WEEK-2

1. **Rotate a linked list**

```
class Solution {
public:
   int getlength(ListNode* head1)
   {
      int count=1;
      while(head1->next!=NULL)
      {
         count++;
         head1=head1->next;
      }
      return count;
   }
   ListNode* rotateRight(ListNode* head, int k) {
      if(head==NULL || head->next==NULL)return head;
      int l=getlength(head);
      k=k%l;
   for(int i=1;i<=k;i++)
   {
      ListNode* tail=head;
      ListNode* tailprev=NULL;
      while(tail->next!=NULL)
      {
         tailprev=tail;
         tail=tail->next;


      }
      tail->next=head;
      tailprev->next=NULL;
```

```
        head=tail;
    }
    return head;
    }
};
```

2. **Clone a linked list with random and next pointer**

```
class Solution {
   private:
    void attail(Node* &head,Node* &tail,int val){
        Node* temp=new Node(val);
        if(head==NULL){
            head=temp;
            tail=temp;

        }
        else{
            tail->next=temp;
            tail=temp;
        }
    }
    public:
    Node *copyRandomList(Node *head)
    {
        //Write your code here
        Node* temp=head;
        Node* clonehead=NULL;
        Node* clonetail=NULL;
        while(temp!=NULL){
            attail(clonehead,clonetail,temp->val);
            temp=temp->next;
        }

        Node* orignal=head;
        Node* clone=clonehead;
        while(orignal!=NULL&&clone!=NULL){
            Node* next1=orignal->next;
          orignal->next=clone;
           orignal=next1;

           next1=clone->next;
            clone->next=orignal;
            clone =next1;
        }


         temp=head;
        clone=clonehead;
        while(temp!=NULL){
            if(temp->next!=NULL){
                if(temp->random!=NULL){
                    temp->next->random=temp->random->next;
                }
                else{
                    temp->next->random=temp->random;
                }
                temp=temp->next->next;
            }

        }
        orignal=head;
```

```
        clone=clonehead;
        while(orignal!=NULL&&clone!=NULL){
            orignal->next=clone->next;
            orignal=orignal->next;
            if(orignal!=NULL)
            clone->next=orignal->next;
            clone=clone->next;
        }
        return clonehead;
    }
};
```

3. 3 Sum

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        int target = 0;
        sort(nums.begin(), nums.end());
        set<vector<int>> s;
        vector<vector<int>> output;
        for (int i = 0; i < nums.size(); i++){
            int j = i + 1;
            int k = nums.size() - 1;
            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];
                if (sum == target) {
                    s.insert({nums[i], nums[j], nums[k]});
                    j++;
                    k--;
                } else if (sum < target) {
                    j++;
                } else {
                    k--;
                }
            }
        }
        for(auto triplets : s)
            output.push_back(triplets);
        return output;
    }
};
```

4. **Trapping Rainwater**

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        int lmax = height[0];
        int rmax = height[n-1];
        int lpos = 1;
        int rpos = n-2;
        int water = 0;
        while(lpos <= rpos)
        {
            if(height[lpos] >= lmax)
            {
                lmax = height[lpos];
                lpos++;
            }
            else if(height[rpos] >= rmax)
            {
                rmax = height[rpos];
                rpos--;
            }
            else if(lmax <= rmax && height[lpos] < lmax)
            {
                water += lmax - height[lpos];
                lpos++;
            }
            else
            {
                water += rmax - height[rpos];
                rpos--;
            }

        }
        return water;
    }
};
```

5. **Remove duplicate from sorted array**

```
class Solution {
public:
   int removeDuplicates(vector<int>& nums) {
      int i=1;
      int n=nums.size();
      if(n == 0) return 0;
      for(int j=1; j<n; j++) {
         if(nums[j] != nums[i-1]) {
            nums[i] = nums[j];
            i++;
         }
      }
```

```
        return i;
    }
};
```

6. Max consecutive ones

```
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int i = 0;
        int maxLen = 0;
        int n = nums.size();

        while(i < n) {
            if(nums[i] == 1) {
                int len = 0;
                while(i < n && nums[i] == 1) {
                    len++;
                    i++;
                }
                maxLen = max(maxLen, len);
            } else i++;
        }
        return maxLen;
    }
};
```

7. N meetings in one room

```
int maximumMeetings(vector<int> &start, vector<int> &end)
{
    // Write your code here.
    int n=start.size();
    vector<pair<int,int>> meetings;
    for(int i=0;i<n;i++){
        meetings.push_back({end[i],start[i]});
    }

    sort(meetings.begin(),meetings.end());

    int maxmeetings=1;
    int prevEnd=meetings[0].first;
    for(int i=1;i<n;i++){
        if(prevEnd<meetings[i].second){
            maxmeetings++;
            prevEnd=meetings[i].first;
        }
    }

    return maxmeetings;
}
```

8. Minimum number of platforms required for a railway

```
int calculateMinPatforms(int at[], int dt[], int n) {
    // Write your code here.
    sort(at,at+n);
    sort(dt,dt+n);
    int i=0,j=0;
    int maxi=0;
    int platforms=0;
    while(i<n&& j<n){
        if(at[i]<=dt[j])i++,platforms++;
        else j++,platforms--;
        maxi=max(platforms,maxi);
    }
    return maxi;
}
```

9. Job sequencing problem

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

```
int jobScheduling(vector<vector<int>> &jobs)
{
    // Write your code here
    priority_queue<int>p;
    unordered_map<int,  vector<int>>mp;
    int mx=0;
    for(int i=0; i<jobs.size(); i++){
        mx=max(mx, jobs[i][0]);
        mp[jobs[i][0]].push_back(jobs[i][1]);
    }
    int ans=0;
    for(int i=mx; i>0; i--){



          for (auto &q : mp[i]) {
            p.push(q);
          }
          if(p.size()==0){
              continue;
          }
          ans += p.top();
          p.pop();


    }
    return ans;


}
```

10. Fractional knapsack problem

```
#include <bits/stdc++.h>
bool comp(pair<int,int>&a,pair<int,int>&b)
{
     return ((double) a.second / a.first) > ((double) b.second / b.first);
}
double maximumValue (vector<pair<int, int>>& items, int n, int w)
{
    double ans=0;
    sort(items.begin(),items.end(),comp);
    for (int i=0;i<n;i++)
    {
        if (items[i].first<=w)
        {
            ans+=items[i].second;
            w-=items[i].first;
        }
        else {
            double temp= (double)(w/(double)items[i].first)*(double)(items[i].second);
            ans+=temp;
            break;
        }
    }
    return ans;
}
```

11. Greedy algorithm to find min num of coins

```
#include <bits/stdc++.h>
int findMinimumCoins(int amount)
{
    // Write your code here
    vector<int> coins = {1,2,5,10,20,50,100,500,1000} ;
    int ct = 0 ;
    for(int i = 8 ; i >= 0 ; i--)
    {
        while(amount>=coins[i])
        {
            amount-=coins[i] ;
            ct++ ;
        }
    }
    return ct ;
}
```

12. N meetings in one room

```
#include<bits/stdc++.h>
bool cmp(pair<int,pair<int,int>> a,pair<int,pair<int,int>> b){
    if(a.second.first==b.second.first) return a.second.second<b.second.second;
    return a.second.first<b.second.first;


}
int maximumActivities(vector<int> &start, vector<int> &finish) {
    // Write your code here.
    vector<pair<int,pair<int,int>>> v;
    for(int i=0;i<start.size();i++){
        v.push_back({start[i],{finish[i],i+1}});
    }
    sort(v.begin(),v.end(),cmp);
    int l=v[0].second.first;
    int c=1;
    for(int i=1;i<start.size();i++){
        if(v[i].first>=l){
            //cout<<"k"<<v[i].first<<endl;
            c++;
            l=v[i].second.first;
        }
    }
    return c;
}
```

13. **Subset Sums**

```
#include<bits/stdc++.h>
void f(vector<int> &num,int i,int sum,vector<int>& ans){
   if(i==num.size()){
                    ans.push_back(sum);
                    return;
            }
            f(num,i+1,sum,ans);
            f(num,i+1,sum+num[i],ans);
}
vector<int> subsetSum(vector<int> &num){
            // Write your code here.
            vector<int> ans;
            f(num,0,0,ans);
            sort(ans.begin(),ans.end());
            return ans;
}
```

14. **Subset – II**

```
class Solution {
public:
   set<vector<int>> st;
   vector<int> num, v;
   void solve(int id) {
      st.insert(v);
      for(int i=id; i<num.size(); i++) {
         v.push_back(num[i]);
         solve(i+1);
         v.pop_back();
      }
   }
   vector<vector<int>> subsetsWithDup(vector<int>& nums) {
      sort(nums.begin(), nums.end());
```

```
        num = nums;
        solve(0);
        vector<vector<int>> ans;
        for(auto x: st) ans.push_back(x);
        return ans;
    }
};
```

15. Combination sum – I

```
class Solution {
private:
    void ans(vector<vector<int>>& res, vector<int>& curr, vector<int>& candidates, int &target, int
curr_sum, int i) {
        if(curr_sum == target) {
            res.push_back(curr);
            return;
        }
        if(i >= candidates.size() || curr_sum > target) {
            return;
        }
        curr.push_back(candidates[i]);
        ans(res, curr, candidates, target, curr_sum+candidates[i], i);
        curr.pop_back();
        if (i + 1 < candidates.size()) {  // Check if i+1 is a valid index
            ans(res, curr, candidates, target, curr_sum, i + 1);
        }
    }
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> res;
        vector<int> curr;
        ans(res, curr, candidates, target, 0, 0);
        return res;
    }
};
```

16. **Combination sum – II**

```
class Solution {
public:
    void helper(int idx, vector<int>& candidates, int n, int sum, int target, vector<int>& temp,
vector<vector<int>>& ans) {
        if (sum == target) {
            ans.push_back(temp);
            return;
        }
        for (int i = idx; i < n; i++) {
            if (i != idx &&  candidates[i] == candidates[i-1])
                continue;
            if (sum + candidates[i] > target)
                break;
            sum += candidates[i];
            temp.push_back(candidates[i]);
            helper(i+1, candidates, n, sum, target, temp, ans);
            sum -= candidates[i];
            temp.pop_back();
        }
    }
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        int n = candidates.size();
        vector<vector<int>> ans;
        vector<int> temp;
        sort(candidates.begin(), candidates.end());
        helper(0, candidates, n, 0, target, temp, ans);
        return ans;
    }
};
```

17. Palindrome partitioning

```
class Solution {
public:
  void partitionHelper(int idx, string s, vector<string>& temp, vector<vector<string>>& ans) {
    if (idx == s.size()) {
      ans.push_back(temp);
      return;
    }
    for (int i = idx; i < s.size(); i++) {
      if (palindrome(s, idx, i) == true) {
        temp.push_back(s.substr(idx, i-idx+1));
        partitionHelper(i+1, s, temp, ans);
        temp.pop_back();
      }
    }
  }

  bool palindrome(string s, int start, int end) {
    while (start <= end) {
      if (s[start] != s[end])
        return false;
      start++;
      end--;
    }
    return true;
  }

  vector<vector<string>> partition(string s) {
    vector<vector<string>> ans;
    vector<string> temp;
    partitionHelper(0, s, temp, ans);
    return ans;
    }
};
```

18. K-th permutation sequence

```
class Solution {
public:
    string getPermutation(int n, int k) {
        int fact=1;
        vector<int> nums;
        for(int i=1;i<n;i++){
            fact=fact*i;
            nums.push_back(i);
        }
        nums.push_back(n);
        k=k-1;
        string ans="";
        while(true){
            ans=ans+to_string(nums[k/fact]);
            nums.erase(nums.begin()+k/fact);
            if(nums.size()==0){
                break;
            }
            k=k%fact;
            fact=fact/nums.size();
        }
        return ans;
    }
};
```

19. Print all permutations of a string/array

```
class Solution {
public:
    void helper(int idx, vector<vector<int>> &ans, vector<int> v){
        if(idx == v.size()){
            ans.push_back(v);
            return;
        }
        for(int i = idx;i < v.size();i++){
            swap(v[i], v[idx]);
            helper(idx+1, ans, v);
            swap(v[i], v[idx]);
        }
    }
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> ans;
        helper(0, ans, nums);
        return ans;
    }
};
```

20. N queens problem

```
class Solution {
public:

    void solve(int col,vector<string>&board,vector<vector<string>> &ans,vector<int>&
leftRow,vector<int>&upperDiagonal,vector<int>&lowerDiagonal,int n){
        if(col==n){
            ans.push_back(board);
            return;
        }

        for(int row=0;row<n;row++){
            if(leftRow[row]==0 && lowerDiagonal[row+col]==0 && upperDiagonal[n-1+col-row]==0){
                board[row][col]='Q';
                leftRow[row]=1;
                lowerDiagonal[row+col]=1;
                upperDiagonal[n-1+col-row]=1;
                solve(col+1,board,ans,leftRow,upperDiagonal,lowerDiagonal,n);
                board[row][col]='.';
                leftRow[row]=0;
                lowerDiagonal[row+col]=0;
                upperDiagonal[n-1+col-row]=0;
            }
        }
    }
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n,'.');
        for(int i=0;i<n;i++){
            board[i]=s;
        }
        vector<int> leftRow(n,0), upperDiagonal(2*n-1,0), lowerDiagonal(2*n-1,0);
        solve(0,board,ans,leftRow,upperDiagonal,lowerDiagonal,n);
        return ans;
    }
};
```

21. Sudoko Solver

```
class Solution {
public:
    bool isValid(vector<vector<char>>&board,int row ,int col,char c){
        for(int i=0;i<9;i++){
            if (board[i][col] == c)
                return false;

            if (board[row][i] == c)
                return false;

            if (board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
                return false;
        }
        return true;
    }
    bool solve(vector<vector<char>>&board){
        for(int i=0;i<9;i++){
            for(int j=0;j<9;j++){
                if(board[i][j]=='.'){
                    for(char c='1';c<='9';c++){
                        if(isValid(board,i,j,c)){
                            board[i][j]=c;
                            if(solve(board)){
                                return true;
                            }
                            else{
                                board[i][j]='.';
                            }
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }
    void solveSudoku(vector<vector<char>>& board) {
        solve(board);
    }
};
```

22. M coloring problem

```
bool isPossible(int node,vector<vector<int>>& mat,int m,int col,vector<int>& color){
//Checking adjacent nodes color
   for(int i=0;i<mat.size();i++){
       if(i!=node && mat[node][i]==1 && color[i]==col) return false;
   }
   return true;
}
bool f(int node,vector<vector<int>> &mat, int m,vector<int> color){
   if(node==mat.size())    return true;
   for(int i=1;i<=m;i++){
       if(isPossible(node,mat,m,i,color)){
           color[node] = i;
           if(f(node+1,mat,m,color))   return true;
           color[node] = 0;
       }
   }
   return false;
}
string graphColoring(vector<vector<int>> &mat, int m) {
   // Write your code here
   int n = mat.size();
   vector<int> color(n,0);
   if( f(0,mat,m,color))   return "YES";
   return "NO";
}
```

23. Rat in a maze

```
void saveWay(vector<char> &ds, vector<string> &sol) {
     string temp = "";
     for(char dir : ds) {
         temp += dir;
     }
     sol.push_back(temp);
}
bool isOpen(int i, int j, int n, vector<vector<int>> &mat, vector<vector<bool>> &visited) {
 if(i < 0 || j < 0) return false;
 if(i >= n || j >= n) return false;
if(!mat[i][j]) return false;
 if(visited[i][j]) return false;
 return true;
}
void solve(int row, int col, int n, vector<vector<int>> &mat, vector<vector<bool>> &visited,
vector<char> &ds, vector<string> &sol) {
    if(row == col && row == n - 1) {
        saveWay(ds, sol);
        return;
    }
if(isOpen(row - 1, col, n, mat, visited)) {
        ds.push_back('U');
        visited[row - 1][col] = true;
        solve(row - 1, col, n, mat, visited, ds, sol);
        visited[row - 1][col] = false;
        ds.pop_back();
    }
if(isOpen(row + 1, col, n, mat, visited)) {
        ds.push_back('D');
        visited[row + 1][col] = true;
        solve(row + 1, col, n, mat, visited, ds, sol);
        visited[row + 1][col] = false;
        ds.pop_back();
    }
if(isOpen(row, col - 1, n, mat, visited)) {
        ds.push_back('L');
        visited[row][col - 1] = true;
        solve(row, col - 1, n, mat, visited, ds, sol);
        visited[row][col - 1] = false;
        ds.pop_back();
    }
if(isOpen(row, col + 1, n, mat, visited)) {
        ds.push_back('R');
        visited[row][col + 1] = true;
        solve(row, col + 1, n, mat, visited, ds, sol);
        visited[row][col + 1] = false;
        ds.pop_back();
    }
}
vector<string> ratMaze(vector<vector<int>> &mat) {
    // Write your code here.
    if(!mat[0][0])
      return {};
    int n = mat.size();
    vector<char> ds;
    vector<string> sol;
    vector<vector<bool>> visited(n, vector<bool>(n, false));
    visited[0][0] = true;
    solve(0, 0, n, mat, visited, ds, sol);
    return sol;
}
```

24. Word Break

```
#include <bits/stdc++.h>
void f(int i, string & s, string & temp, vector<string>& ans, unordered_map<string,bool>& m){
    if(i==s.length()){
        ans.push_back(temp);
        return;
    }
    for(int ind = i; ind<s.length(); ind++){
            int sz = temp.size();
        if(m[s.substr(i,(ind-i+1))]){
            temp.append(s.substr(i,(ind-i+1)));
            temp.push_back(' ');
            f(ind+1,s,temp,ans,m);
            int szn = temp.size();
            while(szn!=sz) {
                temp.pop_back();
                szn--;
            }
        }
    }

}
vector<string> wordBreak(string &s, vector<string> &dic)
{
    // Write your code here
    unordered_map<string,bool> m;
    for(auto i: dic) m[i] = true;
    vector<string> ans;
    string temp;
    f(0,s,temp,ans,m);
    return ans;

}
```

25. The n-th root of an integer

```
#include<math.h>
int NthRoot(int n, int m) {
  // Write your code here.
  int ans= round(pow(m , (1.0/n)));
  if(pow(ans , n)==m ){
    return ans;
  }
  else{
    return -1;
  }
}
```

26. Matrix Median

```
int help(vector<int>& v, int val) {
 int low = 0;
  int high = v.size() - 1;

while (low <= high) {
   int mid = (low + high) / 2;
if (v[mid] <= val) {
      low = mid + 1;
       }
 else {
           high = mid - 1;
       }
   }
   return low;
}

int median(vector<vector<int>>& matrix, int n, int m) {
   int low = 1;
   int high = 1e9;

   while (low <= high) {
     int mid = (low + high) / 2;
       int count = 0;
       for (int i = 0; i < n; i++) {
           count += help(matrix[i], mid);
       }
       if (count <= (n*m) / 2) {
           low = mid + 1;
       }
       else {
           high = mid - 1;
```

```
        }
    }
    return low;
}
```

27. Search single element in a sorted array

```
class Solution {
public:
    int singleNonDuplicate(vector<int>& nums) {
        int left = 0, right = nums.size() - 1;
        while (left < right) {
            int mid = (left + right) / 2;
            if (mid % 2 == 1) {
                mid--;
            }
            if (nums[mid] != nums[mid + 1]) {
                right = mid;
            } else {
                left = mid + 2;
            }
        }
        return nums[left];
    }
};
```

28. Search element in a sorted and rotated array

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        for(int i=0;i<nums.size();i++){
            if(target==nums[i]){
                return i;
            }
        }
        return -1;
    }
};
```

29. Median of two sorted arrays

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        vector<int>v;
 for(auto num:nums1)
            v.push_back(num);
        for(auto num:nums2)
            v.push_back(num);
        sort(v.begin(),v.end());
        int n=v.size();
        return n%2?v[n/2]:(v[n/2-1]+v[n/2])/2.0;
    }
};
```

30. K-th element of 2 sorted arrays

```
int ninjaAndLadoos(vector<int> &row1, vector<int> &row2, int m, int n, int k)
{
if (m > n)
    {
        return ninjaAndLadoos(row2, row1, n, m, k);
    }
    if (m == 0)
    {
        return row2[k - 1];
    }
if (k == 1)
    {
        return min(row1[0], row2[0]);
    }
    int i = min(m, k / 2);
    int j = min(n, k / 2);

    // If row1[i - 1] is greater than row2[j - 1]
    if (row1[i - 1] > row2[j - 1])
    {
        vector<int> newRow2;
        newRow2.assign(row2.begin() + j, row2.end());

        return ninjaAndLadoos(row1, newRow2, m, n - j, k - j);
    }

    vector<int> newRow1;
    newRow1.assign(row1.begin() + i, row1.end());

    return ninjaAndLadoos(newRow1, row2, m - i, n, k - i);
}
```

31. Allocate minimum number of pages

```
#include <bits/stdc++.h>
bool ispossible(long long & hpd, int& limit, vector<int> &times){
        long long count = 1, sum = 0;
        for(int &time: times){
                if(time>hpd) return false;
                if(sum+time<=hpd) sum+=time;
                else {
                        count++;
                        sum = time;
                        }
      }
        return count<=limit;
}
```

```
long long ayushGivesNinjatest(int n, int m, vector<int> time)
{
        long long l = 1, r = LLONG_MAX, mid;
        while(l<r){
                mid = l+(r-l)/2;
                if(ispossible(mid, n, time)) r = mid;
                else l = mid+1;
        }
        return l;
}
```

32. Aggressive cows

```
int aggressiveCows(vector<int> &stalls, int k)
{   sort(stalls.begin(), stalls.end());
    int lo= 1,hi= 0;
    hi = stalls[stalls.size()-1]-stalls[0];
    int ans=lo;int p=0;
    while(hi-lo>=0){
        int mid=lo+(hi-lo)/2;
        int a=1,cnt=stalls[0];
        for(int i=1;i<stalls.size();i++){
            if(stalls[i]-cnt>=mid){
                a++;cnt=stalls[i];
            }
        }
        if(a>=k){ans=mid; lo=mid+1;}
        else{hi=mid-1;}
    }
     return hi;
}
```

33. Max Heap, min heap implementation

```
#include <bits/stdc++.h>
void minheapify(int index,vector<int>& heap){
    int lchild=2*index+1;
    int rchild=2*index+2;
    int smallest=index;
    if(lchild<heap.size()&&heap[lchild]<heap[smallest]) smallest=lchild;
    if(rchild<heap.size()&&heap[rchild]<heap[smallest]) smallest=rchild;
    if(smallest!=index){
        swap(heap[smallest],heap[index]);
        minheapify(smallest,heap);
    }
}
int removemin(vector<int>& heap){
    int ans=heap[0];
    swap(heap[0],heap[heap.size()-1]);
    heap.pop_back();
    minheapify(0,heap);
    return ans;
}
void insert(vector<int>& heap,int element){
    heap.push_back(element);
    int index=heap.size()-1;
    int pi=(index-1)/2;
    while(heap[pi]>heap[index]){
        minheapify(pi,heap);
        index=pi;
        pi=(pi-1)/2;
    }
}
vector<int> minHeap(int n, vector<vector<int>>& q) {
    // Write your code here.
    vector<int> heap;
    vector<int> ans;
    for(int i=0;i<q.size();i++){
        if(q[i][0]==0){
            insert(heap,q[i][1]);
        }
        else{
            ans.push_back(removemin(heap));
        }
    }
    return ans;

}
```

34. Kth largest element

```
class Solution {
public:
    int findKthLargest(vector<int>& v, int k) {
        priority_queue<int,vector<int>,greater<int>> minpq;
        int n = v.size();

        for(int i = 0;i < n;i++){
            if(minpq.size() < k) minpq.push(v[i]);
            else{
                if(minpq.top() < v[i]){
                    minpq.pop();
                    minpq.push(v[i]);
                }
            }
        }
        return minpq.top();
    }
};
```

35. Maximum sum combination

```
#include<bits/stdc++.h>
vector<int> findMedian(vector<int> &arr, int n){
```

```
vector<int>medians;     priority_queue<int>maxh;
priority_queue<int,vector<int>,greater<int>>minh;
for(int i=0;i<n;i++)     {
            maxh.push(arr[i]);
            minh.push(maxh.top());
            maxh.pop();
            if(maxh.size()<minh.size()){
                        maxh.push(minh.top());
                        minh.pop();
}
int median;
if(maxh.size()>minh.size()){
median=maxh.top();
}  else{
median=(maxh.top()+minh.top())/2;
 }
medians.push_back(median);
}
return medians;
}
```

36. Find median from data stream

```
class MedianFinder {
   priority_queue<int> maxHeap;
   priority_queue<int, vector<int>, greater<int>> minHeap;
public:
   MedianFinder() {
   }

   void addNum(int num) {
      if(maxHeap.size() == 0 || maxHeap.top() >= num) maxHeap.push(num);
      else minHeap.push(num);
      balancingHeaps();
```

```
    }

    double findMedian() {
        if(maxHeap.size() > minHeap.size()) return maxHeap.top();
        else if(minHeap.size() > maxHeap.size()) return minHeap.top();
        else return (maxHeap.top() + minHeap.top()) / 2.0;

    }
    void balancingHeaps(){
        if (maxHeap.size() > minHeap.size() + 1) {
                        minHeap.push(maxHeap.top());
                        maxHeap.pop();
                } else if (minHeap.size() > maxHeap.size() + 1) {
                        maxHeap.push(minHeap.top());
                        minHeap.pop();
                }

    }
};
```

37. Merge k sorted arrays

```
#include <bits/stdc++.h>
vector<int> mergeKSortedArrays(vector<vector<int>>&karr, int k)
{
    priority_queue<vector<int>, vector<vector<int>>, greater<vector<int>>>pq; //min heap
    vector<int>ans;
    // insert {first ele of every arr, index, arr no}}
    for(int i=0;i<k;i++) pq.push({karr[i][0], 0, i});
    while(!pq.empty()){
        auto x=pq.top();
        int e=x[0],i=x[1],a=x[2];
        pq.pop();
        ans.push_back(e);
        // put next ele of cur arr in pq if index is valid
        if(i+1<karr[a].size())
            pq.push({karr[a][i+1], i+1, a});
    }
    return ans;
}
```

38. K most freq elements

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k)
    {
        unordered_map<int, int> mp;
        vector<int> ans;

        for(auto& it : nums) {
            mp[it]++;
        }

        priority_queue<pair<int, int>> pq;
        for(auto &it : mp) {
            pq.push({it.second, it.first});
        }

        while (k > 0) {
            ans.push_back(pq.top().second);
            pq.pop();
            k--;
        }
        return ans;
    }
};
```