Heaps

Identification

- largest / greatest → If comes with k build min Heap.
- smallest / closest → If comes with k build max Heap.

If mentioned guaranteed to be solved using heap.

(k)

Bottom elements can be in any order once deleted top; heap reheapifies and max/min element appears on top.

Time Complexity - O(n log k)

STL

priority_queue<int> - maxHeap.

priority_queue<int, vector<int>, greater<int>> → minHeap

—x—

#1) Find k^{th} Smallest Element.

—x—

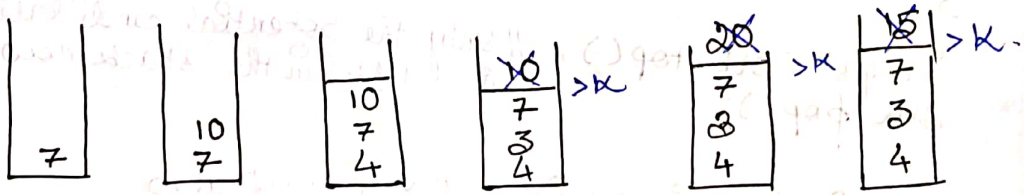
Smallest (Build Max Heap).

I/P = 7 10 4 3 20 15

k = 3.

O/P = 7

Heap (Max)



Code

```
int kthSmallest(int* arr, int l, int r, int k){
    priority_queue<int> maxH;
    for(int i=l; i<=r; i++){
        maxH.push(arr[i]);
        if(maxH.size()>k)
            maxH.pop();
    }
    return maxH.top();
}
```

#2) Print the k largest elements

— x —

Code

```
vector<int> kLargest(vector<int> arr, int n, int k){
    priority_queue<int, vector<int>, greater<int>> minH;
    vector<int> res;
    for(int i=0; i<n; i++){
        minH.push(arr[i]);
        if(minH.size()>k)
            minH.pop()
    }
    while(!minH.empty()){
        res.push_back(minH.top());
        minH.pop();
    }
    reverse(minH.begin(), minH.end())
    return res;
}
```

// on min Heap min or top.
Reversing to get greatest numbers first.

#3) k Closest Numbers

— x —

Return k closest integers to x in the given array

arr = {1,2,3,4,6} O/P = 1, 2, 3, 4.

k = 4; x = 3.

Code

```
vector<int> kClosest(vector<int>& arr, int k, int x){
    priority_queue<pair<int,int>> maxH;
    for(int i=0; i<arr.size(); i++){
        maxH.push({abs(arr[i]-x), arr[i]});
        if(maxH.size()>k)
            maxH.pop();
    }while(!maxH.empty()){
        res.push_back(maxH.top().second);
        maxH.pop();
    }sort;
    return res
}
```

#4) Sort a k sorted array

Given an array of integers of size n; where each element is at most k away from its target position. Print sorted array.

— x —

O/P : 2 3 6 8 12 56.

I/P -

N = 6 ; k = 3.

arr : 2 6 3 12 56 8

at most k.

O/P : 2 3 6 8 12 56.

① Building minH ; as sort in ascending order.
② Pop and push to O/P as soon as size > k.

Code

```
vector<int> nearlySorted(int *arr, int n, int k){
    priority_queue<int, vector<int>, greater<int>> minH;
    vector<int> res;

    for(int i=0; i<n; i++){
        minH.push(arr[i]);
        if(minH.size() > k){
            res.push_back(minH.top());
            minH.pop();
        }
    }

    while(!minH.empty()){
        res.push_back(minH.top());
        minH.pop();
    }
    return res;
}
```

#5) Top k frequent elements

Given an integer array `nums` and an integer k; return the k most frequent elements (in any order).

— x —

I/P — nums : [1, 1, 1, 2, 2, 3], k = 2.
(1 — 3 times, 2 — 2 times)

O/P — [1, 2]

longest (so minH)

code

```
vector<int> topKfrequent(vector<int>& nums, int k){
    unordered_map<int, int> hash;
    int size = nums.size();

    for(int i=0; i<size; i++){
        if(hash.find(nums[i]) == hash.end())
            hash[nums[i]] = 1;          // Not found.
        else
            hash[nums[i]]++;
    }
    vector<int> res;
    priority_queue<pair<int,int>, vector<pair<int,in
                   greater<pair<int,int>> minH;
    for(auto it: hash){
        minH.push({it.second, it.first});
        if(minH.size() > k)
            minH.pop();
    }
    while(!minH.empty()){
        res.push_back(minH.top());
        minH.pop()
    }reverse(res.begin(), res.end());

    return res;
}
```

#6) Frequency Sort Characters

I/P: s = 'tree'

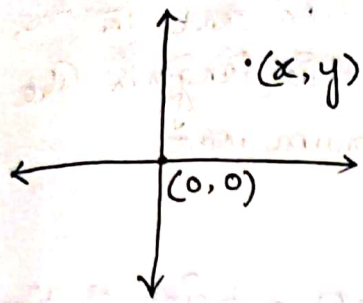
O/P: "eert" / "eetr"

Code

```
string frequencySort(string s) {
    unordered_map<char, int> hash;
    for(int i=0; i<s.length(); i++) {
        if(hash.find(hash[s[i]]) == hash.end())
            hash[s[i]] = 1;
        else
            hash[s[i]]++;
    }
    priority_queue<pair<int, char>> maxH;
    for(auto it: hash)
        maxH.push({it.second, it.first});
    string res;
    while(!maxH.empty()) {
        for(int i=0; i<maxH.top().first; i++)
            res += (maxH.top().second);
        maxH.pop();                    // push number of times it is present
    }
    return res;
}
```

Using Compare function coded in GFG

#7) k Closest Points to the Origin



distance = $\sqrt{(x^2+y^2)}$

Here we will maximize (x^2+y^2)

Closest (MaxHeap).

I/P: points [[1,3], [-2,2]] ; k = 1

O/P: [[-2,2]].

distance, (x,y)

Code

```
vector<vector<int>> kClosest(vector<vector<int>>& pts,
                                        int k){
priority_queue<pair<int, pair<int,int>>> maxH
int size = pts.size();   //no. of colums = 2.
for(int i=0; i<size; i++){
    maxH.push({pts[i][0]*pts[i][0] + pts[i][1]*
                pts[i][1], {pts[i][0], pts[i][1]}});
    if(maxH.size()>k)
        maxH.pop();
}

vector<vector<int>> res;
while(!maxH.empty()){
    vector<int> pts;
    pts.push_back(maxH.top().second.first);
    pts.push_back(maxH.top().second);
    res.push_back(pts);
    maxH.pop();
}
return res;
}
```

#8) Connect "n" ropes to minimise cost.

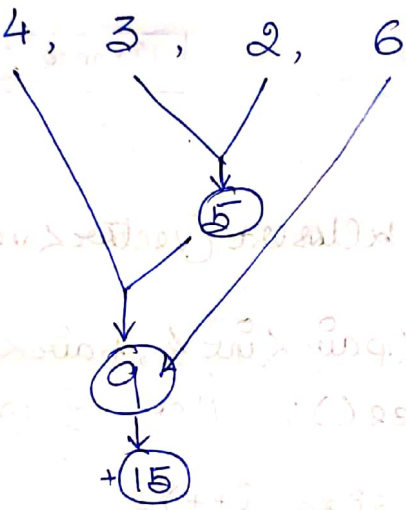
There are given N ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths. The task is to connect ropes with minimum cost.

— x —

n = 4

arr = 4, 3, 2, 6.

**At a point of time, select the 2 minimum from the heap - min



5 + 9 + 15 = 29

Code

```
long long minCost(long long* arr, int n){
    priority_queue<long long, vector<long long>,
                   greater<long long>> minH;
    for(int i=0; i<n; i++)
        minH.push(arr[i]);
    long long cost = 0;
    while(minH.size() >= 2){
        long long a = minH.top(); minH.pop();
        long long b = minH.top(); minH.pop();
        cost = cost + a + b;
        minH.push(a+b);
    }
    return cost;
}
```

#) 9) Reorganize Strings (LeetCode) AMAZON

Rearrange the characters in a string such that no 2 adjacent elements are same

— x —

I/P: "aab"
O/P: "aba"

Idea is to add the most frequently occuring string; followed by the 2nd most occuring string.

```
string reorganizeString (string S){
    unordered_map<char, int> hashmap;
    for(int i=0; i<S.length(); i++){
        if (hashmap.find(S[i]) == hashmap.end())
            hashmap.{ [S[i]] = 1;
        else
            hashmap [S[i]] ++;
    }
    priority_queue <pair<int, int>> maxH;
    for(auto it: hashmap)
        maxH.push({ it.second, it.first });
    string res = " ";
    while(! maxH.empty()){
        pair<int, int> a = maxH.top(); ~~second;~~
        maxH.pop();
        pair<int, int> b = maxH.top(); ~~maxH.pop();~~
        maxH.pop();
        res += a.second;
        res += b.second;
        int freq_most = ~~mass~~ a.first - 1;
        int freq_less = b.first - 1;
        if (freq_most > 0)  maxH.push({ freq_most, a.second});
        if (freq_less > 0) maxH.push({ freq_less, b.second });
    }
    if(! maxH.empty()){ // 1 element left
        pair<int, int> top = maxH.top(); ~~maxH.top();~~
```

```
        maxH.pop();
        if(top.first > 1)  // If the last element has a
            return "";     frequency more than 1 then the
                           result can't be achieved
        else
            res += top.second;(It has to be together)

    }

    return res;

}
```
