Striver’s SDE Sheet

Day 12: Heap

Min Heap

Implement the Min Heap data structure. For information about Heap click here. You will be given 2 types of queries:-

Insert X in the heap.

Print the minimum element from the heap and remove it.

Ans:

Standard question

Code:

```
#include <bits/stdc++.h>
void minHeapify(int i,vector<int>&myheap){
    int lchild=2*i+1;
    int rchild=2*i+2;
    int smallest=i;
    if(myheap[lchild]<myheap[i] && lchild<myheap.size()){
        smallest=lchild;
    }
    if(myheap[rchild]<myheap[i] && myheap[rchild]<myheap[lchild] &&
rchild<myheap.size()){
        smallest=rchild;
    }
    if(smallest!=i){
        swap(myheap[smallest],myheap[i]);
        minHeapify(smallest,myheap);
    }
    return;
}
```

```cpp
void insert(vector<int>&myheap,int x){
    myheap.push_back(x);
    int idx=myheap.size()-1;
    int par=(idx-1)/2;
    while(myheap[par]>myheap[idx]){
        minHeapify(par,myheap);
        idx=par;
        par=(par-1)/2;
    }
}
int removehead(vector<int>&myheap){
    int ans=myheap[0];
    swap(myheap[0],myheap[myheap.size()-1]);
    myheap.pop_back();
    minHeapify(0, myheap);
    return ans;
}
vector<int> minHeap(int n, vector<vector<int>>& q) {

    vector<int>myheap;
    vector<int>ans;
    for(int i=0;i<q.size();i++){
        if(q[i][0]==0){
            insert(myheap,q[i][1]);
        }
        else{
            ans.push_back(removehead(myheap));
        }
    }
    return ans;
}
```

Kth Largest Element in an Array

Given an integer array nums and an integer k, return *the* k^th *largest element in the array*.

Note that it is the k^th largest element in the sorted order, not the k^th distinct element.

You must solve it in O(n) time complexity.

Ans:

When ever found k and largest or min then try to approach by heap ~ said by Aditya Varma sir

Approach :maintain a vice vera heap(if max then min heap)

Code:

```
int findKthLargest(vector<int>& nums, int k) {


        priority_queue<int, vector<int>, greater<int>> pq;


        for(int i=0;i<k;i++)
        {
            pq.push(nums[i]);
        }


        for(int i=k;i<nums.size();i++)
        {
            if(nums[i]>pq.top())
            {
                pq.pop();
                pq.push(nums[i]);
            }
        }


        int ans=pq.top();
        return ans;
    }
```

K Max Sum Combinations

You are given two arrays/lists 'A' and 'B' of size 'N' each. You are also given an integer 'K'. You have to find the 'K' maximum and valid sum combinations from all the possible sum combinations of the arrays/lists 'A' and 'B'. Sum combination is made by adding one element from array 'A' and another element from array 'B'.

Ans;

Approach as discuss in previous question we only need top k element of a and b to get our answer so we use heap

Code:

```
vector<int> kMaxSumCombination(vector<int> &a, vector<int> &b, int n, int k){

    // Write your code here.

    priority_queue<int> pq;

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

        for(int j=0;j<n;j++){

            pq.push(a[i] + b[j]);

        }

    }

    vector<int> ans;

    for(int i=0;i<k;i++){

        int f = pq.top();

        pq.pop();

        ans.push_back(f);

    }

    return ans;

}
```

Find Median from Data Stream

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.

Ans:

As we have to maintain sorted order and also have to find the element in middle in at most Logn time.

For maintaining values sorted and find element by index we have a direct **Data structure Ordered Set** but in we have to maintain a Ordered set that also contain duplicate value.

```
#include <ext/pb_ds/assoc_container.hpp>

#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

typedef tree<pair<int,int>, null_type,

        less<pair<int,int>>, rb_tree_tag,

        tree_order_statistics_node_update>

        ordered_set_pair;


class MedianFinder {

public:

    ordered_set_pair st;

    int a=0;

    MedianFinder() {
    }

    void addNum(int num) {
        //insert pair to store duplicate value in Ordered Set
        st.insert({num,a++});
    }

    double findMedian() {
        int n=st.size();
        auto itr = st.find_by_order(n/2);
        double num= (*itr).first;
        if(n%2==0){
            auto itr2=st.find_by_order((n-1)/2);
            double num1=(*itr2).first;
            num=(num+num1)/2;
        }
        return num;

    }
};
```

Merge K Sorted Arrays

You have been given 'K' different arrays/lists, which are sorted individually (in ascending order). You need to merge all the given arrays/list such that the output array/list should be sorted in ascending order.

Ans:

```
#include <bits/stdc++.h>

vector<int> mergeKSortedArrays(vector<vector<int>>&kArrays, int k)

{

    // Write your code here.

    priority_queue<pair<int,pair<int,int>>,

    vector<pair<int,pair<int,int>>>,

    greater<pair<int,pair<int,int>>>>pq;

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

    {

        pq.push({kArrays[i][0],{i,0}});

    }

    vector<int>ans;

    while(!pq.empty())

    {

        int val=pq.top().first;

        int row=pq.top().second.first;

        int col=pq.top().second.second;

        pq.pop();

        ans.push_back(val);

        if(col+1<kArrays[row].size())

        pq.push({kArrays[row][col+1],{row,col+1}});

    }

    return ans;

}
```

Top K Frequent Elements

Given an integer array nums and an integer k, return *the* k *most frequent elements*. You may return the answer in **any order**.

Ans:

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
         unordered_map<int,int> mp;
   // Count the frequency of element
    for(int i = 0;i<nums.size();i++){
        mp[nums[i]]++;
    }
    vector<int> ans;
    // Max Heap with pair
    priority_queue<pair<int,int>> pq;
    //  Insert element is priority queue
    for(auto it : mp){
    // Pushing in the form of  frequency first and element second
        pq.push({it.second,it.first});
    }
    while(k--){
        // Pushing element in ans while k = 0
        ans.push_back(pq.top().second);
        pq.pop();
    }
    return ans;
    }
};
```