maxheap[parent], maxheap[index]);
```

```
            index = parent;
        }
        // parent is big
        else
        break;
    }
};

void Heapify(vector<int> &maxHeap, int index)
{
    int largest = index;
    int left = 2*index +1;
    int right = 2*index +2;
    int size = maxHeap.size();
    // check for left side
    if(left<size && maxHeap[left]>maxHeap[largest])
    largest = left;
    // check for Right side
    if(right<size && maxHeap[right]>maxHeap[largest])
    largest = right;

    // need to swap
    if(largest != index)
    {
        swap(maxHeap[largest], maxHeap[index]);
        Heapify(maxHeap, largest);
    }
    return;

};

void DeleteHeap(vector<int> &maxHeap)
{
    // Replace First element by last element
    maxHeap[0] = maxHeap[maxHeap.size()-1];

    // Delete the last element
    maxHeap.pop_back();

    //set correct position
    Heapify(maxHeap, 0);
};

int main()
{
    vector<int>maxHeap;
    int n, element;

    // size of heap
    cin>>n;
    for(int i=0; i<n; i++)
    {
        cin>>element;
        maxHeap.push_back(element);
```

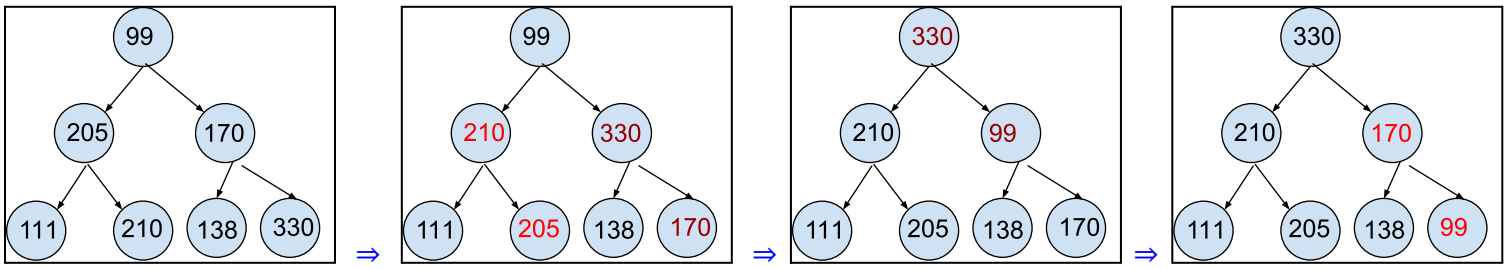
```
        insertHeap(maxHeap);
    };
    DeleteHeap(maxHeap);
    DeleteHeap(maxHeap);


    for(int i=0; i<maxHeap.size(); i++)
    cout<<maxHeap[i]<<" ";

return 0;
};
```

Delete Operation :-	
Time Complexity	O(1) + O(logN) ⇒ O(longN)
Space Complexity	O(longN)

Heap Create within O(N) Time :-



n/4*1 + n/8*2 + n/16*3 + n/32*4 + ...
N[¼ + ⅛ + 1/16 + 1/32 + ...]
N(1) = O(N)---> Time complexity

```
#include<iostream>
#include<vector>
using namespace std;

void Heapify(vector<int> &maxHeap, int index)
{
    int largest = index;
    int left = 2*index +1;
    int right = 2*index +2;
    int size = maxHeap.size();
    // check for left side
    if(left<size && maxHeap[left]>maxHeap[largest])
    largest = left;
    // check for Right side
    if(right<size && maxHeap[right]>maxHeap[largest])
    largest = right;

    // need to swap
```

```
    if(largest != index)
    {
        swap(maxHeap[largest], maxHeap[index]);
        Heapify(maxHeap, largest);
    }
    return;

};
int main()
{
    vector<int>maxHeap;
    int n, element;
    cin>>n;
    for(int i=0; i<n; i++)
    {
        cin>>element;
        maxHeap.push_back(element);
    }
    for(int i=n/2-1; i>=0; i--)
    {
        Heapify(maxHeap, i);
    };

    for(int i=0; i<maxHeap.size(); i++)
    cout<<maxHeap[i]<<" ";

return 0;
};
```

LEARN DSA WITH C++

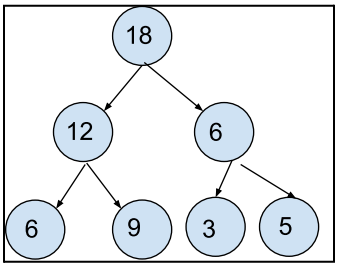
WEEK :: 11 **DAY**: 03 **DATE**: 06-07-2023

HEAP USING SORTING

Shoring :-

Array

18	12	6	5	9	3	5
Idx : 0	1	2	3	5	4	6



Exchange 18 & 5, First and last

5	12	6	5	9	3	18
0	1	2	3	5	4	6

Compare index ⇒ 0 & 1; 1 & 5

12	9	6	5	5	3	18
0	1	2	3	5	4	6

Exchange 12 & 3 , First and last

3	9	6	5	5	12	18
0	1	2	3	5	4	6

Compare index ⇒ 0 & 1; 1 & 3

9	5	6	3	5	12	18
0	1	2	3	5	4	6

Exchange 9 & 5 , First and last

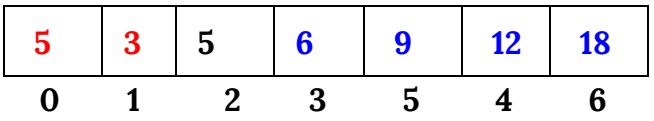
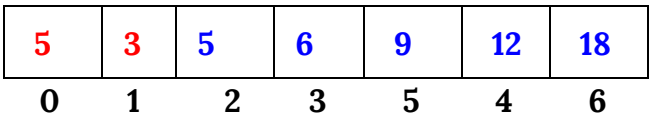
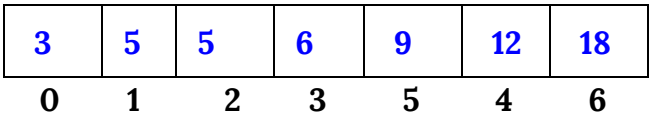
5	5	6	3	9	12	18
0	1	2	3	5	4	6

Compare index ⇒ 0 & 2

6	5	5	3	9	12	18
0	1	2	3	5	4	6

Exchange 6 & 3 , First and last

3	5	5	6	9	12	18
0	1	2	3	5	4	6

Compare index ⇒ 0 & 1	
Exchange 5 & 5 , First and last	
Compare index ⇒ 0 & 1	**No Change**
Exchange 5 & 3 , First and last	

Step for Shorting :-

1. **Swap the first element with the last element.**
2. **Decrease the size of the array by 1.**
3. **Index HeapiFy (then repeat)**

Time Complexity :- N logN

Heap Sort :- **<<GeeksforGeeks>>**

```
class Solution
    {
    public:
        // Heapify function to maintain heap property.
        void heapify(int maxHeap[], int size, int index)
        {
            // Your Code Here
            int largest = index;
            int left = 2 * index + 1;
            int right = 2 * index + 2;
            // check for left side
            if (left < size && maxHeap[left] > maxHeap[largest])
                largest = left;
            // check for Right side
            if (right < size && maxHeap[right] > maxHeap[largest])
                largest = right;

            // need to swap
            if (largest != index)
            {
                swap(maxHeap[largest], maxHeap[index]);
                heapify(maxHeap, size, largest);
            }
        }

    public:
        // Function to build a Heap from array.
```

```
        void buildHeap(int arr[], int n)
        {
            // Your Code Here
        }

    public:
        // Function to sort an array using Heap Sort.
        void heapSort(int arr[], int n)
        {
            // code here
            //  First create max heap
            for (int i = n / 2 - 1; i >= 0; i--)
            {
                heapify(arr, n, i);
            };
            // Heap sort

            for (int i =n - 1; i >= 0; i--)
            {
                swap(arr[i], arr[0]);
                heapify(arr, i, 0);
            }
        }
    };
```

Create Heap Using STL :- Priority Queue

Max - Heap

```
#include<iostream>
#include<queue>
using namespace std;

int main()
{
    // max heap
    priority_queue<int>p;
    // push insert
    p.push(10);
    p.push(19);
    p.push(15);
    p.push(13);
    //Size of Heap
    cout<<p.size()<<endl;
    cout<<p.top()<<endl;
    //pop delete
    p.pop();
    cout<<p.top()<<endl;
    //Is heap Empty
    cout<<p.empty()<<endl;

return 0;
};
```

Min Heap

```
#include<iostream>
#include<queue>
#include<vector>
using namespace std;
int main()
{
    // min heap
    priority_queue<int, vector<int>,
greater<int>>p;
    // push insert
    p.push(10);
    p.push(19);
    p.push(15);
    p.push(13);
    //Size of Heap
    cout<<p.size()<<endl;
    cout<<p.top()<<endl;
    //pop delete
    p.pop();
    cout<<p.top()<<endl;
    //Is heap Empty
    cout<<p.empty()<<endl;
return 0;
};
```

Profit Maximisation << InterviewBit >>

```
int Solution::solve(vector<int> &A, int B) {


    if(B==0)
    return 0;
    int sum=0;
    priority_queue<int>p;
    for(int i=0; i<A.size(); i++)
    {
        if(A[i])
        p.push(A[i]);
    }
    while(B && p.size())
    {
        sum += p.top();
        if(p.top() >1)
        p.push(p.top()-1);
        p.pop();
        B--;
    }
    return sum;
}
```

Minimum Cost of ropes :- << GeeksforGeeks >>

```
class Solution
{
    public:
    //Function to return the minimum cost of connecting the ropes.
    long long minCost(long long arr[], long long n) {
        // Your code here
        long long cost =0;
        // min heap
        priority_queue<long long, vector<long long>, greater<long long>>p;
        for(long long i=0;i<n; i++)
        p.push(arr[i]);

        long long first, second;
        while(p.size() != 1)
        {
            first =p.top();
            p.pop();
            second = p.top();
            p.pop();
            first += second;
            cost += first;
            p.push(first);
        }
        return cost;
    }
};
```

Magician and Chocolates << InterviewBit >>

```cpp
int Solution::nchoc(int A, vector<int> &B) {
    priority_queue<int> p;
    long long int total = 0;

    for (int i = 0; i < B.size(); i++)
        p.push(B[i]);
    while (A && !p.empty()) {
        total = (total + p.top()) % 1000000007;
        if (p.top() / 2)
            p.push(p.top() / 2);
        p.pop();
        A--;
    }
    return total;
}
```

K largest elements << GeeksforGeeks >>

```cpp
class Solution
{
public:
    // Function to return k largest elements from an array.
  vector<int> kLargest(int arr[], int n, int k)
    {
        vector<int> ans;
        // Min Heap
        priority_queue<int, vector<int>, greater<int>> p;
        for (int i = 0; i < k; i++)
        {
            p.push(arr[i]);
        }
        // Compare remaining elements with the top element
        for (int i = k; i < n; i++)
        {
            // If the top element is weaker, pop it and push the current element
            if (p.top() < arr[i])
            {
                p.pop();
                p.push(arr[i]);
            }
        }
        while (!p.empty())
        {
            ans.push_back(p.top());
            p.pop();
        }
        reverse(ans.begin(), ans.end());

        return ans;
    }
};
```

LEARN DSA WITH C++

WEEK :: 11 **DAY: 03** **DATE: 08-07-2023**

HEAP :: PAIR

PAIR :- SLT

(int , char)
| |
First second
	class Node
	{
	int a;
	char b;
	}
		Insert value :-
		P.First = 10;
		P.second = 'D' ;
		P = make_pair(10, 'D');

Nested - Pair :-

class Node
{
int a;
int b;
int d;
}
	Pair<int, int, int>p; ---> Wrong
	
	Pairt < int, pair<int, int>>P; ------> Correct
	
	pair<int, pair<int, pair<int, int>>P; ------> Right
	pair= make-pair(10, make-pair(20, 30);
		Insert value :-
		
		P.first = 10;
		P.second.first = 20;
		P.second.second = 30;

Create Pair :-

```cpp
#include<iostream>
using namespace std;

int main()
{
    // create pair
    pair<int,int>p;
    p.first = 10;
    p.second = 20;

    cout<<p.first<<" "<<p.second;
return 0;
};
```

```cpp
#include<iostream>
using namespace std;

int main()
{
    // create pair
    pair<int, int>p;
    p = make_pair(10, 20);

    cout<<p.first<<" "<<p.second;

return 0;
};
```

```cpp
#include<iostream>
using namespace std;

int main()
{
    // create pair
    pair<int,pair<int,int>>p;
    p.first = 10;
```

```cpp
//   Copy one pair to another
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    // create pair
```

```
    p.second.first = 20;
    p.second.second = 30;

    cout<<p.first<<" "<<p.second.first<<"
"<<p.second.second;

return 0;
};
```

```
    pair<int,int>p;
    p = make_pair(10, 20);
    // copy
    pair<int, int>p2;
    p2 = p;
    cout<<p2.first<<" "<<p2.second;

return 0;
};
```

Sorting Using Vector & Pair :- Base on First Element

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
    vector<pair<int, int>>v;
    v.push_back(make_pair(10, 20));
    v.push_back(make_pair(12, 22));
    v.push_back(make_pair(8, 15));
    v.push_back(make_pair(11, 19));
    v.push_back(make_pair(14, 5));

    // sort element in ascending order
    sort(v.begin(), v.end());
    // sort element in descending order
    sort(v.rbegin(), v.rend());

    // print element
    for(int i=0; i<5; i++)
    cout<<v[i].first<<" "<<v[i].second<<endl;

return 0;
};
```

Sorting Using Vector & Pair :- Base on Second Element

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

bool sortbysecond(pair<int, int>a, pair<int,int>b)
{
    return a.second < b.second || (a.second == b.second && a.first < b.first);  // ascending
    return a.second > b.second || (a.second == b.second && a.first > b.first);  // descending
```

```
};
int main()
{
    vector<pair<int, int>>v;
    v.push_back(make_pair(10, 20));
    v.push_back(make_pair(12, 22));
    v.push_back(make_pair(8, 15));
    v.push_back(make_pair(11, 19));
    v.push_back(make_pair(4, 15));

    // sort element
    sort(v.begin(), v.end(), sortbysecond);

    // print element
    for(int i=0; i<5; i++)
    cout<<v[i].first<<" "<<v[i].second<<endl;

return 0;
};
```

Merge K sorted arrays! << InterviewBit >>

```
vector<int> Solution::solve(vector<vector<int> > &A) {
    vector<int>ans;
    int row = A.size();
    int col = A[0].size();



    // Min Heap
priority_queue<pair<int, pair<int, int>>, vector<pair<int, pair<int, int>>>, greater<pair<int,
pair<int, int>>>> minheap;

// Insert first column into minheap
for (int i = 0; i < row; i++)
    minheap.push(make_pair(A[i][0], make_pair(i, 0)));

pair<int, pair<int, int>> p;
while (!minheap.empty()) {
    // Get minimum element, top
    p = minheap.top();
    // Insert the value into the answer vector
    ans.push_back(p.first);
    row = p.second.first;
    col = p.second.second;
    minheap.pop();
    // Insert the next element of that row into minheap, first check whether that row is valid
    if (col < A[0].size() - 1)
        minheap.push(make_pair(A[row][col + 1], make_pair(row, col + 1)));
}
```

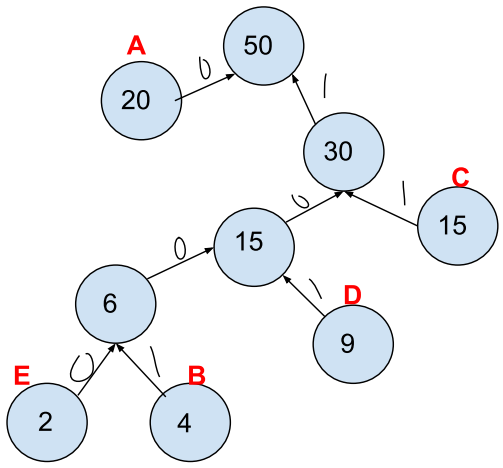
```
return ans;
}
```

Huffman Encoding

Huffman Coding is used because it reduces the size and improves the speed of transmission.
For data transmission using fax and text. It can be used to compress the file.

Bit allocated :-

A B C D E
20 4 15 9 2 <— Freq
Small element
20 15 9 6
20 15 15
20 30
50



A = 0
C = 11
D = 101
E = 1000
B = 1001

Step of Huffman :-

1. Create min Heap : Insert element
2. 2 element pop
3. One left and one right take small element
4. Then combine : repet

Huffman Encoding << GeeksforGeeks >>

```
class Solution {
public:
    class Node {
    public:
        int freq;
        Node* left;
        Node* right;

        Node(int count) {
            freq = count;
            left = right = NULL;
        }
    };

    class Compare {
    public:
        bool operator()(Node* a, Node* b) {
            return a->freq > b->freq;
        }
```

```
    };

    void pre_order(Node* root, string S, vector<string>& Huff) {
        if (!root)
            return;

        if (!root->left && !root->right) {
            Huff.push_back(S);
            return;
        }
        // left
        pre_order(root->left, S + '0', Huff);
        // Right
        pre_order(root->right, S + '1', Huff);
    }

    vector<string> huffmanCodes(string S, vector<int> f, int N) {
        // Create min Heap
        priority_queue<Node*, vector<Node*>, Compare> minheap;

        // Push every freq in heap
        for (int i = 0; i < N; i++) {
            Node* newNode = new Node(f[i]);
            minheap.push(newNode);
        }

        while (minheap.size() > 1) {
            // Extract 2 minimum nodes from the heap
            Node* first = minheap.top();
            minheap.pop();
            Node* second = minheap.top();
            minheap.pop();

            Node* root = new Node(first->freq + second->freq);
            root->left = first;
            root->right = second;
            minheap.push(root);
        }

        Node* root = minheap.top();
        vector<string> huff;
        pre_order(root, "", huff);
        return huff;
    }
};
```

LEARN DSA WITH C++

WEEK :: 11 **DAY**: 05 **DATE**: 09-07-2023

HEAP :: HARD QUESTION

Find median in a stream << GeeksforGeeks >>

```
class Solution
{
    public:
    priority_queue<int>max;
    priority_queue<int, vector<int>, greater<int>>min;
    double median;
    //Function to insert heap.
    void insertHeap(int &x)
    {
        // Heap is empty
        if(max.empty() && min.empty())
        {
            median = x;
            max.push(x);
            return;
        };
        // Element is present in heap

        // Max heap, Left side
        if(x<= median)
        max.push(x);

        // Min heap right side
        else
        min.push(x);

    }

    //Function to balance heaps.
    void balanceHeaps()
    {
        // Left side is greater
        if(max.size()> min.size())
        {
            min.push(max.top());
            max.pop();
        }
        //Right side is greater
        else
        {
            max.push(min.top());
            min.pop();
        }
    }

    //Function to return Median.
    double getMedian()
    {
        if(abs(max.size()-min.size()) >1)
        balanceHeaps();
```

```
        // Max == min
        if(max.size() == min.size())
        {
            median = max.top() + min.top();
            median /= 2;
        }
        //Max > Min
        else if(max.size()> min.size())
        median = max.top();
        // Min > max
        else
        median = min.top();

        return median;
    }
};
```