C++ Beginner

Q.1 Simple Sum Array
https://www.hackerrank.com/challenges/simple-array-sum/problem?isFullScreen=true

Code :

```
int simpleArraySum(vector<int> ar) {
    int sum=0;

    for (int i=0; i<ar.size(); i++) {
    sum = sum+ar[i];
    }
    return sum;
}
```

Description : Initialise sum variable and add it to each elements of an array

Q.2) Compare The Triplets
https://www.hackerrank.com/challenges/compare-the-triplets/problem?isFullScreen=true

Code :

```
vector<int> compareTriplets(vector<int> a, vector<int> b) {
 int cnt1 = 0, cnt2 = 0;
    int size = a.size();

    for (int i = 0; i < size; ++i) {
        if (a[i] > b[i]) {
            cnt1++;
        } else if (a[i] < b[i]) {
            cnt2++;
        }
    }

    return {cnt1, cnt2};
```

```
}
```

Description

For loop is used and two arrays are compared with the help of for loop

Q.3) A Very Big Sum

https://www.hackerrank.com/challenges/a-very-big-sum/problem?isFullScreen=true

Code :

```
long aVeryBigSum(vector<long> ar) {
long int size=ar.size();
long int sum = 0;
for(int i=0;i<size;i++){
    sum = sum+ar[i];
}

return sum;
}
```

Description : In this you only declare variables using ‘long’ and Find the sum of the elements of an array

Q.4 Diagonal Difference
https://www.hackerrank.com/challenges/diagonal-difference/problem?isFullScreen=true

Code :

```
int diagonalDifference(vector<vector<int>> arr) {
  int n = arr.size();
    int sum1 = 0, sum2 = 0;

    for (int i = 0; i < n; i++) {
        sum1 += arr[i][i];       // Primary diagonal
        sum2 += arr[i][n - 1 - i];  // Secondary diagonal
    }

    return std::abs(sum1 - sum2);
```

```
}
```

Description :

See , In this you only use one for loop , for one diagonal the condition is i==j and for second diagonal the condition is [i][n-1-i] in order to calculate the diagonal sum
Return the absolute value of difference

Q.5 Plus - Minus
https://www.hackerrank.com/challenges/plus-minus/problem?isFullScreen=true

Code :

```
void plusMinus(vector<int> arr) {
    int pcnt = 0, ncnt = 0, zcnt = 0;

    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] < 0) {
            ncnt++;
        }
        else if (arr[i] > 0) {
            pcnt++;
        }
        else {
            zcnt++;
        }
    }

    double x1, x2, x3;

    x1 = static_cast<double>(pcnt) /
static_cast<double>(arr.size());
    x2 = static_cast<double>(ncnt) /
static_cast<double>(arr.size());
    x3 = static_cast<double>(zcnt) /
static_cast<double>(arr.size());

    // How to print up to 10 decimal places
    cout <<  setprecision(6) << x1 << endl;
    cout <<  setprecision(6) << x2 << endl;
    cout <<  setprecision(6) << x3 << endl;
```

```
}
```

Description :

In this Question , we are expected to find the total numbers of positive , negative and zero element and find their ratio with respect to the total element using typecast ,to double because int division gives int , precision is upto 6

Q.6 Staircase
https://www.hackerrank.com/challenges/staircase/problem?isFullScreen=true

Code :

```
void staircase(int n) {

for (int i=1; i<=n; i++) {

    for (int sp=n-i; sp>0; sp--) {
    cout<<" ";
    }

    for (int k=1; k<=i;k++) {
    cout<<"#";
    }
    cout<<endl;

}

}
```

Description :

See it is nothing but pattern printing , where you print spaces first then elements

Q.7) Mini Max Sum
https://www.hackerrank.com/challenges/mini-max-sum/problem?isFullScreen=true

Code :

```
void miniMaxSum(vector<int> arr) {

long long maxi=0 , mini =LONG_LONG_MAX;


for (int i=0; i<5; i++) {
    long long  int sum=0;
for (int j=0; j<5; j++) {
    if (i==j) {
        continue;
    }
    sum+=arr[j];
}
maxi=max(maxi,sum);
mini = min(mini,sum);
}
cout<<mini<<" "<<maxi;
}
```

Description :

In this question, it is asked to find sum of elements of four elements out of 5 but in such a way that we get minimum and maximum and return minimum and maximum .
Logic is the first 4 element's sum < from second element to last element , it is another approach .

Q.8) Birthday Cake Candles

https://www.hackerrank.com/challenges/birthday-cake-candles/problem?isFullScreen=true

Code :

```
int birthdayCakeCandles(vector<int> candles) {
// 1sec -> 10 ^ 8 can run
// n^2 -> 10 ^ 10 can run
// Think of O(n ) and O(log n ) Solution

int maxi = *max_element(candles.begin(),candles.end());
// Also there for minimum
```

```
int cnt=0;

for (int i=0; i<candles.size(); i++) {
    if (candles[i]==maxi) {
    cnt++;
    }
}
return cnt;

}
```

Description : In the above question , we are supposed to find out the maximum elements and if there are suppose [4,4,1,2] here element 4 is highest , we need to give count as 2 (return) as there is 4 occurrence is 2 times.

Q.9) Time Conversion
https://www.hackerrank.com/challenges/time-conversion/problem?isFullScreen=true

Code :

```
string timeConversion(string s) {
string format ="";
format +=s[8];
format+=s[9];
string hr="";
hr+=s[0];
hr+=s[1];

if (format=="PM" && hr!="12") {
unordered_map<int,int> mp;

int count =13;

for (int i=1; i<=11; i++) {
    mp[i]=count;
    count++;
```

```
}
int x   =   stoi(hr);
int temp = mp[x];

string newHr = to_string(temp);

s[0]= newHr[0];
s[1] = newHr[1];

}
else if(format=="AM" && hr=="12"){
    s[0]='0';
    s[1]='0';
}
s.pop_back();
s.pop_back();

return s;
}
```

Description :

In the question Given a time in 12-hour AM/PM format, convert it to military (24-hour) time. Logic is that if time is less than 12 PM it will remain as it is , if it is greater than 12 PM the See , a time in the format "hh:mm:ssAM/PM" and converts it to a 24-hour format. The function extracts the hour part, checks if it's in the PM period and not equal to 12, then converts it to the 24-hour format. If it's in the AM period and equal to 12, it changes it to "00". The final result is the time in 24-hour format, and the seconds and AM/PM indicator are removed.

Q.10) Grading Students

https://www.hackerrank.com/challenges/grading/problem?isFullScreen=true

Code :

```
vector<int> gradingStudents(vector<int> grades) {
        for (int i = 0; i < grades.size(); i++) {
        // Skip grades less than 38
        if (grades[i] < 38) {
            continue;
```

```
        } else {
            // Check if the grade is a multiple of 5
            if (grades[i] % 5 == 0) {
                continue; // No need to round
            } else {
                // Extract the last digit of the grade
                int ld = grades[i] % 10;
                int num = grades[i] / 10;

                int newel = -1; // Initialize to an invalid value

                // Determine the next multiple of 5
                if (ld > 0 && ld < 5) {
                    newel = num * 10 + 5;
                } else if (ld > 5 && ld <= 9) {
                    newel = (num + 1) * 10;
                }

                // Check if rounding is needed and the difference
is less than 3
                if (newel != -1 && newel - grades[i] < 3) {
                    grades[i] = newel; // Update the grade
                }
            }
        }
    }

    // Return the modified vector (optional if the vector is
modified in-place)
    return grades;


}
```

Description :

the code iterates through a vector of grades, skips grades less than 38, and rounds the remaining grades based on specific rules. The rounded grades are then returned in a new vector.

- If the difference between the $grade$ and the next multiple of 5 is less than 3, round $grade$ up to the next multiple of 5.
- If the value of $grade$ is less than 38, no rounding occurs as the result will still be a failing grade.

This is the condition

The gradingStudents function processes a vector of grades, skipping grades below 38. For grades 38 and above, it checks if rounding is necessary. If a grade is not already a multiple of 5, it rounds up to the next multiple of 5, unless the difference between the new and original grade is greater than or equal to 3. The modified grades are returned in a new vector.

Q.11) Apple and Oranges
https://www.hackerrank.com/challenges/apple-and-orange/problem?isFullScreen=true

Code :

```
void countApplesAndOranges(int s, int t, int a, int b, vector<int>
apples, vector<int> oranges) {

for (int i=0; i<apples.size(); i++) {
    apples[i]+=a;


}
for (int i=0; i<oranges.size(); i++) {
oranges[i]+=b;

}
int cnt1=0,cnt2=0;

for (int i=0; i<apples.size(); i++) {
if (apples[i]>=s && apples[i]<=t) {
cnt1++;
}
}
```

```
for (int i=0; i<oranges.size(); i++) {
if (oranges[i]>=s && oranges[i]<=t) {
cnt2++;
}
}
cout<<cnt1<<endl;
cout<<cnt2<<endl;


}
```

Description :

The countApplesAndOranges function adjusts the positions of apples and oranges based on tree locations, counts how many fall within a specified range, and prints the counts separately for apples and oranges.

this function takes parameters representing the range and tree positions, adjusts the positions of apples and oranges, counts how many fall within the specified range, and then prints the counts

Given the value of d for m apples and n oranges, determine how many apples and oranges will fall on Sam's house (i.e., in the inclusive range $[s, t]$)?

For example, Sam's house is between $s = 7$ and $t = 10$. The apple tree is located at $a = 4$ and the orange at $b = 12$. There are $m = 3$ apples and $n = 3$ oranges. Apples are thrown $apples = [2, 3, -4]$ units distance from a, and $oranges = [3, -2, -4]$ units distance. Adding each apple distance to the position of the tree, they land at $[4 + 2, 4 + 3, 4 + -4] = [6, 7, 0]$. Oranges land at $[12 + 3, 12 + -2, 12 + -4] = [15, 10, 8]$. One apple and two oranges land in the inclusive range $7 - 10$ so we print

Q.12) Number Line Jumps (Kangaroo Question)

https://www.hackerrank.com/challenges/kangaroo/problem?isFullScreen=true

Code :

```
string kangaroo(int x1, int v1, int x2, int v2) {

    // if v1<v2 , kangaroo never meet  also for v1=v2

    // v1>v2 okay , but not true always , hamesha thodi possible
for everytime

    // x1+t*v2  this t means total number of time that both
kangaroo jumped at same time

    string ans="";
 if (v1<v2 || v1==v2) {
     ans="NO";

 }
 else{
     if (v1>v2 ) {

         int x= x2-x1;
         int v = v1-v2;

         if (x%v==0) {
         ans = "YES";
         }
         else{
             ans="NO";
         }
     }
 }

    return ans;

}
```

Description :

The formula for calculating the position of an Object after a certain number of jumps is given by

P=Initial Position+(Number of Jumps×Jump Distance)

See for two kangaroos , number of jumps should be same and V1 > V2

Kangaroo Meeting Calculation:

In the fascinating scenario of a kangaroo circus act, where two kangaroos are leaping along a number line, the determination of whether they meet at the same location becomes a captivating mathematical challenge. To solve this, we employ a formula that calculates the number of jumps required for the kangaroos to synchronize their positions.

The formula is expressed as:

$$n = \frac{x_2 - x_1}{v_1 - v_2}$$

Here:

- n represents the number of jumps needed for the kangaroos to meet.
- x_1 and x_2 denote the initial positions of the first and second kangaroos, respectively.
- v_1 and v_2 represent the jump distances or rates of movement for the first and second kangaroos.

To meet at the same location at the same time, the calculated n must be a non-negative integer, ensuring that the kangaroos land after the same whole number of jumps. The condition $(x_2 - x_1) \mod (v_1 - v_2) = 0$ is applied to verify this, where the modulo operation ensures that n is an integer.

This intriguing calculation adds a layer of precision to the choreography, ensuring that the kangaroos deliver a perfectly synchronized and visually captivating performance in the circus act.

Q.13) Between 2 Sets
https://www.hackerrank.com/challenges/between-two-sets/problem?isFullScreen=true

Code :

```
int getTotalX(vector<int> a, vector<int> b) {
int st = *max_element(a.begin(),a.end());
int ed = *min_element(b.begin(),b.end());
int cnt=0;

for (int i=st; i<=ed; i++) {
bool flag1=0;
for (int j=0; j<a.size(); j++) {
if (i%a[j]==0) {
continue;
}
else{
    flag1=1; // element present inin array for x%a[i]!=0
    break;
}
}
if (flag1==0) {
bool flag2=0;
for (int k=0; k<b.size(); k++) {
if (b[k]%i==0)
continue;


else{
    flag2=1;
    break;
}

}
if(flag2==0)cnt++;
}
}
return cnt;
}
```

Description :

1. The elements of the first array are all factors of the integer being considered
2. The integer being considered is a factor of all elements of the second array

These numbers are referred to as being between the two arrays.

Determine how many such numbers exist.

Example

$a = [2, 6]$

$b = [24, 36]$

There are two numbers between the arrays: 6 and 12.

$6\%2 = 0, 6\%6 = 0, 24\%6 = 0$ and $36\%6 = 0$ for the first value.

$12\%2 = 0, 12\%6 = 0$ and $24\%12 = 0, 36\%12 = 0$ for the second value. Return 2.

See , Both conditions means x -> divisible by each number of a , or ,x is multiple of a
And this x should divide each number of b,or, factor of b

Okay , jitna elements , un sabka multiple same hona chahiye aur jo common hai first array se and second array ka factors (a factor is a number or expression that divides another number or expression without leaving a remainder.) common Hona chahiye and return karo total count jitna common hai

Q.14) Breaking the Records
https://www.hackerrank.com/challenges/breaking-best-and-worst-records/problem?isFullScreen=true

Code :

```
vector<int> breakingRecords(vector<int> scores) {
int n = scores.size();
vector<int>ans(2,0);
```

```
vector<int>prefix_max(n,0);
vector<int>prefix_min(n,0);

prefix_max[0]=scores[0];
prefix_min[0]=scores[0];

for (int i=1; i<n; i++) {
    prefix_max[i]=max(prefix_max[i-1],scores[i]);
    if (scores[i]>prefix_max[i-1]) {
    ans[0]+=1;

    }
    prefix_min[i]=min(prefix_min[i-1],scores[i]);
    if (scores[i]<prefix_min[i-1]) {
    ans[1]+=1;

    }

}
return ans;
}
```

Description :

The `breakingRecords` function takes a vector of integer scores as input and calculates the number of times an athlete breaks their highest and lowest score records during a series of competitions. It initializes two vectors, `prefix_max` and `prefix_min`, to keep track of the maximum and minimum scores encountered so far. The function iterates through the input scores, updating these vectors and counting the occurrences of breaking records.

For each score in the input vector, the function compares it with the previous maximum and minimum scores. If the current score surpasses the previous maximum, it increments the count of breaking the highest score record (`ans[0]`). Similarly, if the current score is lower than the previous minimum, it increments the count of breaking the lowest score record (`ans[1]`). The final result is a vector (`ans`) containing the counts of breaking the highest and lowest score records, which is returned as the output of the function.

Q.15) Subarray Division
https://www.hackerrank.com/challenges/the-birthday-bar/problem?isFullScreen=true

Code :

```
```

```
/*
A -> [1,2,3,4,5] -> [2,3,4] or [3,4]

//length
[1]->1
2->1
3->1
4->1
5->1
1,2->2 .................vector<vector<int>>ds;
1,2,3->3
1,2,3,4->4
1,2,3,4,5->5
2,3->2 ..................
2,3,4->3
2,3,4,5->4
3,4->2 ....................
3,4,5->3
4,5 ->2 ................

vector<int>arr -> arr is a vector of type int :- means ->
vector<vector<int>>ds :- ds -> a vector which will store the
vector

subarray :- It is the part of the array in which elements are
present in contiguous manner & the relative order of element must
be maintained , n-> total subarray :- n*(n+1)/2 = 15 subarray

subseq :- It is the part of thr array in which elements may or
may not be in contiguous but the relative order must be
maintained

subset :- It is something which may or may not be contiguous & it
may or may not maintain the relative ordering of the elements

 */

int birthday(vector<int> s, int d, int m) {
vector<vector<int>>ds;

 for (int i=0; i<s.size(); i++) {
```

```
 for (int j=i; j<s.size(); j++) {
 vector<int>temp;
 for (int k=i; k<=j; k++) {
 temp.push_back(s[k]);

 }
 ds.push_back(temp);

 }
 }
 int cnt=0;

 for (int i=0; i<ds.size(); i++) {
 if (ds[i].size()==m) {
 int sum=0;
 for (int j=0; j<ds[i].size(); j++) {
 sum+=ds[i][j];

 }
 if(sum==d) cnt++;
 }
 }
 return cnt;
}
```

Description :
Sure, let's break down each line of code in the `birthday` function:

```cpp
int birthday(vector<int> s, int d, int m) {
```
- Function signature: `int birthday(vector<int> s, int d, int m)`.
- Parameters:
  - `s`: A vector of integers representing chocolate bar pieces.
  - `d`: An integer representing Ron's birthday chocolate sum.
  - `m`: An integer representing the length of the contiguous segment (subarray) of the chocolate bar.

```cpp
vector<vector<int>> ds;
```

- Definition of a 2D vector `ds` to store all possible contiguous subarrays.

```cpp
for (int i = 0; i < s.size(); i++) {
    for (int j = i; j < s.size(); j++) {
        vector<int> temp;
        for (int k = i; k <= j; k++) {
            temp.push_back(s[k]);
        }
        ds.push_back(temp);
    }
}
```
- Nested loops iterate over all possible subarrays of the given chocolate bar `s`.
- The outer loop (`i`) represents the starting index of the subarray.
- The middle loop (`j`) represents the ending index of the subarray.
- The inner loop (`k`) fills a temporary vector `temp` with elements from the current subarray.
- The temporary vector is then pushed into the 2D vector `ds`, capturing all possible contiguous subarrays.

```cpp
int cnt = 0;
```
- Initialize a variable `cnt` to count the number of valid subarrays.

```cpp
for (int i = 0; i < ds.size(); i++) {
    if (ds[i].size() == m) {
        int sum = 0;
        for (int j = 0; j < ds[i].size(); j++) {
            sum += ds[i][j];
        }
        if (sum == d) cnt++;
    }
}
```
- Loop through all the contiguous subarrays stored in `ds`.
- Check if the size of the current subarray (`ds[i].size()`) is equal to the specified length `m`.
- If the size matches, calculate the sum of elements in the subarray.
- If the sum matches the specified target sum `d`, increment the count `cnt`.

```cpp
return cnt;
```
- Return the count of valid contiguous subarrays that meet the given criteria.

The `birthday` function aims to solve a problem related to Ron's birthday celebration, where he receives a chocolate bar represented by a vector `s` containing integer values. The function takes three parameters:

1. `s`: A vector of integers representing the chocolate bar pieces.
2. `d`: An integer representing Ron's birthday chocolate sum.
3. `m`: An integer representing the length of the contiguous segment (subarray) of the chocolate bar.

Here's a step-by-step description of the code:

1. **Initialization of Variables:**
   ```cpp
   int cnt = 0;
   ```
   - Initialize a variable `cnt` to keep track of the count of valid subarrays that meet the specified criteria.

   ```cpp
   vector<vector<int>> ds;
   ```
   - Define a 2D vector `ds` to store all possible contiguous subarrays.

2. **Generate All Contiguous Subarrays:**
   ```cpp
   for (int i = 0; i < s.size(); i++) {
       for (int j = i; j < s.size(); j++) {
           vector<int> temp;
           for (int k = i; k <= j; k++) {
               temp.push_back(s[k]);
           }
           ds.push_back(temp);
       }
   }
   ```
   - Nested loops iterate over all possible subarrays of the given chocolate bar `s`.
   - The outer loop (`i`) represents the starting index of the subarray.
   - The middle loop (`j`) represents the ending index of the subarray.
   - The inner loop (`k`) fills a temporary vector `temp` with elements from the current subarray.
   - The temporary vector is then pushed into the 2D vector `ds`, capturing all possible contiguous subarrays.

3. **Count Valid Subarrays:**
   ```cpp
   for (int i = 0; i < ds.size(); i++) {
       if (ds[i].size() == m) {
```

```
        int sum = 0;
        for (int j = 0; j < ds[i].size(); j++) {
            sum += ds[i][j];
        }
        if (sum == d) cnt++;
    }
  }
  ```
  - Loop through all the contiguous subarrays stored in `ds`.
  - Check if the size of the current subarray (`ds[i].size()`) is equal to the specified length `m`.
  - If the size matches, calculate the sum of elements in the subarray.
  - If the sum matches the specified target sum `d`, increment the count `cnt`.

4. **Return Result:**
  ```cpp
  return cnt;
  ```
  - Return the count of valid contiguous subarrays that meet the given criteria.

In summary, the function generates all possible contiguous subarrays of the chocolate bar, checks the length and sum of each subarray, and returns the count of valid subarrays that meet the specified conditions.

Q.16) Divisible Sum Pair
https://www.hackerrank.com/challenges/divisible-sum-pairs/problem?isFullScreen=true

Code :

```cpp
int divisibleSumPairs(int n, int k, vector<int> ar) {
int cnt=0;
for (int i=0; i<ar.size(); i++) {
for (int j=i+1; j<ar.size(); j++) {
    if ((ar[i]+ar[j])%k==0) {
    cnt++;
    }
}
}
return cnt;
}
```

Description :

The `divisibleSumPairs` function is designed to find and count pairs of elements in an array (`ar`) such that their sum is divisible by a given integer (`k`). Here's a detailed description of the working of the function:

1. **Initialization:**
   - Initialize a counter variable (`cnt`) to keep track of the number of valid pairs.

2. **Nested Loop Iteration:**
   - Use two nested loops to iterate through all possible pairs of elements in the array.
   - The outer loop selects the first element (`ar[i]`), and the inner loop iterates over elements following the selected one (`ar[j]`).

3. **Pair Check:**
   - For each pair of elements selected by the loops, calculate their sum (`ar[i] + ar[j]`).
   - Check if the sum is divisible by the specified integer (`k`).
   - If the sum is divisible, increment the counter (`cnt`) as this pair satisfies the condition.

4. **Counting Valid Pairs:**
   - The counter (`cnt`) keeps track of the total number of pairs that meet the criteria of having a sum divisible by `k`.

5. **Result:**
   - Return the final count (`cnt`) as the result of the function.

In summary, the function systematically examines all pairs of elements in the array, checking if the sum of each pair is divisible by `k`. The count of pairs that satisfy this condition is then returned as the output of the function.

Q.17) Migratory Birds
https://www.hackerrank.com/challenges/migratory-birds/problem?isFullScreen=true

Code :

```
int migratoryBirds(vector<int> arr) {
// if we have to figure out the occurrence of each element then
how to do that ?

unordered_map<int, int>mp;
for (int i=0;i<arr.size(); i++) {
mp[arr[i]]++;



}
// i have to return the no. whose occurrence(or freq.) is max.
```

```
// if there are more than 1 no. whose occ. is same as that of
maxi then i will return the value which is min

int maxi=0;
for (auto it:mp) {
    maxi=max(maxi,it.second);


}
// maxi -> max freq. of a particular element
int ans=1e9;
for(auto it:mp){
    // for a element i'm checking whether the element is
occurring maxi times or not .

    if (it.second==maxi) {
    ans=min(ans,it.first);

    }
}
return ans;
}
```

Description :

The `migratoryBirds` function takes a vector of integers (`arr`) representing types of birds and aims to find the most common type of bird. In case of a tie, the function returns the smallest type number.

Here's a detailed description of the code:

1. **Initialization of Hash Map:**
   - Create an unordered map (`mp`) to store the occurrences of each bird type.
   - Iterate through the input vector (`arr`) and increment the count for each bird type in the map.

   ```cpp
   unordered_map<int, int> mp;
   for (int i = 0; i < arr.size(); i++) {
       mp[arr[i]]++;
   }
   ```

2. **Find Maximum Occurrence:**

- Iterate through the map to find the maximum occurrence of any bird type (`maxi`).

```cpp
int maxi = 0;
for (auto it : mp) {
    maxi = max(maxi, it.second);
}
```

3. **Find Minimum Bird Type with Maximum Occurrence:**
   - Initialize a variable (`ans`) with a large value (`1e9`).
   - Iterate through the map again, and for each bird type with occurrences equal to `maxi`, update `ans` with the minimum bird type number.

```cpp
int ans = 1e9;
for (auto it : mp) {
    if (it.second == maxi) {
        ans = min(ans, it.first);
    }
}
```

4. **Return Result:**
   - The variable `ans` now holds the type of bird that occurs most frequently and has the smallest type number.
   - Return `ans` as the output of the function.

```cpp
return ans;
```

In summary, the function uses an unordered map to count the occurrences of each bird type, finds the maximum occurrence, and then determines the bird type with the smallest number that has the maximum occurrence in case of a tie. The result is the type of bird that is most commonly observed.

Q.18) Day of the Programmer
https://www.hackerrank.com/challenges/day-of-the-programmer/problem?isFullScreen=true

Code :

```cpp
string dayOfProgrammer(int year) {
int tot_days_eight=0;
```

```
// Now if my current year is leap year then the only thing I have
to do is add 1 to tot_days_eight

//julian calender
if (year<1918){
tot_days_eight =243 ;//31+28+31+30+31+30+31+31=243

// leap year -> This is the only way to find leap year
 if (year%4==0)
tot_days_eight+=1;

}
// gregorian calender
else if (year==1918) {
//The transition from the Julian to Gregorian Calender system
occurred in 1918, when the next day after January 31st was
February 14th

tot_days_eight =230; //31+15+31+30+31+30+31+31=230
// if (year%4==0 && year%100!=0)
// tot_days_eight+=1;

}

else if (year>1918) {
tot_days_eight=243; //31+28+31+30+31+30+31+31 =243

if (year%400==0)
tot_days_eight+=1;

else if(year%4==0 && (year%100)!=0)
tot_days_eight+=1;

}
string ans="";
int day = 256 - tot_days_eight;
string d = to_string(day);
string y = to_string(year);

if (d.size()==2) {
ans+=d;
```

```
}
else {
ans+='0';
ans+=d;

}

ans+='.';
ans+="09.";
ans+=y;



return ans;
}
```

Description :

The `dayOfProgrammer` function is designed to determine the date of the Programmer's Day based on the given year. The problem involves accounting for the transition from the Julian to the Gregorian calendar in 1918. The function calculates the day and month for Programmer's Day and returns it as a string in the format "dd.mm.yyyy".

Here is a detailed description of the code:

1. **Initialization:**
   - Initialize a variable `tot_days_eight` to keep track of the total days until the Programmer's Day in the given year.

2. **Julian Calendar Calculation:**
   - If the given year is before 1918, it is considered under the Julian calendar.
     - Set `tot_days_eight` to 243, representing the total days until September 12th (31+28+31+30+31+30+31+31).
     - If the year is a leap year (divisible by 4), increment `tot_days_eight` by 1.

   ```cpp
   if (year < 1918) {
       tot_days_eight = 243;
       if (year % 4 == 0)
           tot_days_eight += 1;
   }
   ```

3. **Gregorian Calendar Calculation (Transition Year 1918):**

- If the year is 1918, it is the transition year from the Julian to the Gregorian calendar.
  - Set `tot_days_eight` to 230, representing the days until February 14th (31+15+31+30+31+30+31+31).
// Transitional year was 1918 , instead of 1st feb , feb starts from 14th feb , In 1918 the total no. of days was 14->28 and 1918 was not a leap year , 32nd day was 14th Feb in Gregorian

```cpp
else if (year == 1918) {
    tot_days_eight = 230;
}
```

4. **Gregorian Calendar Calculation (Years after 1918):**
- If the year is after 1918, it is under the Gregorian calendar.
  - Set `tot_days_eight` to 243, representing the total days until September 12th (31+28+31+30+31+30+31+31).
  - If the year is a leap year (divisible by 400 or divisible by 4 but not divisible by 100), increment `tot_days_eight` by 1.

```cpp
else if (year > 1918) {
    tot_days_eight = 243;
    if (year % 400 == 0)
        tot_days_eight += 1;
    else if (year % 4 == 0 && year % 100 != 0)
        tot_days_eight += 1;
}
```

5. **Date Calculation:**
- Calculate the day of the Programmer's Day by subtracting `tot_days_eight` from 256.
- Convert the day and year to strings (`d` and `y`).
- If the day is a single digit, add a leading zero to the result.

```cpp
int day = 256 - tot_days_eight;
string d = to_string(day);
string y = to_string(year);
if (d.size() == 2) {
    ans += d;
} else {
    ans += '0';
    ans += d;
}
```

6. **Result Formatting:**
- Format the result string (`ans`) in the "dd.mm.yyyy" format.

```cpp
ans += '.';
ans += "09.";
ans += y;
```

7. **Return Result:**
   - Return the final result string.

```cpp
return ans;
```

In summary, the function carefully calculates the total days based on the calendar system (Julian or Gregorian) and the specific year, then converts the result into the required date format for Programmer's Day.

Q.19) Bill Division
https://www.hackerrank.com/challenges/bon-appetit/problem?isFullScreen=true

Code :

```
void bonAppetit(vector<int> bill, int k, int b) {

int sum=0;
for (int i=0;i<bill.size(); i++) {
    if (i!=k) {
    sum+=bill[i];

    }
}
    int split = sum/2;

    if (split==b) {
    cout<<"Bon Appetit";
    }
    else{
        cout<<b-split;
    }
}
```

Description :

The `bonAppetit` function is designed to determine if the total cost of shared items in a restaurant bill is fairly split between two friends. It takes three parameters: a vector of integers (`bill`) representing the cost of each item ordered, an integer (`k`) representing the index of an item that Anna did not eat, and an integer (`b`) representing the amount Anna contributed.

Here is a detailed description of the code:

1. **Initialization:**
   - Initialize a variable `sum` to keep track of the total cost of shared items, starting from 0.

   ```cpp
   int sum = 0;
   ```

2. **Calculate Total Shared Cost:**
   - Iterate through the `bill` vector using a for loop.
   - Add the cost of each item to `sum`, excluding the item at index `k`.

   ```cpp
   for (int i = 0; i < bill.size(); i++) {
       if (i != k) {
           sum += bill[i];
       }
   }
   ```

3. **Calculate Split Cost:**
   - Calculate the equal split of the total cost by dividing `sum` by 2.

   ```cpp
   int split = sum / 2;
   ```

4. **Check Fair Split:**
   - Compare the calculated split (`split`) with the amount Anna contributed (`b`).
   - If they are equal, print "Bon Appetit" to indicate a fair split.
   - If they are not equal, print the difference between `b` and `split` to indicate the amount by which Anna was overcharged.

   ```cpp
   if (split == b) {
       cout << "Bon Appetit";
   } else {
       cout << b - split;
   }
   ```

```

In summary, the function iterates through the items on the bill, excluding the one Anna did not eat, calculates the total shared cost, determines the equal split, and checks whether Anna was overcharged or not. The function then outputs the result accordingly.

Q.20) Counting Valley
https://www.hackerrank.com/challenges/counting-valleys/problem?isFullScreen=true

Code :

```
int countingValleys(int steps, string path) {

int sea_level=0;
int cnt=0;

for (int i=0; i<steps; i++) {
 if (path[i]=='U') {
  sea_level++;
 }
 else{
     sea_level--;
 }

 if (sea_level==0 && path[i]=='U') {
  cnt++;
 }

}
return cnt;

}
```

Description :

The `countingValleys` function is designed to determine the number of valleys a hiker traverses based on a given sequence of steps. It takes two parameters: an integer `steps`
```

representing the total number of steps in the hike and a string `path` representing the sequence of steps taken.

Here's a detailed description of the code:

1. **Initialization:**
   - Initialize two variables: `sea_level` to keep track of the current altitude relative to sea level and `cnt` to count the number of valleys traversed.

   ```cpp
   int sea_level = 0;
   int cnt = 0;
   ```

2. **Iterate through Steps:**
   - Use a for loop to iterate through each step in the given path.
   - Check the direction of the step (`U` for uphill or `D` for downhill).
   - Adjust the `sea_level` accordingly:
     - If the step is uphill (`'U'`), increment `sea_level`.
     - If the step is downhill (`'D'`), decrement `sea_level`.

   ```cpp
   for (int i = 0; i < steps; i++) {
       if (path[i] == 'U') {
           sea_level++;
       } else {
           sea_level--;
       }
   }
   ```

3. **Count Valleys:**
   - Within the loop, check if the hiker returns to sea level (`sea_level == 0`) and the step is uphill (`'U'`).
   - If the condition is met, increment the `cnt` variable to count the traversal of a valley.

   ```cpp
   if (sea_level == 0 && path[i] == 'U') {
       cnt++;
   }
   ```

4. **Return Result:**
   - After the loop, return the final count of valleys (`cnt`).

   ```cpp
   return cnt;
   ```

In summary, the function simulates a hike by iterating through each step, keeping track of the hiker's altitude relative to sea level. It counts the number of times the hiker returns to sea level from a downhill path, indicating the traversal of a valley. The final count of valleys is returned as the result.

Q.21) Electronics Shop
https://www.hackerrank.com/challenges/electronics-shop/problem?isFullScreen=true

Code:

```
int getMoneySpent(vector<int> keyboards, vector<int> drives, int
b) {

    int m=keyboards.size();
    int n= drives.size();
    int maxi=-1;

    for (int i=0;i<m; i++) {
        for (int j=0; j<n; j++) {
        int sum= keyboards[i]+drives[j];
        if (sum<=b) {
        maxi = max(maxi,sum);
        }
        }
    }

return maxi;
}
```

Description :
The C++ code defines a function `getMoneySpent` that takes two vectors of keyboard and drive prices, along with a budget. The function iterates through all possible combinations, calculates the total price, and keeps track of the maximum affordable amount. The maximum amount is then returned. Note that this approach is simple but may not be the most efficient for large inputs.

Q.22) Cats and a Mouse
https://www.hackerrank.com/challenges/cats-and-a-mouse/problem?isFullScreen=true

Code :

```
vector<string> split_string(string);

// Complete the catAndMouse function below.
string catAndMouse(int x, int y, int z) {
string ans="";

int diff1=abs(x-z);
int diff2=abs(y-z);

if (diff1==diff2) {
ans="Mouse C";
}

else{


if (diff1<diff2) {
ans="Cat A";
}

else
ans="Cat B";
}
return ans;
}
```

Description :

This C++ code defines a function catAndMouse that takes three integers (x, y, and z) representing the positions of two cats (Cat A at position x and Cat B at position y) and a mouse at position z. The function calculates the distances of each cat from the mouse and determines which cat will catch the mouse first or if both cats will catch it simultaneously.

Q.23) Picking Numbers
https://www.hackerrank.com/challenges/picking-numbers/problem?isFullScreen=true

Code :

```
int pickingNumbers(vector<int> a) {
sort(a.begin(),a.end());

int maxi=1;
for (int i=0; i<a.size(); i++) {
    int first=a[i];
for (int j=i+1; j<a.size(); j++) {
int second=a[j];

if (second-first<=1) {
maxi=max(maxi,j-i+1);
}
else{
    break; // agar j usko point karega jo min and max ke beech ka
differnece is >1
}
}
}
return maxi;
}
```

Description :

The given C++ code defines a function pickingNumbers that takes a vector of integers a. The function aims to find the length of the longest subarray such that the absolute difference between any two elements in the subarray is at most 1

Q.24) Designer PDF Viewer
https://www.hackerrank.com/challenges/designer-pdf-viewer/problem?isFullScreen=true

Code :

```
int designerPdfViewer(vector<int> h, string word) {
```

```
    // use map when yoou have access of characters and use vector when you have access of index like if word is "2345" then use vector

unordered_map<char, int > mp;
char ch = 'a';

for (int i=0; i<26; i++) {
mp[ch]=i;
ch++;
}

int width  = word.size();
int max_height = 0;
for (int i=0; i<word.size(); i++) {
int idx=mp[word[i]];
int height = h[idx];

max_height = max(max_height,height);

}
return max_height*width;
}
```

Description :
The `designerPdfViewer` function calculates the area of the rectangle formed by a given word when displayed in a specified font. It takes a vector of integer heights (`h`) corresponding to each letter of the alphabet and a string (`word`). The function uses an unordered map to assign indices to letters, then iterates through the word to find the maximum height and calculates the area as the product of the maximum height and the word's length. The result is returned as an integer.

Q.25) Utopian Tree
https://www.hackerrank.com/challenges/utopian-tree/problem?isFullScreen=true

Code:

```
int utopianTree(int n) {
    int height=1;
    // initially , sapling of height 1 is planted in the ground
```

```
    //starts with spring and Summer

    for (int i=1; i<=n; i++) {
    // if i->odd ->spring -> double in height
    // if i ->even -> summer -> only increased by 1

    if (i%2!=0) {
    height*=2;

    }
    else{
        if (i%2==0) {
        height++;
        }
    }
    }
    return height;
}
```

Description :

The given code defines a function named `utopianTree` that takes an integer parameter `n` representing the number of growth cycles and returns the height of a special type of tree after the specified number of cycles. The tree undergoes alternating growth patterns during spring and summer cycles.

Here's a breakdown of the code:

1. `int height = 1;`: Initializes a variable `height` to 1, representing the initial height of the tree.

2. `for (int i = 1; i <= n; i++) {`: Iterates through growth cycles from 1 to `n`.

3. `if (i % 2 != 0) { height *= 2; }`: If the current cycle is odd (spring), the tree's height is doubled.

4. `else { if (i % 2 == 0) { height++; } }`: If the current cycle is even (summer), the tree's height is increased by 1.

5. `return height;`: Returns the final height of the tree after the specified number of growth cycles.

In summary, the function simulates the growth of a special tree, where its height doubles during spring cycles and increases by 1 during summer cycles. The process is repeated for the specified number of growth cycles (`n`), and the final height of the tree is returned.

Q.26) Circular Rotation
https://www.hackerrank.com/challenges/circular-array-rotation/problem?isFullScreen=true

Code :

```
/*

a -> 1 2 3 4 5 6 7 8 9

k=1 -> 9 1 2 3 4 5 6 7 8

k=2 -> 8 9 1 2 3 4 5 6 7

k=3 -> 7 8 9 1 2 3 4 5 6

k=4 -> 6 7 8 9 1 2 3 4 5

 q -> 0 5 6

 6, 2, 3

*/

vector<int> circularArrayRotation(vector<int> a, int k,
vector<int> queries) {
 if (k>=a.size()) {
 // Actual rotation
 while (k>=a.size()) {
 k=k-a.size();
 }

 }
 vector<int> ans;
 vector<int>ip;

 for (int i=a.size()-k; i<a.size(); i++) {
 ip.push_back(a[i]);
```

```
 }
 for (int i=0; i<a.size()-k; i++) {
 ip.push_back(a[i]);
 }
 for (int i=0;i<queries.size() ; i++) {
 int idx = queries[i];
 ans.push_back(ip[idx]);
 }
 return ans;
}
```

Description :

The `circularArrayRotation` function performs circular rotations on an input array `a` and then returns the values at specified indices (`queries`) after the rotations. It takes the array, the number of rotations (`k`), and a vector of query indices.

The function handles cases where the number of rotations (`k`) is greater than or equal to the array size by adjusting `k` to avoid unnecessary rotations. It then creates a new array (`ip`) by combining the rotated and non-rotated portions.

Finally, the function iterates through the query indices, retrieves the corresponding values from the rotated array, and returns them in a vector (`ans`).

Q.27) Angry Professor
https://www.hackerrank.com/challenges/angry-professor/problem?isFullScreen=true

Code :

```
string angryProfessor(int k, vector<int> a) {
int cnt=0;
// print yes if class will get cancelled otherwise no

string ans="";

for (int i=0;i<a.size(); i++) {
    if (a[i]<=0) {
    cnt++;
    }
}
if (cnt>=k) {
```

```
ans="NO";
}
else{
    ans="YES";
}
return ans;
}
```

Description :

The given code defines a function named `angryProfessor` that takes two parameters: an integer `k` representing the minimum number of students required for the class not to be canceled and a vector of integers `a` representing the arrival times of students. The function returns a string "YES" if the class is not canceled and "NO" otherwise.

Here's a breakdown of the code:

1. `int cnt = 0;`: Initializes a variable `cnt` to zero, which will be used to count the number of students who arrive on time or early.

2. `string ans = "";`: Initializes an empty string `ans` to store the final result ("YES" or "NO").

3. `for (int i = 0; i < a.size(); i++) {`: Iterates through the vector `a` to check each student's arrival time.

4. `if (a[i] <= 0) { cnt++; }`: If the arrival time of a student is less than or equal to zero, increment the counter `cnt`. This means the student arrived on time or early.

5. `if (cnt >= k) { ans = "NO"; }`: If the number of students who arrived on time or early is greater than or equal to the minimum required (`k`), set `ans` to "NO" (class will be canceled).

6. `else { ans = "YES"; }`: If the number of students who arrived on time or early is less than the minimum required (`k`), set `ans` to "YES" (class will not be canceled).

7. `return ans;`: Returns the final result indicating whether the class is canceled ("NO") or not ("YES").

In summary, the function determines whether the class will be canceled based on the minimum required students (`k`) and the arrival times of the students. The result is returned as a string ("YES" or "NO").

Q.28) Sequence Equation

https://www.hackerrank.com/challenges/permutation-equation/problem?isFullScreen=true

Code :

```
vector<int> permutationEquation(vector<int> p) {
vector<int> ans;
// for number I need indexes


//          no. idx (1-based)
unordered_map<int, int> mp ;
for (int i=0; i<p.size(); i++) {
mp[p[i]] = i+1;

}
int n =p.size();
for(int i=1;i<=n;i++){
    int x=i;
    int idx=mp[x];

    int idx2=mp[idx];

    ans.push_back(idx2);
}
return ans;
}
```

Description :

The `permutationEquation` function takes a vector `p` representing a permutation and returns a new vector containing the results of applying a specific mathematical operation to the elements of the permutation. The operation involves finding the indices of certain elements in the permutation.

Here's a breakdown:

1. Create an unordered map (`mp`) to store the mapping of numbers to their corresponding indices in the permutation.

2. Populate the map by iterating through the elements of the permutation and storing each element along with its index (1-based) in the map.

3. Iterate through the numbers from 1 to the size of the permutation (`n`), and for each number `x`, find its index in the original permutation (`idx`). Then, find the index of `idx` itself in the original permutation (`idx2`).

4. Push the result (`idx2`) into the answer vector (`ans`).

5. Return the resulting vector (`ans`).

In summary, the function generates a new vector based on a specific mathematical operation involving the indices of elements in a permutation.

Q.29) Find Digits

https://www.hackerrank.com/challenges/find-digits/problem?isFullScreen=true

Code :

```
int findDigits(int n) {
int temp=n;

int cnt=0; // store the count of divisor which divides the n
while (n>0) {
int ld=n%10;

if(ld!=0 && temp%ld==0) cnt++; // is it possible for n/0 ? no !!
    n/=10;

}
return cnt;

}
```

Description :

The `findDigits` function takes an integer `n` and counts the number of digits in `n` that evenly divide `n`. It initializes a counter (`cnt`) to zero and iterates through the digits of `n`. For each digit (`ld`), it checks if the digit is not zero and if `n` is divisible evenly by that digit. If both conditions are met, the counter is incremented. The function then returns the count of such digits. The function is designed to handle the case where the digit being checked is not zero to avoid division by zero.

Q.30) The Hurdle Race

https://www.hackerrank.com/challenges/the-hurdle-race/problem?isFullScreen=true

Code :

```
int hurdleRace(int k, vector<int> height) {
int maxi=INT_MIN;
for (int i=0;i<height.size() ; i++) {
maxi = max(maxi,height[i]);

}
int ans = maxi-k;

if (k>=maxi) {
return ans= 0;
}
else{

return ans;
}


}
```

Description :

The `hurdleRace` function takes an integer `k` representing the jumping ability of a character and a vector `height` representing the heights of hurdles. The function finds the highest hurdle (`maxi`) and calculates the difference between the character's jumping ability and the highest hurdle. If the jumping ability is greater than or equal to the highest hurdle, the function returns 0; otherwise, it returns the calculated difference.

Here's a summary:

1. Find the maximum height of hurdles (`maxi`) in the given vector.

2. Calculate the difference between the character's jumping ability (`k`) and the highest hurdle (`maxi`).

3. If the jumping ability is greater than or equal to the highest hurdle, return 0; otherwise, return the calculated difference.

In essence, the function determines the amount by which the character needs to increase their jumping ability to clear the highest hurdle, or returns 0 if the character can already clear all hurdles.

Q.31) Cut the Sticks
https://www.hackerrank.com/challenges/cut-the-sticks/problem?isFullScreen=true

Code :

```
vector<int> cutTheSticks(vector<int> arr) {


vector<int>ans;
int mini = *min_element(arr.begin(),arr.end());

while (1) {

    int cnt=0; // will tell how many sticks are cut this time

    int cut=mini; // we will update the value of mini after
cutting each stick for next iteration

    mini =1e9; //because we are finding the mini for the next
iteration

    for (int j=0; j<arr.size(); j++) {
        if (arr[j]!=1e9) {
        cnt++; // we have cut the stick

        arr[j]-=cut;

        if (arr[j]==0) {
        arr[j] =1e9;
        }

        }
        mini = min(mini,arr[j]); // for the next iteration
```

```
        // we didnt use complex technique becuase it will give
O(n sqare but we used this which gives us the o(1))
    }


if (cnt!=0) { // few sticks are cut this time
ans.push_back(cnt);
}
else{
    break;
}
}
return ans;


}
```

Description :

The given code defines a function named `cutTheSticks` that takes a vector of integers `arr` representing the lengths of sticks and returns a vector of integers `ans`. The `ans` vector contains the count of sticks remaining after each round of cutting.

Here's a breakdown of the code:

1. `vector<int> ans;`: Initializes an empty vector `ans` to store the count of sticks remaining after each round.

2. `int mini = *min_element(arr.begin(), arr.end());`: Finds the minimum value in the initial vector `arr` and stores it in the variable `mini`.

3. `while (1) {`: Initiates an infinite loop to perform cutting rounds until all sticks are completely cut.

4. Inside the loop:
   - `int cnt = 0;`: Initializes a variable `cnt` to zero, which will be used to count how many sticks are cut in each round.
   - `int cut = mini;`: Initializes a variable `cut` with the current minimum stick length. This value will be used to cut the sticks in the current round.
   - `mini = 1e9;`: Resets the `mini` variable to a large value (1e9) for the next iteration.

5. Nested loop:
   - Iterates through the vector `arr` to cut sticks and update their lengths.
   - If the current stick length is not marked as already cut (`arr[j] != 1e9`):

- Increments the counter `cnt` to indicate that a stick is cut.
- Reduces the length of the stick by the value of `cut`.
- If the stick length becomes zero, marks it as cut by setting its value to 1e9.

- Updates the `mini` variable by finding the minimum stick length encountered in the current iteration.

6. `if (cnt != 0) { ans.push_back(cnt); }`: Checks if any sticks were cut in the current round. If yes, appends the count of cut sticks to the `ans` vector.

7. `else { break; }`: If no sticks were cut in the current round, breaks out of the infinite loop.

8. `return ans;`: Returns the vector `ans` containing the count of sticks remaining after each cutting round.

In summary, the function simulates the process of cutting sticks in each round, updating their lengths and counting the remaining sticks after each round. The result is stored in the `ans` vector and returned.

Q.32) Beautiful Days at the movie
https://www.hackerrank.com/challenges/beautiful-days-at-the-movies/problem?isFullScreen=true

Code :

```
int beautifulDays(int i, int j, int k) {


int cnt=0;

for(int st=i;st<=j;st++){
    int num=st;
    int rev=0;


    //reverse the number
    int temp=num;
    while (temp>0) {
    int ld=temp%10;
    rev=rev*10+ld; // formula to reverse the digits
    temp/=10;
```

```
    }
    int diff = abs(rev-num);
    if (diff%k==0) {
    cnt++;
    }
}
    return cnt;
}
```

Description :

The given code defines a function named `beautifulDays` that takes three integers `i`, `j`, and `k`. The function calculates and returns the count of "beautiful days" within the inclusive range `[i, j]`. A day is considered beautiful if the absolute difference between the original number and its reverse is divisible by `k`.

Here's a breakdown of the code:

1. `int cnt = 0;`: Initializes a variable `cnt` to zero. This variable will be used to count the number of beautiful days.

2. `for(int st = i; st <= j; st++) {`: Iterates through the range of days from `i` to `j`.

3. Inside the loop:
   - `int num = st;`: Initializes a variable `num` with the current day's number.
   - `int rev = 0;`: Initializes a variable `rev` to store the reversed number.

4. Reverse the number:
   - `int temp = num;`: Creates a temporary variable to store the original number.
   - A `while` loop is used to reverse the digits of the number:
     - `int ld = temp % 10;`: Extracts the last digit of the number.
     - `rev = rev * 10 + ld;`: Builds the reversed number.
     - `temp /= 10;`: Removes the last digit from the original number.

5. `int diff = abs(rev - num);`: Calculates the absolute difference between the original number and its reverse.

6. `if (diff % k == 0) { cnt++; }`: Checks if the absolute difference is divisible by `k`. If true, increments the counter `cnt`.

7. `return cnt;`: Returns the final count of beautiful days within the specified range `[i, j]`.

In summary, the function iterates through each day in the range `[i, j]`, calculates the absolute difference between the original number and its reverse, and increments the counter if the difference is divisible by `k`. The final count is returned as the result.

Q.33) Modified Kaprekar Number

https://www.hackerrank.com/challenges/kaprekar-numbers/problem?isFullScreen=true

Code :

```
/*
0<p<q<10^5

cant think of O(n^2)


while(d1=0) is not recommended


use

string s = "abcdef"

s.substr(0,4); ->start,end  .. start is inclusive and end is
exclusive means [start,end)

s.subustr(3) -> start -> [start,end]
//in this version def will be print

so this is the 2 version of substr


for single digit 1 -> 1 digit ka sqaured number ayega , so there
won't be right part , only left part


If I square a number , if the squared number is single digit then
I will have only left part   not right part

 */

void kaprekarNumbers(int p, int q) {
bool flag=0;

for (int i=p; i<=q; i++) {
```

```
string s = to_string(i);

int digits = s.size(); // d digits

long long squared = (long long) i * (long long )i;

string s1= to_string(squared);

string leftpart ="",rightpart="";


if (s1.size()==1) {
leftpart=s1;
}
else{
    leftpart = s1.substr(0,s1.size()-digits);
    rightpart= s1.substr(s1.size()-digits);
    // suppose 16 ka 256 aya and n=3
    // then   n-d => 3-2=1 means 1
}

if (rightpart=="") { // squared value is of single digit
long long left = stoi(leftpart);

if (left==i) {
cout<<i<<" ";
flag=1;

}
}
else{
    long long left = stoi(leftpart);
    long long right = stoi(rightpart);

    if (left+right==i) {
    cout<<i<<" ";
    flag=1;

    }
}
```

```
}

if (flag==0) {
cout<<"INVALID RANGE";

}
}
```

Description :

This code defines a function `kaprekarNumbers` that takes two integers `p` and `q` as input, representing a range of numbers. The function finds and prints Kaprekar numbers within that range.

A Kaprekar number is a non-negative integer, the representation of whose square can be split into two parts that add up to the original number. For example, 45 is a Kaprekar number because 45^2 = 2025, and 20 + 25 = 45.

The code iterates through the range from `p` to `q`, converts each number to its squared form, and checks if it is a Kaprekar number. It uses string manipulation to split the squared number into left and right parts based on the number of digits. If the right part is empty, it means the squared number is a single digit, and only the left part needs to be considered.

The function prints the Kaprekar numbers found in the specified range and outputs "INVALID RANGE" if no Kaprekar numbers are found. The code emphasizes avoiding an O(n^2) solution and demonstrates the use of the `substr` function for string manipulation.

Q.34) Cavity Map

https://www.hackerrank.com/challenges/cavity-map/problem?isFullScreen=true

Code : /*

```
    1. Cell has cavity if not in border line
    2. Adjacent cells should be strictly smaller in depth


    declare 2-d array

    vector<vector<int>>>grid;
    vector<string>grid;
```

```
    All the cells at 0th row and last row are border cells
    All the cells at 0th column and last column are border cells

    So we do not have to check
 */

vector<string> cavityMap(vector<string> grid) {
int m=grid.size();
int n= grid[0].size();

vector<string>ans = grid;

// start i and j from 1 because borders ko include nahi karna
hota hai
for (int i=1; i<m-1; i++) {
for (int j=1; j<n-1; j++) {

//top                              down                        left
if (grid[i-1][j]<grid[i][j] && grid[i+1][j]<grid[i][j]
&&grid[i][j-1]<grid[i][j] && grid[i][j+1]<grid[i][j])  {
ans[i][j]='X';
}

}
}
return ans;


}
```

Description :

This C++ code defines a function `cavityMap` that takes a vector of strings representing a 2D grid as input and returns a modified grid based on specified conditions. The function aims to identify and mark cells within the grid that have a cavity, satisfying two conditions:

1. The cell is not located on the border of the grid.
2. The depths of the adjacent cells (top, down, left, right) are strictly smaller than the depth of the current cell.

The function iterates through the interior cells of the grid and checks the specified conditions for each cell. If both conditions are met, the corresponding cell in the result grid (`ans`) is marked with 'X'. The modified grid is then returned.

Note: The provided code assumes that the input `grid` is a vector of strings and also declares `grid` as a 2D vector of integers, which might lead to confusion. Additionally, there is a comment mentioning the declaration of `vector<vector<int>> grid;`, but it is commented out, and the code uses the input vector of strings directly.

Q.35) Challenge 1 - Intro to Tutorial Challenges

https://www.hackerrank.com/challenges/tutorial-intro/problem?isFullScreen=true

Code :

```
int introTutorial(int V, vector<int> arr) {
    int ans;
for(int i=0;i<arr.size();i++){
    if (V==arr[i]) {
        ans =i;
        break;
    }
}
return ans;
}
```

Description :

The introTutorial function takes an integer V and a vector arr. It searches for the value V within the vector and returns the index of the first occurrence. The function iterates through the vector elements and stops when it finds a match. The index of the matching element is stored in the variable ans, and this index is returned.

Q.36) Challenge 2 - Insertion Sort Part 1

https://www.hackerrank.com/challenges/insertionsort1/problem?isFullScreen=true

Code :

```
void insertionSort1(int n, vector<int> a) {
    int num = a[n-1];

    bool flag =0;
    for (int i=n-2;i>=0 ; i--) {
    if (a[i]>num) {
    a[i+1]=a[i];
    }
    else if (a[i]<num) {
    a[i+1]=num;
    flag=1;

    }
    for (int i=0; i<n; i++) {
    cout<<a[i]<<" ";

    }
    cout<<endl;

    if (flag==1)
    break;


if (i==0 && flag==0) {
    a[0]=num;

    for(int j=0;j<n;j++)cout<<a[j]<<" ";

    }
}
}
```

Description :

The function insertionSort1 takes two parameters - n representing the size of the vector a and a which is a vector containing the elements to be sorted.
It initializes a variable num with the last element of the vector a.
A boolean flag variable flag is set to 0 to indicate whether the current element (num) has been correctly inserted into its sorted position.

The function enters a loop that iterates through the elements of the vector in reverse order, starting from the second-to-last element (n-2) and moving towards the first element (0).
Inside the loop, it compares the current element with num:

- If the current element is greater than num, it shifts the element to the right (a[i+1] = a[i]).
- If the current element is less than num, it inserts num at the correct position, sets the flag to 1, and breaks out of the loop.

After each iteration of the loop, it prints the current state of the vector.
Following the loop, there is a check: if the flag is set to 1, it breaks out of the loop. If not, it means that num is the smallest element, and it sets the first element of the vector to num. The vector is then printed.

Q.37 Challenge 3 - Insertion Sort Part 2

https://www.hackerrank.com/challenges/insertionsort2/problem?isFullScreen=true

Code :

```
void insertionSort2(int n, vector<int> arr) {
   for (int i = 1; i < n; i++)
    {
       for (int j = i; j > 0; j--)
       {
        if (arr[j]<arr[j-1])
        {
            swap(arr[j],arr[j-1]);
        }
        else{
            break;
        }

       }
        for (int i = 0; i < n; i++)
    {
        cout<<arr[i]<<" ";
    }
    cout<<endl;
```

```
    }
}
```

Description :

The function insertionSort2 takes two parameters - n representing the size of the vector arr and arr, which is a vector containing the elements to be sorted.
The function uses a nested loop structure. The outer loop iterates from the second element (index 1) to the last element of the vector.
The inner loop iterates backward from the current outer loop index (i) to the first element (index 0). This loop compares adjacent elements and swaps them if the element on the left is smaller than the element on the right.
Inside the inner loop, there is a conditional statement (if) that checks if the element at index j is less than the element at index j-1. If true, a swap is performed using the swap function.
If the else condition is triggered, indicating that the elements are in sorted order, the inner loop breaks.
After each iteration of the inner loop, the current state of the vector is printed.
The outer loop continues until all elements are processed, resulting in a fully sorted vector.

Q.38) Challenge 4 -Running Time of Algorithm

https://www.hackerrank.com/challenges/runningtime/problem?isFullScreen=true

Code :

```
int runningTime(vector<int> arr) {
   int n = arr.size();
   int cnt=0;
for (int i = 1; i < n; i++)
    {
       for (int j = i; j > 0; j--)
       {
        if (arr[j]<arr[j-1])
        {
            swap(arr[j],arr[j-1]);
        }
        else{
            break;
        }
```

```
            cnt++;

        }

    }
    return cnt;
}
```

Description :

The function runningTime takes a single parameter - arr, which is a vector containing the elements to be sorted.
The function initializes an integer variable n to store the size of the input vector arr and a counter variable cnt to keep track of the number of operations performed during the sorting process.
The function uses a nested loop structure, where the outer loop iterates from the second element (index 1) to the last element of the vector.
The inner loop iterates backward from the current outer loop index (i) to the first element (index 0). This loop compares adjacent elements and swaps them if the element on the left is smaller than the element on the right.
Inside the inner loop, there is a conditional statement (if) that checks if the element at index j is less than the element at index j-1. If true, a swap is performed using the swap function.
If the else condition is triggered, indicating that the elements are in sorted order, the inner loop breaks.
After each successful swap, the counter cnt is incremented to keep track of the number of operations.
The function returns the final count of operations, representing the running time of the insertion sort algorithm on the input vector.

Q.39) Find the Median

https://www.hackerrank.com/challenges/find-the-median/problem?isFullScreen=true

Code :

```
/*

median -> middle element

whenever array is sorted then only I can find median
```

list -> odd or even elements

1 2 3 4 5 n->5 odd

1 2 3 4 5 6 n->6 -> even array size so 3+4 = 7 and 7/2 is 3 in cpp

int mid = (i+j)/ 2

int mid = i+(j-i)/2; more preferred because

i+ j , it will be stored in memory with data type int but what if it exceeds the range on INTEGER

so here in i+(j-i)

(j-i) this is solved first then

then i+v1 (v1 is subtraction of j-i);
then i+v1/2;

Now it is in range

i+v1 value contains <= 10 digits not more than that
 */

```
int findMedian(vector<int> a) {

    sort(a.begin(),a.end());
int  n = a.size();

int i=0,j=n-1;

int mid = i+(j-i)/2;

return a[mid];
}
```

Description :

The `findMedian` function calculates the median of a given vector `a` by sorting it and then determining the middle element. It uses the formula `i + (j - i) / 2` to find the midpoint index in a safe manner, avoiding integer overflow. The function returns the calculated median.

Here's a concise summary:

1. Sort the input vector `a`.

2. Calculate the midpoint index using `i + (j - i) / 2`, ensuring safe computation to avoid integer overflow.

3. Return the element at the calculated midpoint index as the median.

Q.40) Closest Numbers

https://www.hackerrank.com/challenges/closest-numbers/problem?isFullScreen=true

Code :

```
vector<int> closestNumbers(vector<int> arr) {

// // diff elements

// unordered_map<int, vector<int>>mp;

// for (int i=0; i<arr.size(); i++) {
// for (int j=i+1; j<arr.size(); j++) {

// mp[abs(arr[j]-arr[i])].push_back(arr[i]);
// mp[abs(arr[j]-arr[i])].push_back(arr[j]);
// }
// }
// int mini =1e9;
// // vector<int>ans;
// for (auto it:mp) {
// mini = min(mini,it.first);
// }
// // for(auto it:mp){
// //     if (it.first==mini) {
```

```
// //     ans=it.second;
// //     break;
// //     }
// // }
// // or
// vector<int >ans=mp[mini];
// sort(ans.begin(),ans.end());

// return ans;

// we get an error , so sort the array before
 unordered_map<int, vector<int>>mp;
 sort(arr.begin(),arr.end());
 for (int i=0; i<arr.size()-1; i++) {
//  i<arr.size()-1 because hame pairs se deal karna hai naa isliye


int diff = arr[i+1]-arr[i];
mp[diff].push_back(arr[i+1]);
mp[diff].push_back(arr[i]);



 }
 int mini =1e9;
// vector<int>ans;
for (auto it:mp) {
mini = min(mini,it.first);
}
// for(auto it:mp){
//     if (it.first==mini) {
//     ans=it.second;
//     break;
//     }
// }
// or
// vector<int >ans=mp[mini];
sort(mp[mini].begin(),mp[mini].end());

return mp[mini];
```

```
//T.C IS O(n)
}
```

Description :

The `closestNumbers` function takes a vector `arr` and finds the pair of elements with the smallest difference. It uses an unordered map (`mp`) to store the differences and corresponding pairs. The array is sorted first to simplify the process of finding the closest numbers. The function returns a vector containing the pair(s) with the smallest difference.

Here's a concise summary:

1. Sort the input array `arr`.

2. Iterate through the sorted array and calculate the difference between adjacent elements, storing the differences and corresponding pairs in the unordered map (`mp`).

3. Find the minimum difference (`mini`) among the calculated differences.

4. Return the pair(s) with the minimum difference by accessing the corresponding vector in the unordered map.

Note: The code includes both the commented-out version and the corrected version. The corrected version sorts the array before finding the differences, improving efficiency.

Q.41) Mark and Toys

https://www.hackerrank.com/challenges/mark-and-toys/problem?isFullScreen=true

Code :

```
int maximumToys(vector<int> prices, int k) {
sort(prices.begin(),prices.end());
int cnt=0; // total no. of toys

for (int i=0; i<prices.size(); i++) {
    if (k>=prices[i]) {
    cnt++;
    k-=prices[i];
    }
    else{
        break;
    }
```

```
}
return cnt;
}
```

Description :

The maximumToys function takes a vector of toy prices (prices) and an integer budget (k). It aims to determine the maximum number of toys that can be bought with the given budget. The function sorts the toy prices in ascending order and iterates through the sorted prices, incrementing the toy count (cnt) and subtracting the price from the budget (k) until the budget is exhausted. The function then returns the total count of toys that were bought within the budget.

Q.42) Sherlock and Arrays

https://www.hackerrank.com/challenges/sherlock-and-array/problem?isFullScreen=true

Code :

```
/*
 Concept -> Prefix Sum -> used to reduce the time complexity

arr :- ith -> can be idx for which s1==s2

s1 -> for(0 to i-1)
s2 -> for (i+1 -> n-1)

if(s1==s2) break; yes


 now use better approach


 idx -> 0 1 2 3 4
 A   -> 1 2 3 4 5

 someone says find arr[1] to arr[3]

 time complexity using bruteforce us O(n)
```

```
 What if I tell you to find sum of elements of particular range in
O(1) T.C


 prefix sum means storing sum of 0th till ith idx in some variable
and we will  do for each index

  idx -> 0 1 2 3 4
 A    -> 1 2 3 4 5        sum=0

 i=0 -> sum=1
 i=1 -> sum=3

.
.
 i=4 -> sum = 15


 vector<int> prefix ; --> prefix[i] -> sum of the elements from
0th idx till ith idx.

 prefix -> 1 3 6 10 15

 remember :- size of prefix array == input array
 st ed
 [1 3] -> O(1)

 if (st-1<0){  agar hume [0,4] bola agar st-1 <0 it means it is
not valid index naa
     ans = prefix[ed];
 }
 else

 ans = prefix[ed]-prefix[st-1]
 // common sense , agar hame o to ed diya hai and st se leke ed ka
sum nikalna hai then ed-st-1 karunaga naa mere bhai
 10 - 1 = 9 // I got the sum w/o using for loop
 */

string balancedSums(vector<int> a) {
int n = a.size();
```

```
vector<int>prefix(n,0);
prefix[0]=a[0];
for (int i=1; i<n; i++) {
prefix[i]=a[i]+prefix[i-1];

}
 bool flag=0; // flag==1 when i get the s1==s2
 for (int i=0; i<n; i++) { //index for which s1 can be equal to s2
 int left=0,right=0;
 if (i==0) {
 left=0;
 right=prefix[n-1] - prefix[i];

 }
 else
    if (i==n-1) {
    right=0;
    left = prefix[i-1];
    }
    else{
        left = prefix[i-1];
        right=prefix[n-1]-prefix [i];
    }
if (left==right) {
     flag=1;
      break;
    }

 }

if(flag==1)
return "YES";

 return "NO";
}
```

Description :

The `balancedSums` function checks whether there exists an index in the given vector `a` such that the sum of elements on its left is equal to the sum of elements on its right. The

function uses the prefix sum technique to efficiently calculate cumulative sums up to each index, reducing the time complexity to O(1) for sum calculations within a range. It iterates through possible indices, comparing the left and right sums. If a balanced index is found, the function returns "YES"; otherwise, it returns "NO."

Here's a concise summary:

1. Calculate the prefix sum array (`prefix`) to efficiently compute cumulative sums up to each index.

2. Iterate through possible indices, comparing the left and right sums.

3. If a balanced index is found, return "YES"; otherwise, return "NO."

The function leverages the concept of prefix sums to optimize the calculation of cumulative sums within a given range.

Q.43) Jim and the Orders
https://www.hackerrank.com/challenges/jim-and-the-orders/problem?isFullScreen=true

Code :

```
/*
Concept

Pairs

1. How to sort array in increasing order ?
 sort(arr.begin(),arr.end());

2.  How to sort array in decreasing order ?
sort(arr.begin(),arr.end(),greater<int>);
or ,

sort(arr.rbegin(),arr.rend());


3. sort according to particular condition ?

conditon "k"
```

with array elements we have to do sommething which gives me some value "k" and this "k" will be for each index element

We have to sort the array a/c to the condition k & in increasing order.

```
vector<pair<int,int>>ans;
```
see for sorting we need two elements

concept of lambda function came :
keyword variable name, array , this comparator will take 2 parameter
```
auto     comp =           [](int &k1 , int &k2){

    can write bool also inspite of auto and but is convension to write auto
    return k1<k2; // condition :- True/False
    if true will be returned it means k1<k2 else vice versa

    internally what is happening is
    when we sort the array we get two indexes , k1 will be first then k2 if k1<k2;
};



auto comp =[](int &l1,int&l2){


  return  condition;
};

vector<int>ip;
```
if I have to sort the array according to soe condition then I will use lambda function

```
vector<int>ip;  parameter is int
in lambda function parametere something like this
vector<pair<int,int>>ip; paramter is pair
vector<vector<int>>ip; , in the parameter it will be vector<int>
```

```
parameter is based on what type of vector I have

It will take only 2 parameters

a -> 1,2,3,4,5

sort in decreasing order :- 5,4,3,2,1



We have to return the vecotr in which elements are present
according to their total time taken to recieve the order


2 things :- Customer Id  and total serve time

pair :-  (Customer Id, total preparation time )

sort -> acc. to their total preparation time

vector<pair<int ,int>>ans;

pair :- (Customer id , total preparation time )

sort : a/c  to their increasing order
 */

vector<int> jimOrders(vector<vector<int>> orders) {


// customer id , ,total serve time
vector<pair<int,int>>ans;

for (int i=0;i<orders.size(); i++) {

    // this pair <int> p; p.first= ; p.second =;
    // or {first,second} way to put elements in the pair
ans.push_back({i+1,orders[i][0]+orders[i][1]});

}
```

```
// lambda function
auto comp =[](pair<int, int>&x,pair<int,int>&y){
  // nothing wrong to use bool instead of auto
  if (x.second==y.second) {
  return  x.first<y.first; //if prearation time is same then
return a/c to their customer id
  }
  return x.second<y.second;

};


sort(ans.begin(),ans.end(),comp);
vector<int>op;

for (int i=0; i<ans.size(); i++) {
op.push_back(ans[i].first); // ans[i].first is customer Id
}
return op;


// Main concept was how to sort an array with given some
conditions where we used Lambda function
}
```

Description :

The `jimOrders` function takes a vector of orders, where each order is represented as a vector with two elements: the customer's preparation time and the time the order is received. The function assigns a pair to each order, consisting of the customer ID and the total serve time. It then sorts this vector of pairs according to the total serve time, and in case of a tie, based on the customer ID. Finally, it extracts and returns the sorted customer IDs.

Here's a concise summary:

1. Create pairs representing customer IDs and total serve times for each order.

2. Sort the pairs based on increasing order of total serve times and, in case of a tie, on customer IDs.

3. Extract and return the sorted customer IDs.

The function demonstrates the use of lambda functions for custom sorting based on specific conditions.

Q.44) Missing Numbers
https://www.hackerrank.com/challenges/missing-numbers/problem?isFullScreen=true

Code :

```
vector<int> missingNumbers(vector<int> arr, vector<int> brr) {
unordered_map<int, int >mp;
for (int i=0; i<brr.size(); i++) {
mp[brr[i]]++;
}
for (int i=0; i<arr.size(); i++) {
mp[arr[i]]--; // as a is subset of b , isliye minus kar rahe hai
mere dost

if (mp[arr[i]]==0) {
mp.erase(arr[i]);
}
}
vector<int > ans;

for(auto it:mp){
    ans.push_back(it.first);
}
sort(ans.begin(),ans.end());
return ans;
}
```

Description :

See there are ‘a’ and ‘b’ arrays where ‘a’ is a subset of ‘b’ , add all the elements of ‘b’ in the map , okay ?? aur increment kartey jao for frequency , ,then ‘a’ ke elements jithne hai , means frequency ko subtract karo . if frequency 0 hain then use erase(). Remaining key value pair is the answer