**Neuroverse 2023 Editorial**

**Problem 1: Welcome to Neuroverse 2023!**

In this problem, the task was to check whether the word *neuroverse* was contained in the given string or not. This can be done by iterating over all characters in the string, and check if the current character is the potential starting character for the word *neuroverse*. If we encounter such a character, then we print “YES” along with the index of that character in the string. Otherwise, if we do not find such a character even after scanning the entire string, then we print “NO”.

The solution could be implemented either by manually writing the code for matching the substring starting from the current character, or by using various built in libraries in different programming languages.

C code:

#include <math.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <limits.h>

#include <stdbool.h>

int min(int x,int y){

if(x < y) return x;

return y;

}

bool check(char str[], int idx){

bool done = true;

char nv[] = "neuroverse";

for(int i = idx; i < min(idx+10,strlen(str)); i++){

if(nv[i-idx] != str[i]){

done = false;

break;

}

}

if(strlen(str)-idx < 10) done = false;

return done;

}

int main() {

char str[50];

scanf("%s",str);

for(int i = 0; i < strlen(str); i++){

if(check(str,i)){

printf("YES\n%d\n",i);

return 0;

}

}

printf("NO\n");

return 0;

}

**Problem 2: Tambi’s Bicycle**

The solution is just the difference between and .

C code:

#include <stdio.h>

int main() {

int c,d;

scanf("%d%d",&c,&d);

printf("%d\n",d-c);

return 0;

}

**Problem 3: Array Sum**

The problem basically asks for the number of unordered pairs such that

The solution is basically iterating over all numbers in , and for each number , to count the number of elements in the array such that .

The naïve way to implement the solution was to simply iterate over all and then for each to again iterate over all and count all such such that . This solution has a complexity of and was allowed to pass some (around 50%) of the test cases.

However, the intended solution is a bit optimized than the naïve one.

Firstly, we observe that the order of the elements in is not significant, thus we can sort in non-decreasing order. Then, we iterate over all the elements of . For each , instead of re-iterating over and finding all suitable pairs, we perform binary search to find the lower and upper bounds of in and update our answer accordingly. In this fashion, instead of spending time trying to find the suitable pairs for , we can do the same operation in time, thus optimizing the final solution from to .

An alternate solution is by maintaining the count of all elements of in a data-structure like a map in C++/Java or dictionary in Python/Pypy, which allows inserting and accessing data in at most time, and for all , to check the count of in the map, and updating our answer accordingly.

Python 3 code:

n,s = map(int,input().split())

arr = list(map(int,input().split()))

mp = {}

for e in arr:

mp[e] = mp.get(e,0)+1

ans = 0

for e in arr:

ans += mp.get(s-e,0)

print(ans//2)

**Problem 4: Somewhere Door**

This problem asks that given two numbers and , and an array of integers , we need to find such that and such that and print

We can solve this problem by simply sorting the array , and then finding such that and such that and print in linear time.

Alternately, we can also not sort the array and just find optimal and values by comparing current and values with other elements of and updating them accordingly.

PyPy3 code:

t1,t2 = map(int,input().split())

n = int(input())

arr = list(map(int,input().split()))

arr.sort()

a1,a2 = -1,-1

for e in arr:

if e <= t1: a1 = e

elif e >= t2:

a2 = e

break

print(a2-a1+1)