

# kintsugi-stack-dsa-cpp: COMPETITIVE\_PROGRAMMING

---

"Talk is cheap. Show me the time complexity."

- Author: [Kintsugi-Programmer](#)

Disclaimer: The content presented here is a curated blend of my personal learning journey, experiences, open-source documentation, and invaluable knowledge gained from diverse sources. I do not claim sole ownership over all the material; this is a community-driven effort to learn, share, and grow together.

- <https://codeforces.com/profile/kintsugi-programmer>
- <https://www.tle-eliminators.com/cp-sheet>

## Table of Contents

- [kintsugi-stack-dsa-cpp: COMPETITIVE\\_PROGRAMMING](#)
  - [Table of Contents](#)
- [R800](#)
  - [01 A Halloumi Boxes](#)
  - [02 A Line Trip](#)
  - [03 A Cover in Water](#)
  - [04 A Game with Integers](#)
  - [05 A Jagged Swaps](#)
  - [06 A Doremy's Paint 3](#)
  - [07 A Don't Try to Count](#)
  - [08 A How Much Does Daytona Cost?](#)
  - [09 Goals of Victory](#)
  - [10 Target Practice](#)
  - [11 Ambitious Kid](#)
  - [12 Sequence Game](#)
- [TipsCollectedFromExperiences](#)
- [Array Coloring \[ONSIGHT\]](#)

## R800

---

### 01 A Halloumi Boxes

- <https://codeforces.com/problemset/problem/1903/A>
- brute force, greedy, sortings, \*800
- Analysis
  - n boxes/array a1 a2 ... an
  - a = {a0, a1, a2 ... an-1}
  - subarray = subsegment = segment taken out of array not manipulated ,no change in order

- eg: {a0, a1}, {a1,a2,a3}, etc
- He wants to sort them in non-decreasing order based on their number
- non-decreasing = increasing
- s is atmost k , means :  $s \leq k$
- however, his machine works in a strange way. It can only reverse any subarray of boxes with length **at most k**
- subarrsize  $\leq k$
- Find if it's possible to sort the boxes using any number of reverses.
- So, if  $K \geq 2$  ,Machine's sort is 100%possible at **any number of reverses ANY\_TIMES**
  - if  $k=2$  atleast  $\Rightarrow$  i have power to shift any element anywhere
- eg:
  - 6421
    - 6421 rev 2 nos sub array my initial thought
    - 6412 rev 2 nos sub array
    - 6142 rev 2 nos sub array
    - 1642 rev 2 nos sub array
    - 1624 rev 2 nos sub array
    - 1264 rev 2 nos sub array
    - 1246 rev 2 nos sub array
    - sorting done :0
  - 6421
    - 1246 rev 4 nos sub array optimal from tuts
- atq : according to question
- tl per test = 1sec atq
  - 1sec =  $10^8$  Operations = per test operations
  - $1 \leq k \leq n$  (minitests)  $\leq 100$  acc.to ques (atq)
  - consider  $n=100$  upperbound
  - tl per mini test =  $1\text{sec} / 100$
  - per mini test operations =  $10^8 / 100 = 10^6$
  - if tc per mini test =  $O(n^3)$ 
    - so  $n=100$ , then operation =  $O(n^3) = O(100^3) =$  per mini test operations
    - so  $O(n^3)$  is the upper bound
    - even sol. can have  $O(n^2)$ ,  $O(n)$ ,  $O(n \log n)$  etc. anything below  $O(n^3)$ , but not above  $O(n^3)$
  - Expected TC =  $O(n^3)$
- ml per test = 256mB atq
- at  $k=1$ , no sorting is possible
  - because the foundation of reverse is actually swap any atleast 2 stuff
  - if stuff is only one then it wont make sense to reverse as we lost power to shift any element
- Approach
  - Passing Condition where return YES
    - $K \geq 2$
    - or given array is already sorted
  - else, return False

```

#include<bits/stdc++.h>
using namespace std;
int main(){
    // at extreme proof case use :
    // long long t;
    int t;
    cin>>t;
    while(t--){
        int n,k;
        cin>>n>>k;
        vector<int> v(n);
        int i=0;
        //input
        while(i<n){
            cin>>v[i]; // n order
            i++;
        }
        vector<int> v2=v; // copy // n order
        sort(v2.begin(), v2.end()); // nlogn order
        if ((v2==v)or(k>=2)){ // to check whether initial array is sorted
or not // n order
            cout<<"YES\n";
        } else {
            cout<<"NO\n";
        }
    }
    return 0;
}
// tc O(nlogn) // highest order here
// at n = 100 , tc = 100log100 = 100*7 = 700
// 2^7 ~ 100
// log2(n) = ln n / ln 2
// sc O(n)

```

## 02 A Line Trip

- <https://codeforces.com/problemset/problem/1901/A>
- greedy, math, \*800
- Analysis
  - location&road on number line
    - 0, a1, a2, ..., x
    - Round Trip i.e. total path => 0, a1, ... an, x, an, ..., a1, 0
    - a1, a2 ... Gas station for Tanki Full/ Refill
    - 0 start point
    - if stop at non-station location due to gas empty = gameover
    - no refuel at dest x
  - In this ques, we have to find the capacity of gas tank car should take it for journey, efficiently without stopping
  - tl per test = 2secs

- $2\text{secs} = 2 \times (10^8)$  operations
- $t = 1000$  atq
- $\text{time/testcase} = O(2 \times 10^5)$
- $n = 50$  max atq
- then at  $O(n^3) = O(125000) = O(1.25 \times 10^5) \leq O(2 \times 10^5)$
- TC for minitest = Expected TC =  $O(n^3)$  upper bound
- tlpt = time limit per test
- mlpt = 256mB
- in test case 1
  - $n = 3$
  - $x = 7$
  - $a\{1, 2, 5\}$
  - 0-1-2-5-7-5-2-1-0
  - output = 4
  - gaschanges=
    - 4 start
    - 3 at 1
    - 4 refill
    - 3 at 2
    - 4 refill
    - 1 at 5
    - 4 refill
    - 2 at 7
    - NO Refill at dest x
    - 0 at 5 ,biggest gas consumption, 5->7->5 , 4 units distance
    - 4 refill
    - 1 at 2
    - 4 refill
    - 3 at 1
    - 4 refill
    - 3 end, fully reached ,gas still remaining
  - biggest gas consumption, 5->7->5 , 4 units distance
  - thus min threshold gas capacity is 4 units ,as below it , car would stop at 5->7->5
- now the max capacity of gas tank in any journey = max distance of any 2 gas stations throughout journey
- throughtout journey means a round trip
  - so, after lastGasStation, car will go to x(dest), and find lastGasStation first in return journey
  - so, that distance is  $(\text{lastGasStation} - x) \times 2$
- Approach
  - $C1 = \text{cal. firstGasStation} - 0$
  - $C2 = \text{max}(\text{allDistances}(\text{cal. dist b/w eachGasStation}))$
  - $C3 = \text{cal. } (\text{lastGasStation} - x) \times 2$
  - return  $\text{max}(C1, C2, C3)$

```

#include<bits/stdc++.h>
using namespace std;
int main(){
    long long t;
    cin>>t;
    while(t--){
        long long n,x;
        cin>>n>>x;
        long long i=1;
        long long smax=0;
        vector<long long> v1(n+1,0); //initialize safety //O(n)
        v1[0]=0;
        while(i<=n){// O(n)
            cin>>v1[i];
            // cout<<v1[i];
            long long buff=smax;
            smax=max(buff,(v1[i]-v1[i-1]));
            i++;
        }
        long long smax2=(x-v1[n])*2; // O(1)
        cout<< max(smax,smax2)<<endl;
        // cout<< smax<<" "<<v1[n-1]<<" "<<smax2<<endl;

    }return 0;
}
// Time Complexity: O(n)
// Space Complexity: O(n)

// max(a, b); O(1) // just checks (a < b)
// min(a, b); O(1) // just checks (a < b)
// max_element(v.begin(), v.end()) O(n)
// min_element(v.begin(), v.end()) O(n)

```

- my code is more optimal than tut ;0

## 03 A Cover in Water

- <https://codeforces.com/problemset/problem/1900/A>
- constructive algorithms, greedy, implementation, strings, \*800
- Analysis
  - Filip has a row of cells, some of which are blocked, and some are empty.
  - He wants all empty cells to have water in them.
  - He has two actions at his disposal
    - 1. place water in an empty cell. **FINITE\_TIMES**
    - 2. remove water from a cell and place it in any other empty cell. **ANY\_TIMES**
  - autoOperation
    - if at some moment cell  $i$  ( $2 \leq i \leq n-1$ ) is empty and both cells  $i-1$  and  $i+1$  contains water, then it becomes filled with water. **ANY\_TIMES**
    - magic autofill
  - $N \Rightarrow s = s_1, s_2, s_3, \dots, s_n$

- = ...##.#....##
- now in ...
  - if we just w.w (w=water) => www
  - then we can transfer the middle water to other cells one by one
  - w.w sill it get refill=> www
- ...##.#....##
  - w.w##.#....## 2times fill water manual
  - www##.#....## autofill
  - w.w##.#w...## swap water
  - www##.#w...## autofill
  - w.w##w#w...## swap water
  - www##w#w...## autofill
  - w.w##w#w...## swap water
  - www##w#w...## autofill
  - w.w##w#w...## swap water
  - www##w#w...## autofill
  - w.w##w#w...## swap water
  - www##w#w...## autofill
  - w.w##w#w...## swap water
  - www##w#w...## autofill
  - all buckets filled ;0, count=2 operation 1
- C1: if no. of Consecutive dots(emptyBoxes) >= 3
  - then we only need 1 operation only 2TIMES as we could fill at corner of 3 boxes and middle box will autoOperationAutoFillWater and we can transfer that water to other cells  
**ANY\_TIMES** 2 operation, and regenerate autoOperationAutoFillWater
- C2: else we need to fill all boxes by 1 operation only as here autoOperationAutoFillWater fails
- Expected TC
  - tlpmt 1sec atq
    - mt = t max= 100 atq
    - tlpmt =  $10^8 / 100 = 10^6$
    - n = 100 atq
    - TCpmt =  $O(n^3)$  upperbound
      - as  $O(100^3) = O(10^6)$  = order of tlpmt
  - mlpt 256mB atq
- Approach
  - count no. of dots(emptyBoxes)
  - count no. of Consecutive dots(emptyBoxes)
  - if no. of Consecutive dots(emptyBoxes) >= 3
    - return 2
  - else
    - return no. of dots(emptyBoxes)
- AnotherApproach
  - if i have 3 contineous empty cell, answer is 2 else ,answer is count of all empty cells
  - similar
  - ... => (i-1), (i), (i+1)
    - just fill i-1, i+1
  - if (**s[i]=="."** && i+1<n && **s[i+1]=="."** && i+2<n && **s[i+2]=="."**)
    - return 2

- else
  - return no. of dots(emptyBoxes)

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    long long t;
    cin>>t;
    while (t--){

        long long n;
        cin>>n;

        string s;
        cin>>s;

        int sum=0;
        int i=0;
        int dot=0;

        while(i<s.size()){
            if (s[i]=='.' & sum!=3)
            {
                sum++;
                dot++;
            }
            else if (s[i]=='#' & sum!=3)
            {
                sum=0;
            }
            i++;
        }

        if (sum>=3)
        {
            cout<<2<<endl;
        }
        else
        {
            cout<<dot<<endl;
        }

    }
    return 0;
}
// TC O(n)
// SC O(n)
```

## 04 A Game with Integers

- <https://codeforces.com/problemset/problem/1899/A>
- games, math, number theory, \*800
- Analysis
  - Vanya and Vova are playing a game. Players are given an integer  $n$ . On their turn, the player can add 1 to the current integer or subtract 1
  - Operations any ne
    - $n=n-1$
    - $n=n+1$
  - The players take turns; Vanya starts. If after Vanya's move the integer is divisible by 3, then he wins. If 10 moves have passed and Vanya has not won, then Vova wins.
  - if both players play optimally
    - then in each of player turn he/she will try move to make other one lose
  - eg: if nos is 5
    - $nos = 5$
    - 6 ( vanya  $n++$  ) OR 4 ( vanya  $n--$  )
    - 5 ( vova  $n--$  ) or 7 ( vova  $n++$  ) OR 3 ( vova  $n--$  ) or 5 ( vova  $n++$  )
    - 6 ( vanya  $n++$  ) OR 4 ( vanya  $n--$  ) or 6 ( vanya  $n--$  ) OR 8 ( vanya  $n++$  ) or 2 ( vanya  $n--$  ) OR 4 ( vanya  $n++$  ) or 4 ( vanya  $n--$  ) OR 6 ( vanya  $n++$  )
    - basically she will counter , to remake it even, repetitive till 10
    - she won
  - Expected TC ?
    - $tlpt = 1sec$  atq
    - $mlpt = 256mB$  atq
    - $t = 100$  atq
    - $n = 1000$  atq
    - $1sec = 10^8$  ops
    - $tlpmt = 10^8 / t = 10^8 / 100 = 10^6$
    - Expected TC =  $O(n^2)$ 
      - not  $O(n^3)$ 
        - as  $1000^3 = 10^9 < tlpmt$
      - as putting  $n$  in  $mt$   $O(n^2)$
      - $= 1000^2$
      - $= 10^6$
      - $= tlpmt$
  - NOW , if both play most optimal, then they will reverse each other operations and exhaust the turns
    - eg:  $n=6 \Rightarrow 7$  ( vanya  $n++$  )  $\Rightarrow 6$  ( vova  $n--$  )  $\Rightarrow$  infinite loop
    - $n\%3 == 0$ 
      - if True, divisible before vanya move
        - even vanya could  $+1/-1$
        - it will not be divisible by 3
        - & vova will cancel the effect  $-1/+1$  of vanya till 10 rounds
        - ultimate vova win
      - if False, not divisible before vanya move



- vanya could +1/ -1
- it will be divisible by 3 after vanya move
- & vova will do something
- & vova will cancel the effect -1/ +1 of vova
- and still it will be divisible by 3 after vanya move till 10 rounds
- ultimate vanya win
- numbers
  - 0 DIV
  - 1 (-1=0)
  - 2 (+1=3)
  - 3 DIV
  - 4 (-1)
  - 5 (+1)
  - 6 DIV
  - 7
  - 8
  - 9 DIV
- Eg: 6
  - => 7 => 8 => 9 => .... vanya win
  - => 7 => 6 => 7 => 6 ... vova win if played optimally
- Approach
  - if  $n \% 3 == 0$ 
    - vova win, return Second
  - else if  $n \% 3 != 0$ 
    - vanya win, return Second

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    long long t=0;
    cin>>t;
    while(t--){
        long long n=0;
        cin>>n;
        if (n%3==0){cout<<"Second\n";}
        else{cout<<"First\n";}
    }
    return 0;
}
// TC 0(1)
// SC 0(1)
```

## 05 A Jagged Swaps

- <https://codeforces.com/problemset/problem/1896/A>
- sortings, \*800
- Analysis

- here, permutation is an array
  - of unique elements
  - if array of  $n$  integers, then integer exists of all  $1, \dots, n$
  - $\Rightarrow$  ORDER DOES NOT MATTER
- we want to check if we could sort the permutation with special operation **ANY\_TIMES**
- special operation
  - if  $a[i-1] < a[i] > a[i+1]$ 
    - then swap  $a[i], a[i+1]$
- eg: 1 3 2 5 4
  - target: 1 2 3 4 5
  - now
    - 1 3 2 5 4
    - $1 < 3 > 2$  5 4 satisfies spec ops
    - 1 2 3 5 4 swap :0
    - $1 \ 2 \ 3 < 5 > 4$  satisfies spec ops
    - 1 2 3 4 5 swap :0
    - = target
- Expected TC
  - tlpt 1sec atq
    - t max 5000 atq
    - n max 10 atq
    - 1 sec =  $10^8$  ops
    - $tlpmt = 10^8 / 5 * 10^3$ 
      - =  $10^5 / 5$
      - =  $2 * 10^4$  ops
    - $O(10^4) < tlpmt$
    - $O(n^4) = \text{Expected TC} = tlpmt$
  - mlpt 256mB atq
- Approach optimal
  - IF THE FIRST NUMBER IN THE INITIAL ARRAY IS 1, THEN ANSWER IS YES , ELSE ANSWER IS NO
  - if not 1st element is 1 ,then that element can never shift towards its desired side
  - this is technically bubble sort
  - as if a number is largest, then it would be  $a[i-1] < a[i] > a[i+1]$  obviously
  - our input arrays are already a permutation perfect, so no need to crosscheck

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

int main(){

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

```

        if (arr[0]==1) cout<<"YES\n";
        else cout<<"NO\n";

    }

    return 0;
}
// TC O(n)
// SC O(n)

```

- Approach Brute Force
  - if this is permutation
    - if permutation is already sorted
      - return "YES"
    - else
      - try sort by special operation n times
      - if sorted
        - return "YES"
      - else
        - return "NO"
  - else
    - return "NO"
    - TTYL

```

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

string checkSort(vector<int> arr,vector<int> arr2, int n){
    for ( int faltu=0; faltu<n; faltu++){
        for ( int idx=0; idx<n; idx++){
            if (
                arr[idx]<arr[idx+1] &&
                arr[idx+1]>arr[idx+2] &&
                idx+2<n
            ){
                int temp= arr[idx+1];
                arr[idx+1]= arr[idx+2];
                arr[idx+2]=temp;
            }
        }
        if(arr2==arr) { return "YES"; }
    }
    return "NO";
}

// NO NEED
string checkPert(int n){
    vector<int> arr(n);

```

```

    for (int i=0; i<n; i++) cin>>arr[i];

    vector<int> arr2=arr;
    sort(arr2.begin(),arr2.end());

    for ( int idx=0; idx<n; idx++){if (arr2[idx]!=idx+1) {return "NO";}}
    if(arr2==arr) { return "YES"; } // already sorted
    return checkSort(arr,arr2, n); // we dont feed data types as arguments
}

int main(){

    long long t;
    cin>>t;
    while(t--){
        int n;
        cin>>n;
        cout<<checkPert(n)<<"\n";

    }

    return 0;
}

```

## 06 A Doremy's Paint 3

- <https://codeforces.com/problemset/problem/1890/A>
- constructive algorithms, \*800
- Analysis
  - array  $a = \{a_1, a_2, a_3 \dots a_n\}$
  - n sized array
  - you want equality among the adjacent sums
  - array is good for this ques as
  - there exists a k such that  $a_1 + a_2 = a_2 + a_3 = \dots = a_{n-1} + a_n = k$
  - Can you reorder the elements such that the condition becomes true ???
  - **permute its element** = change its order
  - eg : {1,1,2}
    - 1,2,1 permute done
    - now  $1+2 = 2+1 = 3$  :0 Done
    - "YES"
  - Expected TC?
    - tlpt = 1sec atq
    - =  $10^8$  ops
    - t max=100 = mt atq
    - tlmt =  $10^8 / 100 = 10^6$
    - n = 100
    - $O(n^3) = 100^3 = 10^6 = \text{tlmt}$
    - Expected TC =  $O(n^3)$

- $mlpt = 256mB$  atq
- if Expected TC =  $O(n^3)$ , then
  - $O(n^4)$  NO ABOVE UPPER BOUND
  - $O(n^3)$  YES UPPER BOUND
  - $O(n^2)$  YES BELOW UPPER BOUND
  - $O(n)$  YES BELOW UPPER BOUND
  - $O(n \log^2(n))$  YES BELOW UPPER BOUND
  - $O(n^1)$  YES BELOW UPPER BOUND
  - this helps in thinking solution limits and optimisation
  - solution can be minimal, not exact  $O(n^3)$
  - but still we got to know our limits
- Approach optimal
  - $\Rightarrow$  Generalise the condition
    - $a_1 + a_2 = a_2 + a_3 = \dots = a_{n-1} = a_n$  atq
    - $\Rightarrow a_{i-1} + a_i = a_i + a_{i+1}$
    - $\Rightarrow a_{i-1} + a_i(\text{cancelled}) = a_i(\text{cancelled}) + a_{i+1}$
    - $\Rightarrow a_{i-1} = a_{i+1} !!!$
    - i.e.  $a_1 = a_3 = a_4 = a_5 = \dots$  &&  $a_2 = a_4 = a_6 = \dots$
  - odd index positions should have same nos && even index positions should have same nos
  - NO when i have more than or equal to 3 distinct integers in my array, eg: 1 1 2 3  $\Rightarrow$  no, you cant create any fair ordering
    - $\Rightarrow$  Case of 3 Distinct Integers
  - ELSE NOW if we have  $N_1$  &  $N_2$ , freq.  $f_1, f_2$ 
    - we want either of both cases in  $n=6$ 
      - $\{N_1 N_2 N_1 N_2 N_1 N_2\}$
      - $\{N_2 N_1 N_2 N_1 N_2 N_1\}$
      - $\Rightarrow f_1 = f_2$  AT ODD N
      - YES
    - $n=7$ 
      - $\{N_1 N_2 N_1 N_2 N_1 N_2 N_1\}$
      - $\{N_2 N_1 N_2 N_1 N_2 N_1 N_2\}$
      - $\Rightarrow f_1 = f_2 + 1$
      - $\Rightarrow f_2 = f_1 + 1$
      - $\Rightarrow \text{abs}(f_1 - f_2) = 1$  AT EVEN N
    - if not then we cant achieve our  $a_{i-1} = a_{i+1} !!!$ , then NO
    - $\Rightarrow$  Case of 2 Distinct Integers
  - $\Rightarrow$  Case of 1 Distinct Integers
    - $N_1$ , any n
    - then  $N_1 N_1 N_1 N_1 \dots$
    - whole array same
    - direct YES
  - else NO

```
#include<bits/stdc++.h>
using namespace std;
int main(){
```

```

// t
// mini tests
int t;
cin>> t;
while (t--){
    long long n;
    cin>>n;
    vector<long long> a(n,0);
    // vector input
    for (long long i=0; i<n; i++) {cin>>a[i];} //n

    // freq map
    map<long long, long long> freq_map;
    for (long long i =0; i<n; i++){//n
        freq_map[a[i]]++;//logn
    }
    //nlogn

    if (freq_map.size()>=3) cout<<"No"<<endl;
    else {

        // begin- first element
        // rbegin- last element

        long long freq1 = freq_map.begin()->second;
        long long freq2 = freq_map.rbegin()->second;

        //odd size array
        if (freq1==freq2) cout<<"Yes"<<endl;
        else if ( n%2 ==1 && abs(freq1-freq2)==1) cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
    }
}
return 0;
}

// TC  $O(n\log 2n) = O(100 \cdot \log 2(100)) = O(100 \cdot 7) = O(700)$ 
// SC  $O(n+n) = O(2n) = O(200)$ 

// this problem is imp to teach map,begin,rbegin iterators

```

- this problem is imp to teach map,begin,rbegin iterators

```

// freq map
map<long long, long long> freq_map;
for (long long i =0; i<n; i++){//n
    freq_map[a[i]]++;//logn
}

```

```
if (freq_map.size()>=3) cout<<"No"<<endl;
```

```
// begin- first element
// rbegin- last element

long long freq1 = freq_map.begin()->second;
long long freq2 = freq_map.rbegin()->second;
```

- Approach Brute force
  - Similar thinking but not organised enough at first try out of clue
  - Read number of test cases `t`
  - For each test case:
    - Read array size `x`
    - Read `x` elements into array `v1`
    - Make a copy `v2` and sort it
    - Make a copy `v3` from `v2` and remove duplicates from `v3`
    - If `v3.size() > 2`, return "NO"
    - If all elements are equal, return "YES"
    - If array size is even:
      - If frequency of smallest and largest elements is equal, return "YES"
    - If array size is odd:
      - If the frequency difference between smallest and largest elements is exactly 1, return "YES"
    - If array size is 2, return "YES"
    - Else, return "NO"

```
#include<bits/stdc++.h>
using namespace std;
string goodAPCheck(int x){
    vector<int> v1(x);
    for ( int i=0; i<x; i++) cin>>v1[i];
    vector<int> v2=v1;
    sort(v2.begin(),v2.end());

    vector<int> v3=v2;
    // remove duplicates
    v3.erase(unique(v3.begin(),v3.end()),v3.end());
    if (v3.size()>2 ) return "NO";

    if (count(v1.begin(), v1.end(),v2[0])==x) return "YES";
    if (x%2==0 && count(v1.begin(), v1.end(),v2[0])==count(v1.begin(),
v1.end(),v2[x-1])) return "YES";
    if(x%2!=0 && (count(v1.begin(), v1.end(),v2[0])==count(v1.begin(),
```

```

v1.end(),v2[x-1])+1 || count(v1.begin(),
v1.end(),v2[0])+1==count(v1.begin(), v1.end(),v2[x-1]) )) return "YES";
    if(x==2) return "YES";
    else return "NO";

}
int main(){

    long long t;
    cin>>t;
    while(t--){
        int x = 0;
        cin>>x;
        cout<<goodAPCheck(x)<<"\n";
    }
    return 0;
}

```

## 07 A Don't Try to Count

- <https://codeforces.com/problemset/problem/1881/A>
- brute force, strings, \*800
- Analysis
  - string x, len n
  - string s, len m
  - $n*m \leq 25$ 
    - 1, 25
    - 5, 5
    - 25, 1
  - operation **ANY\_TIMES**
    - if x= "abc"
    - x= x+x "abcabc"
  - Find
    - min. no of operation by which
    - s is substring of x
  - Expected TC?
    - tlpt 2sec atq
      - t  $10^4$  max atq
      - 1sec =  $2*10^8$  ops per test
      - ops/minitests =  $2*10^8 / 10^4 = 2*10^4$  ops = 20000 ops =  $25*10^2$  ops
      - $n*m \leq 25$  max atq
      - $O(nm10^2)$  Upper bound Expected TC
    - mlpt 256mB atq
- Approach Brute Force Tuts
  - where do i finally say ,this is the end ?
    - x-> x+x -> x+x + x+x -> ... -> not infinity but a upperbound



- argument = upper bound is 5
- should not go beyond 5
- $n \rightarrow x$ ,  $m \rightarrow s$
- worst,  $n=1$ ,  $m=25$
- eg :  $x='a'$ ,  $s='aa...25times...a'$
- $x.size() < s.size()$ , till this condition is true, you can never find  $s$  within  $x$
- $a \Rightarrow aa \Rightarrow aaaa \Rightarrow a.8..a \Rightarrow a..16..a \Rightarrow a..32..a$
- $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 8 \Rightarrow 16 \Rightarrow 32$  (its enough ,more than 25 to become super set ), these changes done within 5 operation
- if not done in even 5 operations then ,at 6,  $x=a...64..a$
- if couldnt find str in 25 ,then you can't find in 64 or more ... .answer is impossible  $\Rightarrow -1$

```
#include<bits/stdc++.h>
using namespace std;
bool check(string s,string x)
{
    if (x.size()<s.size()) return false;
    for (int i=0; i<x.size()- s.size()+1; i++) if (x.substr(i,s.size())==s)
return true; //x.substr(i,s.size())==s substring extract
    return false;
} // O((n-m+1)*m)=O(n*m)
int main(){
    int t;
    cin>>t;
    while (t--){
        long long n,m;
        cin>>n>>m;
        string x,s;
        cin>>x>>s;

        string x0 = x;
        string x1 = x0+x0;
        string x2 = x1+x1;
        string x3 = x2+x2;
        string x4 = x3+x3;
        string x5 = x4+x4;

        long long ans = -1;
        if(check(s,x0)) ans=0;
        else if (check(s,x1)) ans=1;
        else if (check(s,x2)) ans=2;
        else if (check(s,x3)) ans=3;
        else if (check(s,x4)) ans=4;
        else if (check(s,x5)) ans=5;
        cout<<ans<<endl;

    }return 0;
}
```

//187 ms      100 KB

```
// TC  $O(2^5 * n * m) = O(32 * n * m)$ 
// SC  $O(2^5 * n) = O(32 * n)$ 
```

- Approach optimal Mine
  - SAME
  - input t testcases
  - each test cases
    - input n,m
    - Wrong, as babb,bbb ,its not -1
      - check if x is substring of s+s
        - if no
          - then x ,even mul by infinite can't be superset or in any combination of s
          - and we took s+s as maybe x="mara", s="rama"
          - return -1
        - if yes
          - then its possible
  - counter=0
  - while counter<=5
    - concatnate till s is substring of x
    - counter++
  - return counter
  - why counter =5 ??
    - counter=m\*n
    - NO, Memory limit exceeded on test 2 1734 ms 262100 KB

```
#include<bits/stdc++.h>
using namespace std;
int checkCount(string x,string s){
    int counter = 0;

    while (counter<=5){
        if ((x).find(s) != string::npos){
            return counter;
        }
        counter++;
        x=x+x;
    }
    return -1;
};

int main(){
    long long t;
    cin >> t;
    while(t--){
```

```

int n=0, m=0;
string x="", s="";
cin>>n>>m>>x>>s;

// if ((s+s).find(x) == string::npos){
//     cout<<-1<<"\n";
// }
// else{
    cout<<checkCount(x,s)<<"\n";
// }

}

return 0;
}

// passed 58 tests containing test cases :)
// 109 ms    100 KB
// Time Complexity: O(n * m)
// Space Complexity: O(n + m) (worst case 32n + m).

```

## 08 A How Much Does Daytona Cost?

- <https://codeforces.com/problemset/problem/1878/A>
- greedy, \*800
- Analysis
  - array a
  - size n
  - int k
  - Find?
    - if exists subsegment(sub array) of a where k is most common element
  - $a = \{a_0, a_1, \dots, a_{n-1}\}$
  - n, k
  - subarray
    - $\{a_1, a_2, a_3\}$
    - $\{a_1\}$
    - $\{a_1, a_2\}$
    - $\{a_3, a_4\}$
  - eg:  $n=5, k=4, a=\{1, 4, 3, 4, 1\}$ 
    - $\Rightarrow \{4, 3, 4\}$
    - $\Rightarrow \text{YES}$
  - Expected TC?
    - tlpt 1sec atq
    - mlpt 256mB atq
    - t max 1000 atq
    - n max 100 atq
    - 1 sec =  $10^8$  ops

- $tlpmt = 10^8 / 1000 = 10^5$
- $O(100^3) = 10^6$  NO
- $O(100^2) = 10^4$  YES
- Expected TC =  $tcpmt = O(n^2)$
- Approach optimal
  - IF k is present in array anywhere, then answer is YES, else no
  - $a = \{a_0, a_1, a_2, k, a_4 \dots a_n\}$
  - we haven't told length of subarray
  - we can take length = 1, {k} is correct too, now in this subarray, k is the highest occurrence as k is only

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int t;cin>>t;while(t--){long long n,k;cin>>n>>k;
    long long a[n];for (int i=0; i<n; i++) cin>>a[i];
    long long number_is_present =0;
    for (int i=0; i<n; i++){if (a[i]==k) {number_is_present=1;break;}}
    if(number_is_present) cout<<"YES"<<endl;
    else cout<<"NO"<<endl;}return 0;
}
// TC O(n)
// SC O(n)
```

- Approach Brute Force
  - SAME nearly
  - One sec,
    - if number exists
      - if array size is 2 or 1
      - or if anywhere  $a[i]=a[i+1]=k$
      - or if its >2
        - if in array bw that 1st occur and last occur ,that number the most occur
  - return yes if any satisfy, else no :0
  - FREAKING, the the limits, loopholes are hidden
  - functions returns>if else with breaks
  - if number exists
    - then its largest at subarray len = 1

```
// template miniTests int1 int2 vectorArrayInt1
#include<bits/stdc++.h>
using namespace std;
string mainGame(int x1, int x2, vector<int> v1){
    // code here
    // x1 n
    // x2 k
    // v1 a
    for ( int i =0; i<x1; i++ ){
```

```

        if (x2==v1[i]) {return "YES";}
    }
    return "NO";
}

void eachMiniTest(){
    int x1=0, x2=0; // factor1 factor2
    cin>>x1>>x2;
    vector<int> v1(x1);

    for (int i=0; i<x1; i++ ) cin>>v1[i];
    cout<< mainGame(x1,x2,v1)<<"\n";
}

int main(){
    long long t; //mini test cases
    cin>>t;
    while(t--){
        eachMiniTest();
    }

    return 0;
}

```

## 09 Goals of Victory

- <https://codeforces.com/problemset/problem/1877/A>
- math, \*800
- Analysis
  - Expected TC ?
    - tlpt 1sec atq
    - mlpt 256 mB atq
    - t max 500 atq =  $5 \cdot 100$
    - n max 100 atq
    - 1sec =  $10^8$  ops
    - tlpmt =  $10^8 / 500$
- Approach
  - given
    - n teams in tournament
    - each match
      - each pair of teams match up once
      - after every match
        - 2 int, as result of match, 2 goals of 2 teams
    - efficiency of team = total no. of goals in each of its matches. - total opponents score in each of its matches.
  - to find
    - efficiency array of each team, one missing

- $a_1, a_2, a_3, \dots, a_{n-1}$
  - $n-1$  teams
  - efficiency of missing team?
    - it can be uniquely determined
- input
  - $t(\text{tests})$
  - $n(\text{teams})$  1
  - efficiencies of  $n-1$  teams 1
  - $n(\text{teams})$  2
  - efficiencies of  $n-1$  teams 2
  - ...
- output
  - missing efficiency 1
  - missing efficiency 2
  - ...
- Approach 1 -- brute force -- optimal
  - suppose 4 teams
    - $n_1 n_2 n_3 n_4$  teams
      - matches (in pairs)
        - $n_1 n_2 \Rightarrow$  scores  $a_1 b_1$
        - $n_1 n_3 \Rightarrow a_2 c_1$
        - $n_1 n_4 \Rightarrow a_3 d_1$
        - $n_2 n_3 \Rightarrow b_2 c_2$
        - $n_2 n_4 \Rightarrow b_3 d_2$
        - $n_3 n_4 \Rightarrow d_3 c_3$
    - $e_1 e_2 e_3 e_4$  eff.s
  - so
    - $e_1 = a_1 + a_2 + a_3 - b_1 - c_1 - d_1 = 3$
    - $e_2 = b_1 + b_2 + b_3 - a_1 - c_2 - d_2 = -4$
    - $e_3 = c_1 + c_2 + c_3 - a_2 - b_2 - d_3 = 5$
    - $e_4 = d_1 + d_2 + d_3 - a_3 - b_3 - c_3$
    - $e_1 + e_2 + e_3 + e_4 = 0$  (oh, just found out !!!)
      - $e_4 = -(e_1 + e_2 + e_3)$  (SOLVED!!!)
      - $e_4 = -(3 - 4 + 5) = -4$
- Formalised
  - Problem Insight
    - Each goal scored by a team increases its own efficiency by 1.
    - The same goal decreases the opponent's efficiency by 1.
    - So for every goal, the total sum of efficiencies changes by +1 and -1.
    - Net change in total efficiency is always 0.
  - Key Observation
    - Initially, before any match, all teams have efficiency 0.
    - So the total sum of efficiencies starts at 0.
    - Since the sum never changes, the final sum of efficiencies is also 0.
  - Given
    - There are  $n$  teams.

- Efficiencies of  $n - 1$  teams are given.
- One team's efficiency is missing.
- Logic to Find Missing Efficiency
  - Let the given efficiencies be:  $A_1, A_2, A_3, \dots, A_{(n-1)}$
  - Let the missing efficiency be  $A_n$ .
  - Since total sum is 0:  $A_1 + A_2 + A_3 + \dots + A_{(n-1)} + A_n = 0$
  - Therefore:  $A_n = -(A_1 + A_2 + A_3 + \dots + A_{(n-1)})$
- Algorithm
  - Read  $n$ .
  - Read the  $n - 1$  efficiencies.
  - Compute their sum.
  - Output the negative of this sum.
- Complexity
  - Time complexity:  $O(n)$
  - Space complexity:  $O(1)$

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    // Read Question and Analyse it Bit-by-bit
    // write all pts in depth, leave no missing dots
    // then dots will connect easily and will give you answer
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t=0; // safe ,no to garbage entry
    cin>>t;
    while(t--){
        int n=0; // safe ,no to garbage entry
        cin>>n;
        n--; // we have n-1 entries
        int res=0; // safe ,no to garbage entry
        while(n--){
            int buff=0; // safe ,no to garbage entry
            cin>>buff;
            res+=buff;
        }
        res=res*(-1);
        cout<<res<<"\n";
    }
    return 0;
}
```

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

int main()
{
```

```

int t; // Variable to store the number of test cases
cin >> t; // Read the number of test cases
while (t--) // Loop through each test case
{
    long long n; // Variable to store the number of teams
    cin >> n; // Read the number of teams
    long long a[n]; // Array to store the efficiency of n-1 teams
    for (long long i = 0; i < n - 1; i++) // Loop to read the
efficiency of n-1 teams
        cin >> a[i]; // Read efficiency of each team
    // inputs

    long long sum = 0; // Variable to store the sum of efficiencies of
n-1 teams
    for (long long i = 0; i < n - 1; i++) // Loop to calculate the sum
of efficiencies
        sum += a[i]; // Add each team's efficiency to the sum

    cout << -1 * sum << endl; // Output the efficiency of the missing
team, which is the negative of the sum
}
return 0; // Return 0 to indicate successful execution
}

// Time Complexity (TC): O(n) = O(100)
// Space Complexity (SC): O(n) = O(100)

```

## 10 Target Practice

- <https://codeforces.com/problemset/problem/1873/C>
- implementationmath, \*800
- Analysis
  - tc and sc
    - 1sec =  $10^8$  ops
    - tlpt = 1sec
    - 1 test =  $10^8$  ops
    - 1 test = 1000 mtest
    - $\Rightarrow$  tlpmt =  $10^8 / 10^3 = 10^5$  ops =  $O(n^5)$  max allowed and  $n = 10$  (10x10matrix)
    - mlpt = 256mb
  - given board
  - 10x10
  - each ring deeper, more points, outermost is 1, innermost is 5
  - person shot "X" in grid of 10x10 "."
  - Find?
    - Scores
  - technically

0123456789



```

1111111111 0
1222222221 1
1233333321 2
1234444321 3
1234554321 4
1234554321 5
1234444321 6
1233333321 7
1222222221 8
1111111111 9

0 1234 5678 9

1 1111 1111 1 0

1 2222 2222 1 1
1 2333 3332 1 2
1 2344 4432 1 3
1 2345 5432 1 4

1 2345 5432 1 5
1 2344 4432 1 6
1 2333 3332 1 7
1 2222 2222 1 8

1 1111 1111 1 9

```

- condition of scoring

- 1
  - 0,0 to 9,0 L 00 10 20 30 40 50 60 70 80 90
  - 0,9 to 9,9 R 09 19 29 39 49 59 69 79 89 99
  - 
  - 0,0 to 0,9 T 00 01 02 03 04 05 06 07 08 09
  - 9,0 to 9,9 B 90 91 92 93 94 95 96 97 98 99
  - relation
    - $i = 0$  or  $9$
    - $j = 0$  or  $9$
    - $1 = 0+1 == 10-9$
- 2
  - 1,1 to 8,1 L 11 21 31 41 51 61 71 81
  - 1,8 to 8,8 R 18 28 38 48 58 68 78 88
  - 
  - 1,1 to 1,8 T 11 12 13 14 15 16 17 18
  - 8,1 to 8,8 B 81 82 83 84 85 86 87 88
  - relation
    - $i, j = 1$  or  $8$
    - $1 = 1+1 == 10-8$
- 5
  - 4,4

- 4,5
  - 5,4
  - 5,5
- this not 2D Array
- this is char incoming
- Approach 1 -- brute force
  - The grid is fixed at 10×10 and consists of 5 concentric square rings, where the outermost ring gives 1 point and each inner ring gives one more point, up to 5 at the center
    - For every test case, we scan the grid cell by cell
      - When a cell contains 'X', we must determine which ring it belongs to
        - We simulate rings using two boundaries: `bound1` starting at 0 and `bound2` starting at 9
          - Ring 1 checks the outer boundary (row or column equal to 0 or 9)
          - Ring 2 checks the next inner boundary (1 or 8)
          - This continues until ring 5
        - If the current cell lies on any side of the current boundary square
          - `(row == bound1 || row == bound2 || col == bound1 || col == bound2)`
          - The current ring number is returned as the score
        - After each ring check, the boundaries are moved inward (`bound1++`, `bound2--`)
      - The returned ring score is added to the total score
    - After processing all cells, the accumulated score is printed
  - The idea works because each cell belongs to exactly one concentric square ring, and shrinking boundaries correctly model these rings without needing an extra 2D scoring array

```
#include<bits/stdc++.h>
using namespace std;
// 1
/*
00 01 02 03 04 05 06 07 08 09
10 19
20 29
30 39
40 49
50 59
60 69
70 79
80 89
90 91 92 93 94 95 96 97 98 99
*/
// 2
/*
11 12 13 14 15 16 17 18
21 28
31 38
41 48
51 58
61 68
```

```

71 78
81 82 83 84 85 86 87 88
*/

int calScore(int row, int col, char c){
    int bound1 =0;
    int bound2 =9;

    for ( int ring =1; ring<=5; ring++){
        if ( (row==bound1) || (row==bound2) ){ return ring;}
        else if ((col==bound1) || (col==bound2)) {return ring;}

        bound1++;bound2--;
    }
    return 0; // fallback
}

void miniTest(){
    int finalScore=0;
    for (int row=0; row<10; row++){
        for (int col=0; col<10; col++){
            char c;
            cin>>c;
            if (c=='X') finalScore+=calScore(row,col,c);
        }
    }
    cout<<finalScore<<"\n";
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin>>t;
    while(t--){
        miniTest();
    }
    return 0;
}

// Approach 1: Time complexity is O(1) per test case (100 cells × 5 rings,
all constants) and space complexity is O(1).

```

- Approach 2 -- optimal
  - The grid is fixed at 10×10 and forms 5 concentric square rings with scores from 1 (outermost) to 5 (innermost)
    - For each test case, the grid is read cell by cell
      - When a cell contains 'X', its score depends only on how close it is to the nearest border
        - Compute the distance from the cell to all four edges:
          - top → *i*
          - left → *j*

- bottom  $\rightarrow 9 - i$
- right  $\rightarrow 9 - j$
- The minimum of these four values gives the ring index (0-based)
  - 0  $\rightarrow$  outer ring, 1  $\rightarrow$  second ring, ..., 4  $\rightarrow$  center
- Add 1 to convert the index into the actual score
  - $\text{score} += \min(\{i, j, 9 - i, 9 - j\}) + 1$
- Repeat for all 100 cells
  - After processing the grid, output the total score
- The intuition is that every step away from the border moves one ring inward, so the closest border uniquely determines the ring of any cell

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

int calScore(int row, int col, char c){
    int top = row;
    int left = col;
    int bottom = 9 - row;
    int right = 9 - col;
    return min({top,bottom,left,right})+1; // fallback
}

void miniTest(){
    int finalScore=0;
    for (int row=0; row<10; row++){
        for (int col=0; col<10; col++){
            char c;
            cin>>c;
            if (c=='X') finalScore+=calScore(row,col,c);
        }
    }
    cout<<finalScore<<"\n";
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin>>t;
    while(t--){
        miniTest();
    }
    return 0;
}

// Approach 2: Time complexity is O(1) per test case (100 cells, constant
work per cell) and space complexity is O(1).
```

```

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t;
    cin >> t;

    while (t--) {
        int score = 0;
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                char c;
                cin >> c;
                if (c == 'X') {
                    score += min({i, j, 9 - i, 9 - j}) + 1;
                }
            }
        }
        cout << score << '\n';
    }
}

```

- Approach 3 -- optimal
  - A predefined 10×10 **score** matrix stores the score of each cell based on its concentric ring on the target
  - The number of test cases **t** is read, and each test case is processed independently
  - For each test case, a 10×10 character grid is read row by row and stored
  - The grid is scanned cell by cell to check for the presence of 'X' (an arrow hit)
  - Whenever an 'X' is found, the corresponding value from the **score** matrix is added to **total\_score**
  - After scanning all 100 cells, the final accumulated score is printed for that test case
  - Time complexity is O(1) per test case (fixed 10×10 grid), and space complexity is O(1) since all data structures are of constant size

```

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

// Predefined score matrix representing the target's rings
// Each element represents the score for that position on the target
const int score[10][10] = {
    {1,1,1,1,1,1,1,1,1,1},
    {1,2,2,2,2,2,2,2,2,1},
    {1,2,3,3,3,3,3,3,2,1},
    {1,2,3,4,4,4,4,3,2,1},
    {1,2,3,4,5,5,4,3,2,1},
    {1,2,3,4,5,5,4,3,2,1},
    {1,2,3,4,5,5,4,3,2,1},
    {1,2,3,4,5,5,4,3,2,1},
    {1,2,3,4,5,5,4,3,2,1},
    {1,2,3,4,5,5,4,3,2,1}
};

```

```

        {1,2,3,4,4,4,4,3,2,1},
        {1,2,3,3,3,3,3,3,2,1},
        {1,2,2,2,2,2,2,2,2,1},
        {1,1,1,1,1,1,1,1,1,1}
    };

    int calScore(int row, int col, char c){
        return score[row][col];
    }

    void miniTest(){
        int finalScore=0;
        for (int row=0; row<10; row++){
            for (int col=0; col<10; col++){
                char c;
                cin>>c;
                if (c=='X') finalScore+=calScore(row,col,c);
            }
        }
        cout<<finalScore<<"\n";
    }

    int main(){
        ios::sync_with_stdio(0);
        cin.tie(0);
        int t;
        cin>>t;
        while(t--){
            miniTest();
        }
        return 0;
    }

    // Time complexity is O(1) per test case (fixed 10×10 grid), and space
    complexity is O(1) since all data structures are of constant size

```

## 11 Ambitious Kid

- <https://codeforces.com/problemset/problem/1866/A>
- math, \*800
- Analysis
  - What the question gives:
    - An array of N integers (can be positive, negative, or zero)
    - One operation allows increasing or decreasing any single element by 1
    - Unlimited operations are allowed on any elements
  - What the question asks:
    - Find the minimum number of operations needed so that
      - the product of all elements in the array becomes 0
    - eg: [2,3,4] mul = 24
      - op1(-1): [1,3,4] mul = 12

- op2(-1): [0,3,4] mul = 0 !!!
  - eg: [-2,3,4] mul = -24
    - op1(+1): [-1,3,4] mul = -12
    - op2(+1): [0,3,4] mul = 0 !!!
- Key observation:
  - A product is 0 if and only if at least one element is 0
  - min(A1, A2 ... An) = solution
- tc
  - tlpt 1sec
  - mlpt 256mb
  - 1sec =  $10^8$  ops
  - ops per testcase =  $10^8 / 1 = 10^8$
  - total minitests = 1
  - tlpmt =  $10^8$  ops
  - N =  $10^5$
  - $O(N^2) \Rightarrow 10^{10} \Rightarrow \text{NO}$
  - $O(N \log_2 N) \Rightarrow \text{YES}$
  - $O(N) \Rightarrow \text{YES}$
  - $O(\log_2 N) \Rightarrow \text{YES}$
  - $O(1) \Rightarrow \text{YES}$
  - Expected TC  $\Rightarrow O(N^2)$
- Approach 1 -- brute force -- optimal
  - To make the product of all elements equal to 0, at least one element must be 0
    - So we only need to convert one element to 0 with minimum operations
  - Each operation changes a number by +1 or -1
    - Therefore, converting any element  $A_i$  to 0 takes  $|A_i|$  steps
  - Intuition:
    - The element closest to 0 needs the fewest steps
      - For [2, 3, 4] → closest is 2 → 2 steps
      - For [-2, -3, -4] → take absolute values [2, 3, 4] → closest is 2 → 2 steps
  - Steps:
    - Read N
    - For each element:
      - Convert it to absolute value (represents steps needed to reach 0)
    - Find the minimum value among all absolute values
      - This value is the answer
  - Complexity:
    - Time Complexity:  $O(N)$
    - Space Complexity:  $O(N)$

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n;
    cin >> n;
    vector<int> v1(n,0);
    for (int i=0; i<n;i++){
```

```

        int i1 = 0;
        cin >> i1;
        v1[i] = (i1 >= 0) ? i1 : (i1 * -1); // or v1[i] = abs(i1);
    }
    cout << *min_element(v1.begin(), v1.end()); // min_element & max_element
    return pointer
    // not sort then v1[1] as its nlogn
    return 0;
}
// TC O(n)
// SC O(n)

```

- Approach 2 -- optimal
  - Read integer N
  - Read the array of N integers
  - Initialize a variable min\_ops to a very large value
    - This will store the minimum operations required
  - For each element in the array:
    - Compute the absolute value  $|A_i|$ 
      - This represents the number of operations needed to make  $A_i$  equal to 0
    - Update min\_ops with the minimum of current min\_ops and  $|A_i|$
  - Output min\_ops
    - This is the minimum number of operations needed to make the product of the array equal to 0
  - Reasoning:
    - The product becomes 0 if at least one element is 0
    - Making the element closest to 0 reach 0 requires the fewest operations
  - Complexity:
    - Time Complexity:  $O(N)$
    - Space Complexity:  $O(N)$

```

#include <bits/stdc++.h>
using namespace std;
int main()
{
    long long n; cin >> n;
    long long a[n];
    for (int i = 0; i < n; i++) { cin >> a[i]; } //n
    long long min_ops = INT_MAX;
    for (int i = 0; i < n; i++) { min_ops = min(min_ops, abs(a[i])); } //n
    cout << min_ops;
    return 0;
}
// TC O(n)
// SC O(n)

```

## 12 Sequence Game



- <https://codeforces.com/problemset/problem/1862/B>
- constructive algorithms, \*800
- Analysis
  - backcreate/guessback/reverse engineering!!!
  - tc
    - tlpt = 2 sec
    - 1sec =  $10^8$  ops
    - mlpt = 256 mb
    - mt max =  $10^4$
    - n max =  $2 \cdot 10^5$
    - Ques Line "The sum of the values of n over all test cases does not exceed  $2 \cdot 10^5$ "
      - this sum of minitest element length  $\leq 2 \cdot 10^5$
      - its the main bottleneck
      - so we can assume not  $10^4$ , but only 1 test case with  $n = 2 \cdot 10^5$ 
        - no need to cal tlpmt
    - 1 test =  $2 \cdot 10^8$  ops
    - $O(n^2) \Rightarrow (2 \cdot 10^5)^2 \Rightarrow 4 \cdot 10^{10} > 2 \cdot 10^4$
    - $O(n^2)$  No ( $4 \cdot 10^{10} > 2 \cdot 10^4$ )
    - $O(n \log 2n)$  Yes ( $< 2 \cdot 10^4$ )
    - $O(n)$  Yes
    - $O(\log 2n)$  Yes
    - ...  $O(1)$  Yes
    - $\Rightarrow$  so avoid making  $O(n^2)$  / 2 nested loops in code!!!
  - Ques
    - Tema and Vika Plays a game
    - Vika : a= +ve int, len m
    - b= new sequence, acc to rule
      - a1 as 1st element
      - then  $a_i (2 \leq i \leq m)$  such that  $a_{i-1} \leq a_i$
      - length of this seq n
    - eg:
      - a=[4,3,2,6,3,3]
      - b=[4,6,3]
      - analysis

```
a 6 [4 3 2 6 3 3]
    Y N N Y N Y
b 3 [4      6   3]
```

```
4 1st element YES => b1
3>=4 => NO (ai-1 <= ai)
2>=3 => NO
6>=2 => YES => b2
3>=6 => NO
3>=3 => YES => b3
```

- then vika gives  $b$  to tema and tema tries to guess  $s$  seq  $a$
- help vika to guess at least 1 num.

- Note that the length of the sequence you output should not exceed the input sequence length by more than two times.

- Test Cases

- input

- $t (1 \leq t \leq 10^4)$
- $n$
- $b_1 b_2 \dots b_n$
- The sum of the values of  $n$  over all test cases does not exceed  $2 \cdot 10^5$

- for each test, output gives 2 lines

- $m$
- $a_1 a_2 \dots a_m$

- mt1

- 3
- 4 6 3
- out
- 6
- 4 3 2 6 3 3

```
b [4 6 3]
possibilities
a [4 3 2 6 3 3] (as described in question)
a [4 6 3 3] also possibilities
... so on , make/guess any, if fits the rule then good !!!
```

- mt2

- 3
- 1 2 3
- out
- 3
- 1 2 3
- In the second sample, Vika could have chosen the original sequence.

- mt3

- 5
- 1 7 9 5 7
- out
- 6
- 1 7 9 3 5 7

```
b 1 < 7 < 9 > 5 < 7
a 1 7 9 ..5 7
```

9 and 5 can't relate as b coming from a  
untill and unless there is unlikely element which exists in  
a which is <9 and <5

possibilities

1 7 9 3 5 7  
1 7 9 6 5 7  
1 7 9 5 5 7  
... so on

- mt4
  - 1
  - 144
  - out
  - 1
  - 144
- mt5
  - 2
  - 1 1
  - out
  - 2
  - 1 1
- mt6
  - 5
  - 1 2 2 1 1
  - out
  - 6
  - 1 2 2 1 1 1
- If there are multiple suitable sequences, you can output any of them
- Approach 1 -- optimal
  - Based on Hint
    - if  $b[i-1] > b[i]$
    - append  $b[i]$  twice

a1 a2 a3 a4 ... am  
to b1 b2 b3 b4 ... bn  
where generally  $b_1 < b_2 < b_3 < b_4 \dots$  ( as only  $a_{i-1} < a_i$  are inserted  
in the array of b)

there can be numerous possibilities (  $a_1 > a_2 < a_3$  )  
one of them can be like (  $a_1 > a_2 = a_3$  ), we are only considering this  
possibility and reverse engineer it..  
so if  $b_1 < b_2 > b_3 < b_4$  (where b3 dont make sense)  
then a would be  $b_1 < b_2 < b_3 = b_3 < b_4$  (to make sense)

- insert all mini test elements into arr b
- then traverse b to append into arr a
  - use the hint
- Reasoning
  - If there are multiple suitable sequences, you can output any of them. (in ques)
  - don't confused by inputs and outputs given, use what question demands and make your own outputs

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int mt=0;
    cin>>mt;
    while(mt--){
        int n=0;
        cin>>n;
        vector<int> b(n);
        for(int i=0; i<n; i++){ //n
            cin>>b[i];
        }
        // input done
        // main output
        vector<int> a;
        a.push_back(b[0]); //1
        for(int i=1; i<n; i++){ // n
            if(b[i-1]>b[i]){
                a.push_back(b[i]); //1
            }
            a.push_back(b[i]);
        }
        cout<<a.size()<<"\n";
        for(int i=0;i<a.size(); i++){ // n
            cout<<a[i]<<" ";
        }
        cout<<"\n";
    }
    return 0;
}
```

```
// Time Complexity (TC):  $O(n) = O(2 \cdot 10^5)$ 
// Space Complexity (SC):  $O(n) = O(2 \cdot 10^5)$ 
```

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

```
int main()
{
    int t; // Number of test cases
    cin >> t;
    while (t--)
    {
        long long n; // Length of sequence b
        cin >> n;
        vector<long long> b(n), a; // Vector b to store input sequence,
// vector a to store the reconstructed sequence
        for (int i = 0; i < n; i++) // Loop to read the sequence b
            cin >> b[i];
        // Initial input of sequence b is completed

        a.push_back(b[0]); // Always add the first element of b to a
        for (int i = 1; i < n; i++) // Loop through the rest of the
// sequence b
        {
            if (b[i] >= b[i - 1])
                a.push_back(b[i]); // If current element is greater than or
// equal to the previous, add it to a
            else
            {
                a.push_back(b[i]); // Add the current element to a
                a.push_back(b[i]); // Add the current element again to a to
// ensure a[i-1] <= a[i]
            }
        }
        cout << a.size() << endl; // Output the size of the reconstructed
// sequence a
        for (auto it : a) // Output each element of the sequence a
            cout << it << " ";
        cout << endl; // New line after each test case
    }
    return 0;
}

// Time Complexity (TC):  $O(n) = O(2 \cdot 10^5)$ 
// Space Complexity (SC):  $O(n) = O(2 \cdot 10^5)$ 
```

## TipsCollectedFromExperiences

---

- Read Question and Analyse it Bit-by-bit
  - write all pts in depth, leave no missing dots
  - reconstruct the test cases with your written logic
    - test cases are misleading and full of confusion
  - then dots will connect easily and will give you answer
- when check TC& SC of program, don't consider TestCasesLoop&Spaces in counting

- 1sec =  $10^8$  Operations
- if 1sec = totalTests
  - operationsPerTestCase =  $10^8 / \text{totalTestCases}$
  - if totalTestCases = 100
    - operationsPerTestCase =  $10^6$  operations
    - $O(n^3)$  is UpperLimit of the question's code
      - as  $O(n^3) = O(100^3) = 10^6 === \text{operationsPerTestCase}$
- always think of extra testcases
- if Expected TC =  $O(n^3)$  ,then
  - $O(n^4)$  NO ABOVE UPPER BOUND
  - $O(n^3)$  YES UPPER BOUND
  - $O(n^2)$  YES BELOW UPPER BOUND
  - $O(n)$  YES BELOW UPPER BOUND
  - $O(n \log 2(n))$  YES BELOW UPPER BOUND
  - $O(n1)$  YES BELOW UPPER BOUND
  - this helps in thinking solution limits and optimisation
  - solution can be minimal, not exact  $O(n^3)$
  - but still we got to know our limits
- and in cp submission , you can see testcases in ID :0
- FREAKING, the the limits, loopholes are hidden
- functions returns>if else with breaks
- => Generalise the condition in question 6R800
  - $a_1 + a_2 = a_2 + a_3 = \dots = a_{n-1} = a_n$  atq
  - =>  $a_{i-1} + a_i = a_i + a_{i+1}$
  - =>  $a_{i-1} + a_i(\text{cancelled}) = a_i(\text{cancelled}) + a_{i+1}$
  - =>  $a_{i-1} = a_{i+1}$  !!!
  - i.e.  $a_1=a_3=a_4=a_5=\dots$  &&  $a_2=a_4=a_6=\dots$
- put this at 1st line of main() code, to fix bug of compiler at running test cases, not interactive program

```
ios::sync_with_stdio(0);
cin.tie(0);
```

- templates

```
// template miniTests int1 int2 vectorArrayInt1
#include<bits/stdc++.h>
using namespace std;
void mainGame(int x1, int x2, vector<int> v1){
```

```

        // code here

    }

    void eachMiniTest(){
        int x1=0, x2=0;// factor1 factor2
        vector<int> v1;
        cin>>x1>>x2;
        for (int i=0; i<x1; i++ ) cin>>v1[i];
        mainGame(x1,x2,v1);

    }

    int main(){
        ios::sync_with_stdio(0);
        cin.tie(0);
        long long t; //mini test cases
        cin>>t;
        while(t--){
            eachMiniTest();
        }

        return 0;
    }

```

```

// template miniTests int1 int2 string
#include<bits/stdc++.h>
using namespace std;
void mainGame(int x1, int x2, string s){
    // code here

}

void eachMiniTest(){
    int x1=0, x2=0;// factor1 factor2
    string s;
    cin>>x1>>x2>>s;
    mainGame(x1,x2,s);

}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    long long t; //mini test cases
    cin>>t;
    while(t--){
        eachMiniTest();
    }
}

```

```
    return 0;
}
```

- Vectors CPP STL

```
vector<int> v1(n); // create ,with n elements mandatory
cin>>v1[0]; // insert
vector<int> v2 = v1; // copy
sort(v2.begin(),v2.end()); // sorting in stl, asc
bool compare= (v1==v2); // compare
```

```
#include<vector>
#include<iostream>
#include<algorithm>
```

- use `long long` instead of `int` for bigger stuff
- max/min : `max(var1 ,var2), min(var1, var2)`
  - make sure var1,var2 has SAME DATATYPE
  - inbuilt
- subarray = sub segment =segment taken out of array not manipulated ,no change in order
  - eg of a = {a0, a1, a2 ... an-1}, subarrs : {a0, a1}, {a1,a2,a3}, etc
- non-decreasing = increasing
- s is atmost k , means :  $s \leq k$
- atq : according to question
- How to Calculate Expected TC? eg: in 1R800
  - tl per test = 1sec atq
    - 1sec =  $10^8$  Operations = per test operations
    - $1 \leq k \leq n$  (minitests)  $\leq 100$  acc.to ques (atq)
    - consider  $n=100$  upperbound
    - tl per mini test = 1sec /100
    - per mini test operations =  $10^8 / 100 = 10^6$
    - if tc per mini test =  $O(n^3)$ 
      - so  $n=100$ , then operation =  $O(n^3) = O(100^3) =$  per mini test operations
      - so  $O(n^3)$  is the upper bound
      - even sol. can have  $O(n^2)$ ,  $O(n)$ ,  $O(n \log n)$  etc. anything below  $O(n^3)$ , but not above  $O(n^3)$
    - Expected TC =  $O(n^3)$



- ```
// 1R800
// at extreme proof case use :
long long t;
```

- ```
// 1R800
// at n = 100 , tc = 100log100 = 100*7 = 700
// 2^7 ~ 100
// log2(n) = ln n / ln 2
```

- ```
// 1R800
// input, n order
cin>>v[i]; // n order

// vector copy, n order
vector<int> v2=v; // copy // n order

// sort stl func, n order
sort(v2.begin(), v2.end()); // nlogn order

// vector compare, n order
if (v2==v) // vector compare // n order
```

- tlpt = time limit per test
- 2secs =  $2 \times (10^8)$  operations
- at 2R800 , tl per test = 2secs
  - 2secs =  $2 \times (10^8)$  operations
  - t=1000 atq
  - time/testcase =  $O(2 \times 10^5)$
  - n=50 max atq
  - then at  $O(n^3) = O(125000) = O(1.25 \times 10^5) \leq O(2 \times 10^5)$
  - TC for minitest = Expected TC =  $O(n^3)$  upper bound

- ```
// max(a, b); O(1) // just checks (a < b)
// min(a, b); O(1) // just checks (a < b)
// max_element(v.begin(), v.end()) O(n)
// min_element(v.begin(), v.end()) O(n)
```

- `vector<long long> v1(10,0);` initialize safety vector
- : What the Fish Ques

- Parity
  - Parity is simply whether a number is even or odd.
  - Even parity: divisible by 2 (like 2, 4, 6, 8...)
  - Odd parity: not divisible by 2 (like 1, 3, 5, 7...)
- 6R800 this problem is imp to teach map,begin,rbegin iterators

```
// freq map
map<long long, long long> freq_map;
for (long long i = 0; i < n; i++) { // n
    freq_map[a[i]]++; // log n
}
```

```
if (freq_map.size() >= 3) cout << "No" << endl;
```

```
// begin- first element
// rbegin- last element

long long freq1 = freq_map.begin()->second;
long long freq2 = freq_map.rbegin()->second;
```

- 11R800, you can use `INT_MAX INT_MIN` to initialise Extreme Value and compare as reference and play with it

```
long long min_ops = INT_MAX;
for (int i = 0; i < n; i++) { min_ops = min(min_ops, abs(a[i])); } // n
```

## Array Coloring [ONSIGHT]

---

- <https://codeforces.com/problemset/problem/1857/A>
- greedy, math, \*800
- Analysis
  - given Array, n integers
  - to do
    - if you can
      - colour array elements in 2 groups/ 2 colors
      - parity of color 1 elements sum = parity of color 2 elements sum
      - print YES
    - else print NO

- Parity
  - Parity is simply whether a number is even or odd.
  - Even parity: divisible by 2 (like 2, 4, 6, 8...)
  - Odd parity: not divisible by 2 (like 1, 3, 5, 7...)
- eg: [1,2,4,3,2,3,5,4]
  - c1: [1,2,3] , c1 sum = even parity
  - c2: [4,2,3,5,4], c2 sum = odd parity
  - YES
- eg: [4,7]
  - NO
- eg: [3,9,8]
  - YES
  - c1: [3,9]
  - c2: [8]
  - both sum even parity
- eg: [1,7]
  - YES
  - c1: [1]
  - c2: [7]
  - both sum odd parity
- eg: [5,4,3,2,1]
  - NO
  - can't make 2color groups with same parity
- Approach
  - if no. of odds = no. of evens
    - YES
  - else if n=3 && no. of odds != 3 or !=1
    - YES
    - actually if [even,even,even] works YES
    - [odd,odd,odd] NO
    - [odd,odd,even] YES
    - [odd,even,even] NO
  - so iff n is odd && odd < even && abs(odd-even)!=1
    - YES
    - [odd,odd,odd,even,even] NO
    - [odd,odd,even,even,even] YES
    - [odd,odd,odd,even,even,even,even] NO

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

```
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin>> t;
```

```
while (t--)\n{\n    int n,odd=0,even=0;\n    cin>>n;\n    vector<int> v1(n,0);\n    for (int i=0; i<n; i++){ \n        cin>>v1[i];\n        if (v1[i]%2==0) {even++;}\n        else {odd++;}\n    }\n    // cout<<odd<<" "<< even<<endl;\n    if (odd==even && n>2) {\n        cout<<"YES\\n";\n    }\n    else if ((n==2 && odd!=even) || (n==3 && odd>even) || (n%2!=0 && \nodd<even && abs(odd-even)!=1) || even==1 || odd==1)\n    {\n        cout<<"YES\\n";\n    }\n    else\n    {\n        cout<<"NO\\n";\n    }\n\n}\n\nreturn 0;\n}
```

---

End-of-File

The [kintsugi-stack](#) repository, authored by Kintsugi-Programmer, is less a comprehensive resource and more an Artifact of Continuous Research and Deep Inquiry into Computer Science and Software Engineering. It serves as a transparent ledger of the author's relentless pursuit of mastery, from the foundational algorithms to modern full-stack implementation.

Made with  Kintsugi-Programmer