Q1) CHALLENGE ACCEPTED (CAKEWALK) **topic: RECURSION**

Odyssey has concluded, but no one can forget it, particularly the performance by Shankar–Ehsaan–Loy. All ADA students are feeling melancholic. Udit, the most generous individual in our college, has come up with a challenge to reignite the enthusiasm for ADA among students. Udit always remains alone due to his obsession with subsequences, and his challenge is evidence of it.

Challenge: You are given an array with m elements. Udit asks you to determine whether a subsequence exists in that array where the sum of elements of that subsequence is a multiple of n. However, as you all know, Udit is the most generous person in our college. He **removes any one element** from the array that you don't know. Your task is to check whether the **remaining array** formed by removing "any one" element has a **subsequence whose sum** is a multiple of n(sarcasm spotted). Here, the selected subsequence must contain atleast one element.

A subsequence of an array A, denoted as B, can have none, some, or all elements of array A. For example, if A={1,2,3}, then its subsequences are {},{1},{2},{3},{1,2},{2,3},{1,3}, and {1,2,3}. (According to the question, null subsequence is not allowed to use)

If a subsequence exists from the remaining array after removing any one element whose sum is divisible by n, then print "YES"; otherwise, print "NO".

Input: t (number of test cases), m (number of elements in the array), n (check whether the subsequence sum is divisible by n), arr (array with m elements)

Output: "YES" or "NO"

Test Case:

t=1
m=4, n= 4
arr={1, 2, -2, -1}

Output = "YES"

Explanation:

If we delete 1: Subsequence: 2 + (-2) = 0

If we delete 2: Subsequence: 1 + (-1) = 0

If we delete -2: Subsequence: 1 + (-1) = 0

1If we delete 1: Subsequence: 2 + (-2) = 0
We are getting a subsequence divisible by k, no matter which element is deleted; therefore the answer is ‘YES”

SOLUTION:

Pseudocode:

```
help(m,n,arr,skip,j,sum,flag):
        // base cases
        If (sum divisible by n and flag==1):
                return TRUE

        If j==m:     // array ends
                return FALSE

        If j==skip:
                return help(m,n,arr,skip,j+1,sum,flag)

        // recurrence relation
        If (help(m,n,arr,skip,j+1,sum+arr[j],TRUE)==TRUE or
help(m,n,arr,skip,j+1,sum,flag)==TRUE):
                return TRUE

        Else: return FALSE


CHALLENGEACCEPTED(m,n,arr):

        flag=TRUE
        For (i: 1 to m):
                If help(m,n,arr,i,0,0,0)==FALSE:
                        flag=FALSE
                        break

        If flag==TRUE:
                print(“YES”)
        Else:
                print(“NO”)
```

Time Complexity: exponential:- O(2^m)

Code for Reference:

```
#include<stdio.h>
int help(int n,long long int arr[],int skip,int j,long long sum,int
k,int null){
    if(sum%k==0 && null==1){
        return 1;
    }
    if(j==n){
        return 0;

    }
    if(j==skip){
        return help(n,arr,skip,j+1,sum,k,null);
    }
    int x1=help(n,arr,skip,j+1,sum+arr[j],k,1);
    int x2=help(n,arr,skip,j+1,sum,k,null);
    if(x1==1 || x2==1){
        return 1;
    }
    else{
        return 0;
    }
}
int main(){
    int t;
    scanf("%d",&t);
    for(int i=0;i<t;i++){
        int n;
        scanf("%d",&n);
        int k;
        scanf("%d",&k);
        long long int arr[n];
```

```
        for(int j=0;j<n;j++){
            scanf("%lld",&arr[j]);
        }
        int flag=1;
        for(int j=0;j<n;j++){
            int x=help(n,arr,j,0,0,k,0);
            if(x==0){
                flag=0;
                break;
            }
        }
    if(flag==1){
        printf("YES");
    }
    else{
        printf("NO");
    }
    printf("\n");
    }

}
```

Q2) THE BATTLE OF HOGWARTS **(EASY-MEDIUM)** **topic: BINARY SEARCH**

The battle of Hogwarts has finally begun, and it is Harry's team vs Voldemort's Army.

Voldemort has divided his army into b bases (spread throughout Hogwarts). Each base has a defensive power **d**, and **w** innocent wizards which it is keeping hostage and whom they threaten to kill, if Harry doesn't surrender.

Seeing this, Harry also decided to divide his team into **n** teams, each with a certain attacking power **a.** A team can attack all bases which have a defensive power less than or equal to its attacking power. If a team attacks a base it saves all the wizards in that base.

Harry is still undecided which team to send out first, so he would like to know for each team what is the maximum amount of wizards it can save.(independently of all others).

Testcase:
n=5 , b=4
a= [1 3 5 2 4] (attacking powers)

d, w (defensive powers and wizards at a base) - 4 bases in total
0 1
4 2
2 8
9 4

ans= [1 9 11 9 11]

The first team can only attack the first base

The second team can attack the first and third bases

The third team can attack the first, second and third bases (1 + 2 + 8= 11) and so on

Solution:
Sort the bases in increasing order of their defensive strength.

Then, compute the prefix sum of the wizards (that is, **prefix[j]** would hold the sum of wizards of all the bases from **0...j).**

Finally, for each team, perform a binary search over the prefix sum and the bases arrays to compute how much wizards it could save.

Overall Time complexity: O(s log b + b log b).

Code for Reference:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
#define pb push_back

int main()
{
   ios_base::sync_with_stdio(false);
   int n,b,d,w;
    cin>>n>>b;
```

```
    vector<int> powers(n,0);
    vector<pair<int,int>> bases(b+1,{0,0});
    vector<int> p_sum(b+1,0);

    for(int i=0;i<n;i++)
    {
       cin>>powers[i];
    }

    for (int i=1;i<=b;i++)
    {
        cin>>d>>w;
        bases[i]={d,w};
    }
    sort(bases.begin(),bases.end());
    vector<int> defences(b+1,0);
    for (int i=0;i<=b;i++)
    {
        defences[i]=bases[i].first;
    }

    for(int i=1;i<=b;i++)
    {
        p_sum[i]=p_sum[i-1]+bases[i].second;
    }

    for (int i=0;i<n;i++)
    {
        int x=lower_bound(defences.begin(),defences.end(),powers[i])
        x=x-defences.begin();  // finding the index
        if (defences[x]!=powers[i]) x--;
        cout<<p_sum[x]<<" ";
    }
    cout<<endl;
}
```

Q3) RPG BUT WITH STRINGS (MEDIUM) **topic- DNC & RECURSION**

Welcome to the RPG Quest for the Perfect String!

You are the lone hero who has survived the waves of DSA on the island of C. Now, on a quest to save the world, you come across a dungeon of strings. A string is considered to be perfect for character **'char'** if either of these conditions are true:

1. The length of the string is 1, and it has the required character **'char'**.
2. Length isn't 1, but the first half is perfect for the character (**'char'**+1) while the second half is filled with character **'char'** only.
3. Length isn't 1, but the second half is filled with character 'char' only, while the first half is perfect for the character (**'char'**+1).

Now, guessing whether a string is perfect or not is just too easy for you, so you decide to make it more challenging for yourself. You will have a certain amount of energy initially, and an energy unit is used to change a character at any place into any other character at any instant. You have to try and tell if you can convert the string into a perfect string for the character 'a'. If you can do the task, print the minimum energy required to do it; otherwise, print "-1".

Note- there is no character after 'z'. So there is no ('z'+1). Also, it is given that all the strings are in lowercase alphabets.

Constraints:

- **$n=2^k$** (where k is a whole number and n is the size of the string)
- **$1 \leq n \leq 131,072$**

Solution:

This will always be possible since we can change the characters to whatever we want. The recursive function is quite straightforward once we focus on those 3 conditions:

recur(string &s, char c):

```
if s.size()==1:                                  // BASE CASE
        return 1-(c==s[0])
mid=s.size()/2
string l = s[0:mid], string r= s[mid:s.size()]
int ans1 = recur(l,c+1)                          // Recursive call on left substring
int f1 = count(s.begin()+mid,s.end(),c)          // Count occurrences of c in r
ans1+=(s.size()/2 - f1)
int ans2 = recur(r,c+1)                          // Recursive call on right substring
```

```
int f2 = count(s.begin(),s.begin()+mid,c)          // Count occurrences of c in l
ans2+=(s.size()/2 - f2)
return min(ans1,ans2)                    //Return the minimum of the two computations
```

This will return the minimum energy required to do the task at hand.

Overall Time complexity- O(nlogn)

Sample Test Cases:

n=8, s="cdbbaaaa"

In this case, the answer will be 0. This is because it is already the perfect string for the character 'a'. Explanation-

- the second half of the string ("aaaa") consists of only the character 'a'.
- the first half of the string ("cdbb") is a '*b*'-*good* string, because:
  - the second half of the string ("bb") consists of only the character 'b'
  - the first half of the string ("cd") is a '*c*'-*good* string, because:
    - the first half of the string ("c") consists of only the character 'c';
    - the second half of the string ("d") is the 'd'-good string.

n=8, s="ceaaaabb"

In this case, the answer will be 4.

There will be several ways to do this. Here is one of them:

- we make sure that the first half of the string ("ceaa") consists of only character 'a'. This increases our answer by 2 and converts the string into "aaaa". (ans=2)
- in the second half ("aabb"), we further divide the string:
  - the right half of the string ("bb") is already perfect for character 'b'.
  - the left half of the string ("aa") can be made perfect for character 'c' in the following manner:
    - the right half ("a") will convert to "c". (ans=3)
    - the left half will convert to "d". (or vice versa) (ans=4)

Thus, in 4 steps, we convert it to a perfect string for letter 'a'.

Code for reference:

```
#include <bits/stdc++.h>
using namespace std;

int recur(string &s,char c){
    if (s.size()==1)
        return c!=s[0];

    int mid=(s.size())/2;
    string l=string(s.begin(),s.begin()+mid);    // left half
    string r=string(s.begin()+mid,s.end());      // right half

    int ans1=recur(l,c+1);
    int ans2=recur(r,c+1);

    int f1=count(s.begin()+mid,s.end(),c);
    int f2=count(s.begin(),s.begin()+mid,c);

    ans1+=(s.size()/2-f1);
    ans2+=(s.size()/2-f2);
    return min(ans1,ans2);
}

void solve(int test_case)
{
    long long n;
    string s;
    cin >> n >> s;
    cout<<recur(s,'a')<<"\n";
}

int32_t main()
{
    long long num_test_case = 1;
    cin >> num_test_case;
    for (long long i = 1; i <= num_test_case; i++)
        solve(i);
    return 0;
}
```

Q4) POKEMON MASTER QUEST (HARD) topic: DNC (Divide & Conquer)

Ash has set out on his journey to become a pokemon master. To choose his starter pokemon he is currently standing in professor Oak's lab . Professor Oak has **n** pokemon (all standing next to each other) for Ash to choose from . Ash can choose any number of pokemon to start with, however he wants to choose only that subset which is most friendly.

Each pokemon has a friendliness stat represented by **ai** (**a** is array of lenght **n**) . To help him, Professor provides him with a list of q numbers, the ith of which is the total friendliness stat choosen by the greatest pokemon trainers of history when they started out on their journey.

For all **q** numbers Ash wants to know whether he can choose some pokemon with total stat = **qi.** However there is a condition, he can only choose pokemon in the following way :-

- assume **mid=⌊max(array)+min(array)⌋ / 2,** where max and min — are functions that find the maximum and the minimum stats . In other words, mid is the sum of the maximum and the minimum friendliness stat divided by 2 rounded down.

- Then the array of pokemon is split into two parts left and right. The left array contains all Pokemon which have stat less than or equal mid, and the right array contains all pokemon which have stat greater than mid. Pokemon in left and right keep their relative order from array.

- During the third step we choose which of the left and right arrays we want to keep. The chosen array replaces the current one, and the pokemon in the other array get angry and run away.

You need Ash to find the result of all **q** tests. For each test, he starts again with the same set of **n** Pokemon (whose stats are represented by array **a**).

Solution:

To begin with, you can notice that the split operation does not depend on the order of the ‘a’ array. So we can sort it.

Now let's build a tree of transitions from the original array to all its possible states. You can simply prove that the height of this tree does not exceed log(max). Since **max(current a) − min(current a**) after each operation of the section is reduced at least twice.

Having understood this, we can write a simple recursive search over the states (left,right). The state will describe a sub-segment of the ‘a’ array that is the current array. For each state, we can calculate the current amount (on the segment from left to right) and add it to any convenient collection (set).
Next, to respond to requests, we can simply look at our collected collection

Time complexity: O (n . log n + q log l) **l** is len of set where **l<=n**

CODE FOR REFERENCE:

```
#include <bits/stdc++.h>
using namespace std;

typedef long long int ll;
#define pb push_back

void bisect (vector<int> &arr, set<ll> &st,int low,int high)
{
    if (low>high) return;

    int mid=(arr[low]+arr[high])/2;
    ll sum=0;
    int idx=-1;

    for (int i=low;i<=high;i++)
    {
        sum+=arr[i];
        if (arr[i]<=mid) idx=i;
    }

    st.insert(sum);
    if (idx==high) return;

    bisect(arr,st,low,idx);
    bisect(arr,st,idx+1,high);
}

int main()
{
   ios_base::sync_with_stdio(false);
   int t,n,q,s;

   cin>>t;
```

```
   for(int i=0;i<t;i++)
   {
       cin>>n>>q;
       vector<int> arr(n,0);
       for(int i=0;i<n;i++)
       {
            cin>>arr[i];
       }

       set <ll> st;
       sort(arr.begin(),arr.end());
       bisect(arr,st,0,n-1);

       for (int i=0;i<q;i++)
       {
          cin>>s;
          if (st.find(s)==st.end()) cout<<"No"<<endl;
          else cout<<"Yes"<<endl;
       }
   }
}
```