PROBLEM SOLVING

(Solving Various Problems Using C Language & Python)

Summer Internship Report Submitted in partial fulfillment

of the requirement for under graduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

M S S BHARADWAJ

221710313031

https://github.com/manthribharadwaj12

Under the Guidance of

Ms. B. K. V. P. S. Mahalakshmi

Assistant Professor

Department Of Computer Science Engineering
GITAM School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
July 2020

GITAM (DEEMED TO BE UNIVERSITY)
Hyderabad-502329, India
Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **"Solving Various Problems Using C Language & Python"** is being submitted by M S S BHARADWAJ (221710313031) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2019-20

It is faithful record work carried out by him at the **Computer Science Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Ms. B. K. V. P. S. Mahalakshmi
Assistant Professor
Department of CSE

Dr. S. Phani Kumar,
Professor and HOD
Department of CSE

Dhyanahitha
Educational Society

INTERNSHIP CERTIFICATE

This certificate is presented to

M S S Bharadwaj

in recognition of his/her hardwork and dedication in completing Advanced Programming Internship project in

Problem Solving

from **25th May, 2020 to 11th July, 2020** under the sponsorship of **Dhyanahitha Educational Society** in accordance with all requirements of graduation by **GITAM (Deemed to be University), Hyderabad.**

ANIL KUMAR M
PROJECT MANAGER
DHYANAHITHA EDUCATIONAL SOCIETY

DHYANAHITHA EDUCATIONAL SOCIETY Hyderabad

SURESH N
DIRECTOR OF OPERATIONS
DHYANAHITHA EDUCATIONAL SOCIETY

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad,** Pro Vice Chancellor, GITAM Hyderabad and **Dr. N. Seetharamaiah,** Principal, GITAM Hyderabad

I would like to thank respected **Dr. S. Phani Kumar,** Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculty **Ms. B. K. V. P. S. Mahalakshmi** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

M S S BHARADWAJ
221710313031

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION TO THE PROJECT

Problem Solving is the Process of Designing and carrying out certain steps to reach a solution. There are eight problems which are listed below are of different complexity and require different approach and logics in order to achieve the desired Output/ Solution.

1. **Last Digit** - In this problem, we compute the last digit of a sequence which is calculated with the given formula.
2. **Replace Occurrences With The Given Words -** In this problem, we search for a particular word from the input and we replace it with another word.
3. **Bike Tour -** In this problem, we calculate the number of peak points as per given conditions.
4. **Robot Path Decoding -** In this problem, we calculate the position of the robot based on the given input.
5. **Perfect Subarray** - In this problem, we find out the count of subarrays which are perfect as per given conditions.
6. **Big City Skylines** - In this problem, we calculate the maximum area rectangle from inputs.
7. **ATM** - In this problem, we check multiple transactions based on the given conditions.
8. **Cryptopangrams** - In this problem, we recover pangrams based on the inputs.

I have executed projects in C language and Python. For C language, I have used DEV C++ to execute the codes and for Python, I have used Jupyter Notebook and GOOGLE KICKSTART interpreter.

PROBLEM-1

LAST DIGIT

This is a problem where we are going to use a formula to compute the number and get the last digit of that number.

2.1 Problem Statement

Compute the last digit of the following sequence

$$S = \sum_{i=0,2^i \leq n}^{\infty} \sum_{j=0}^{n} 2^{2^i+2j}$$

Fig 2.1.1

i.e. Summation of F(n) from s=0 to 2^i &tl;= n, where F(n) is the summation of 2^(2^i+2j) Where j varies from 0 to n.

Input: -

A Single Integer n

Output: -

Last digit of the number S

Sample Input	Sample Output
1) 3	0
2) 10	8

Concepts Used To Solve: -

For loop, math.h pow function and modulo concept

FOR Loop:

A for-loop (or simply for loop) is a control flow statement for specifying iteration, which allows code to be executed repeatedly. The name for-loop comes from the English word for, which is used as the keyword in many programming languages to introduce a for-loop.

Syntax:

for (initializationStatement; testExpression; updateStatement)

{

// statements inside the body of loop

}

Math.h:

The C <math.h> header file declares a set of functions to perform mathematical operations such as: sqrt() to calculate the square root, log() to find natural logarithm of a number, etc.

pow( ) to calculate a^b .

Syntax:

#include<math.h>

pow(a,b) // a^b

Modulo:

The modulo operator, denoted by %, is an arithmetic operator. The modulo division operator produces the remainder of an integer division.

Syntax:

If x and y are integers, then the expression:

x % y

Produces the remainder when x is divided by y.

2.2 Coding

```
#include<stdio.h>
#include<math.h>
int main()
{
 int i,j,n,b,s=0,a;
 scanf("%d",&n);
 for(i=0;pow(2,i)<=n;i++)
 {
    a=pow(2,i);
    for(j=0;j<=n;j++)
    {
        b=pow(2,(a+2*j));
        s=s+b;
    }
 }
 printf("%d",s%10);
 return 0;
}
```

Fig 2.2.1

2.3 Output

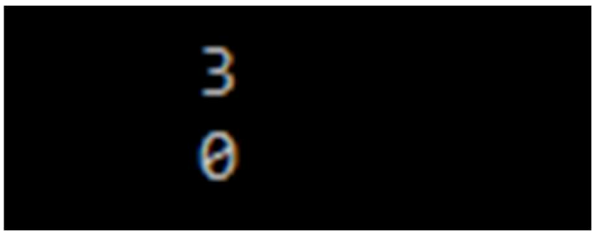

Fig 2.3.1

PROBLEM-2

REPLACE OCCURENCES WITH THE GIVEN WORD

This is a problem where we are going to search for a particular word from the input and we replace it with another word.

3.1 Problem Statement

The program must accept three string values S1, S2 and S3 as the input. The program must find all occurrences of the string S1 in the string S3 and replace all the occurrences of the string S1 by the string S2. Finally, the program must print the modified string S3 as the output.

Note: String values are only in lower case

Input: -

3 Strings S1, S2 and S3

Output: -

Modified string S3

Sample Input	Sample Output
1) tiger lion the tiger is a wild animal the tiger is known as the king of the jungle	the lion is a wild animal the lion is known as the king of the jungle
2) abcd xyz bcd abc abcd cde abcde abcd asdf bcde	bcd abc xyz cde abcde xyz asdf bcde

Concepts Used To Solve: -

String Functions - string compare(strcmp) and string tokenization(strtok)

String Compare (strcmp):

It compares the two strings and returns an integer value. If both the strings are same (equal) then this function would return 0 otherwise it may return a negative or positive value based on the comparison.

Syntax:

int strcmp(const char *str1, const char *str2)

Return Value:

- **Zero (0)**: A value equal to zero when both strings are found to be identical. That is, All of the characters in both strings are same.

- **Greater Than Zero (>0)**: A value greater than zero is returned when the first not matching character in leftStr have the greater ASCII value than the corresponding character in rightStr or we can also say.
- **Less Than Zero (<0)**: A value less than zero is returned when the first not matching character in leftStr have lesser ASCII value than the corresponding character in rightStr.

String Tokenization (strtok):

The C library function char *strtok(char *str, const char *delim) breaks string str into a series of tokens using the delimiter delim.

Syntax:

Following is the declaration for strtok() function.

char *strtok(char *str, const char *delim)

Parameters:

- **str** − The contents of this string are modified and broken into smaller strings (tokens).
- **delim** − This is the C string containing the delimiters. These may vary from one call to another.

Return Value:

This function returns a pointer to the first token found in the string. A null pointer is returned if there are no tokens left to retrieve.

strcmp( ) and strtok( ) belongs to string.h header file.

3.2 Coding

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
    char string[1010],a[30],b[30];
    scanf ("%s", a);
    scanf ("%s", b);
    fflush (stdin);
    scanf ("%[^\n]s", string);
    char *p;
    p = strtok (string, " ");
    while (p != NULL)
    {
        if (strcmp (a, p) == 0)
        printf ("%s ", b);
        else
        printf ("%s ", p);
        p = strtok (NULL, " ");
    }
return 0;
}
```

Fig 3.2.1

3.3 Output

```
tiger lion
the tiger is a wild animal the tiger is known as the king of the jungle
the lion is a wild animal the lion is known as the king of the jungle
```

Fig 3.3.1

PROBLEM-3

BIKE TOUR

This is a problem where we calculate the number of peak points as per given conditions.

4.1 Problem Statement

Li has planned a bike tour through the mountains of Switzerland. His tour consists of **N** checkpoints, numbered from 1 to **N** in the order he will visit them. The i-th checkpoint has a height of $\mathbf{H_i}$.

A checkpoint is a *peak* if:

- It is not the 1st checkpoint or the **N**-th checkpoint, and
- The height of the checkpoint is *strictly greater than* the checkpoint immediately before it and the checkpoint immediately after it.

Please help Li find out the number of peaks.

Input: -

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a line containing the integer **N**. The second line contains **N** integers. The i-th integer is $\mathbf{H_i}$.

Output: -

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the number of peaks in Li's bike tour.

Sample Input	Sample Output
3	Case #1: 1
3	Case #2: 0
10 20 14	Case #3: 2
4	
7 7 7 7	
5	
10 90 20 90 10	

Concepts Used To Solve: -

Lists

Lists:

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Syntax:

List_name = [E1,E2,…….]

4.2 Coding

```
t=int(input())
for i in range(1,t+1):
    n=int(input())
    a=list(map(int,input().split()))
    c=0
    for j in range(1,n-1):
        if (a[j-1]<a[j] and a[j+1]<a[j]):
            c+=1
    print("Case #"+str(i)+": "+str(c))
```

Fig 4.2.1

4.3 Output

```
4
3
10 20 10
Case #1: 1
4
7 7 7 7
Case #2: 0
5
10 90 20 90 10
Case #3: 2
```

Fig 4.3.1

PROBLEM-4

ROBOT PATH DECODING

This is a problem where we calculate the position of the robot based on the given input.

5.1 Problem Statement

Your country's space agency has just landed a rover on a new planet, which can be thought of as a grid of squares containing 10^9 columns (numbered starting from 1 from west to east) and 10^9 rows (numbered starting from 1 from north to south). Let (w, h) denote the square in the w-th column and the h-th row.

The rover begins on the square (1, 1).

The rover can be maneuvered around on the surface of the planet by sending it a *program*, which is a string of characters representing movements in the four cardinal directions. The robot executes each character of the string in order:

- N: Move one unit north.
- S: Move one unit south.
- E: Move one unit east.
- W: Move one unit west.

There is also a special instruction X(Y), where X is a number between 2 and 9 inclusive and Y is a non-empty subprogram. This denotes that the robot should repeat the subprogram Y a total of X times. For example:

- 2(NWE) is equivalent to NWENWE.
- 3(S2(E)) is equivalent to SEESEESEE.
- EEEE4(N)2(SS) is equivalent to EEEENNNNSSSS.

Since the planet is a torus, the first and last columns are adjacent, so moving east from column 10^9 will move the rover to column 1 and moving south from row 10^9 will move the rover to row 1. Similarly, moving west from column 1 will move the rover to column 10^9 and moving north from row 1 will move the rover to row 10^9. Given a program that the robot will execute, determine the final position of the robot after it has finished all its movements.

Input: -

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains a single string: the program sent to the rover.

Output: -

For each test case, output one line containing Case #x: w h, where x is the test case number (starting from 1) and w h is the final square (w, h) the rover finishes in.

Sample Input	Sample Output
4	Case #1: 4 4
SSSEEE	Case #2: 1 1000000000
N	Case #3: 3 1
N3(S)N2(E)N	Case #4: 3 999999995
2(3(NW)2(W2(EE)W))	

Concepts Used To Solve: -

Visualization of tough situations and string processing

Visualization Of Tough Situations:

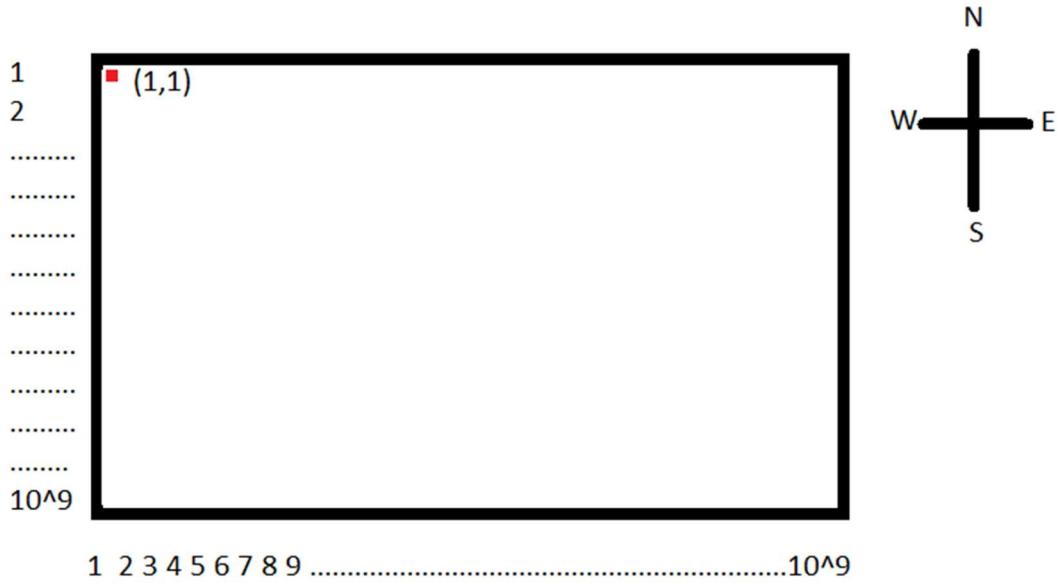

Fig 5.1.1

String Processing:

Processing to string to get desired output

5.2 Coding

```
t=int(input())
for t in range(1, t+1):
    p=input()
    cur=[0, 0]
    stack=[]
    for char in p:
        if char == 'N':
            cur[0] -= 1
        elif char == 'S':
            cur[0] += 1
        elif char == 'W':
            cur[1] -= 1
        elif char == 'E':
            cur[1] += 1
        elif char.isdigit():
            stack.append((cur[0], cur[1], int(char)))
        elif char == '(':
            cur = [0, 0]
        elif char == ')':
            pop = stack.pop()
            cur = [pop[i] + pop[2]*cur[i] for i in [0, 1]]
    final_row = (1+cur[0])%10**9
    if final_row == 0:
        final_row = 10**9

    final_column = (1+cur[1])%10**9
    if final_column == 0:
        final_column = 10**9
    print("Case #%d: %d %d" % (t, final_column, final_row))
```

Fig 5.2.1

5.3 Output

```
4
SSSEEE
Case #1: 4 4
N
Case #2: 1 1000000000
N3(S)N2(E)N
Case #3: 3 1
2(3(NW)2(W2(EE)W))
Case #4: 3 999999995
```

Fig 5.3.1

PROBLEM-5

PERFECT SUBARRAY

This is a problem where we find out the count of subarrays which are perfect as per given conditions

6.1 Problem Statement

Cristobal has an array of **N** (possibly negative) integers. The i-th integer in his array is $\mathbf{A_i}$. A contiguous non-empty subarray of Cristobal's array is *perfect* if its total sum is a perfect square. A perfect square is a number that is the product of a non-negative integer with itself. For example, the first five perfect squares are 0, 1, 4, 9 and 16.

How many subarrays are *perfect*? Two subarrays are different if they start or end at different indices in the array, even if the subarrays contain the same values in the same order.

Input: -

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of each test case contains the integer **N**. The second line contains **N** integers describing Cristobal's array. The i-th integer is $\mathbf{A_i}$.

Output: -

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the number of perfect subarrays.

Sample Input	Sample Output
3	Case #1: 1
3	Case #2: 3
2 2 6	Case #3: 9
5	
30 30 9 1 30	
4	
4 0 0 16	

Concepts Used To Solve: -

Arrays, subarrays, perfect square and sqrt

Array: An array is a collection of data items, all of the same type, accessed using a common name.

Subarray: Subarrays are arrays within another array. Subarrays contain contiguous elements.

Perfect Square: The perfect squares are the squares of the whole numbers.

Sqrt: It is function in math.h header file. It finds square root of a number.

Syntax:

sqrt(number)

6.2 Coding

```
import math
t=int(input())
for i in range(1,t+1):
    n=int(input())
    list1=list(map(int,input().split()))
    sublist = [math.sqrt(sum(list1[k:j]))-math.floor(math.sqrt(sum(list1[k:j])))==0
               for k in range(len(list1) + 1) for j in range(k + 1, len(list1) + 1)]
    print("Case #"+str(i)+": "+str(sublist.count(1)))
```

Fig 6.2.1

6.3 Output

```
3
3
2 2 6
Case #1: 1
5
30 30 9 1 30
Case #2: 3
4
4 0 0 16
Case #3: 9
```

Fig 6.3.1

PROBLEM-6

BIG CITY SKYLINES

This is a problem where we calculate the maximum area rectangle from inputs.

7.1 Problem Statement

You've just moved to the Big City so you could start work at the newest Google office, Google BC, and the one thing that interests you most is the beautiful skyline. You're sitting on a hillside ten miles away, looking at the big rectangular buildings that make up the city, and you think back to your childhood.

When you were a child, you used to make toy cities out of rectangular blocks. Each building could be made from multiple blocks, and each block could be part of multiple buildings. Looking at the skyline, you wonder: what is the biggest possible block that could be part of the Big City's skyline?

Write a program that takes the description of the skyline as an input, and gives the area of the maximum area rectangle as output. This program should take less than 4 minutes to run on a 2GHz computer with 512MB of RAM, even for the biggest input size we specify.

Input: -

The city is made out of rectangular buildings, all next to each other. The input will consist of an integer N, the number of buildings, followed by a series of N number pairs (wi, hi), indicating the width and height of each building in order from left to right. The buildings' heights will be less than 100,000,000, and their widths will be less than 1000. Easy: 0 < N < 1,000 Hard: 0 < N < 10,000,000

Output: -

A single number: the area of the largest possible block. (Here's the beautiful skyline of the Big City.)

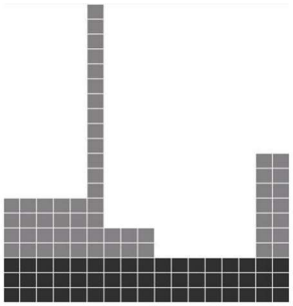

Fig 7.1.1

Sample Input	Sample Output
5 5 7 1 20 3 5 6 3 2 10	51

Concepts Used To Solve: -

Lists, maximum and minimum (max( ) and min( ))

Lists:

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Syntax:

List_name = [E1,E2,…….]

Maximum And Minimum: (max( ) and min( ))

These functions help us to find out the maximum and minimum element of a list.

Syntax:

min(list_name)

max(list_name)

Return Value:

min(list_name) --------- minimum value of the given list

max(list_name) --------- maximum value of the given list

7.2 Coding

```
n=int(input())
a=list(map(int,input().split()))
a1=[a[i] for i in range(0,len(a)) if i%2==0]
a2=[a[i] for i in range(0,len(a)) if i%2!=0]
print(sum(a1)*min(a2))
```

Fig 7.2.1

7.3 Output

```
5
5 7 1 20 3 5 6 3 2 10
51
```

Fig 7.3.1

PROBLEM-7

ATM

This is a problem where we check multiple transactions based on the given conditions.

8.1 Problem Statement

Pooja would like to withdraw X $US from an ATM. The cash machine will only accept the transaction if X is a multiple of 5, and Pooja's account balance has enough cash to perform the withdrawal transaction (including bank charges). For each successful withdrawal the bank charges 0.50 $US. Calculate Pooja's account balance after an attempted transaction.

Sample Input	Sample Output
30 120.00	89.50
42 120.00	120.00

Concepts Used To Solve: -

Validation of conditions

Validation Of Conditions:

Validation Conditions "check" responses to make sure they meet the constraints that you specify or given in the problem.

8.2 Coding

```
n=input().split(' ');

k=float(n[1])
n1=int(n[0])
if(n1%5==0 and n1<=k-0.5):
    n1=float(n1)+0.5;
    k=k-n1;
    print("%.2f" % k)
else:
    print("%.2f" % k)
```

Fig 8.2.1

8.3 Output

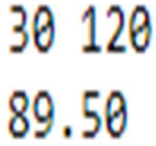

Fig 8.3.1

PROBLEM-8

CRYPTOPANGRAMS

This is a problem where we recover pangrams based on the inputs.

9.1 Problem Statement

On the Code Jam team, we enjoy sending each other *pangrams*, which are phrases that use each letter of the English alphabet at least once. One common example of a pangram is "the quick brown fox jumps over the lazy dog". Sometimes our pangrams contain confidential information, for example, CJ QUIZ: KNOW BEVY OF DP FLUX ALGORITHMS so we need to keep them secure.

We looked through a cryptography textbook for a few minutes, and we learned that it is very hard to factor products of two large prime numbers, so we devised an encryption scheme based on that fact. First, we made some preparations:

- We chose 26 different prime numbers, none of which is larger than some integer **N**.
- We sorted those primes in increasing order. Then, we assigned the smallest prime to letter A, the second smallest prime to letter B, and so on.
- Everyone on the team memorized this list.

Now, whenever we want to send a pangram as a message, we first remove all spacing to form a plaintext message. Then we write down the product of the prime for the first letter of the plaintext and the prime for the second letter of the plaintext. Then we write down the product of the primes for the second and third plaintext letters, and so on, ending with the product of the primes for the next-to-last and last plaintext letters. This new list of values is our ciphertext. The number of values is smaller than the number of characters in the plaintext message.

For example, suppose that **N** = 103 and we chose to use the first 26 odd prime numbers, because we worry that it is too easy to factor even numbers. Then A = 3, B = 5, C = 7, D = 11, and so on, up to Z = 103. Also, suppose that we want to encrypt the CJ QUIZ... pangram above, so our plaintext is CJQUIZKNOWBEVYOFDPFLUXALGORITHMS. Then the first value in our ciphertext is 7 (the prime for C) times 31 (the prime for J) = 217; the next value is 1891, and so on, ending with 3053.

We will give you a ciphertext message and the value of **N** that we used. We will not tell you which primes we used, or how to decrypt the ciphertext. Do you think you can recover the plaintext anyway?

Input: -

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each test case consists of two lines. The first line contains two integers: **N**, as described above, and **L**, the length of the list of values in the ciphertext. The second line contains **L** integers: the list of values in the ciphertext.

Output: -

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is a string of **L** + 1 uppercase English alphabet letters: the plaintext.

Sample Input	Sample Output
2 103 31 217 1891 4819 2291 2987 3811 1739 2491 4717 445 65 1079 8383 5353 901 187 649 1003 697 3239 7663 291 123 779 1007 3551 1943 2117 1679 989 3053 10000 25 3292937 175597 18779 50429 375469 1651121 2102 3722 2376497 611683 489059 2328901 3150061 829981 421301 76409 38477 291931 730241 959821 1664197 3057407 4267589 4729181 5335543	Case #1: CJQUIZKNOWBEVYOFDPFLUXALGORI THMS Case #2: SUBDERMATOGLYPHICFJKNQVWXZ

Concepts Used To Solve: -

Prime numbers, GCD and pangrams

Prime Numbers:

Prime numbers are numbers that have only 2 factors: 1 and themselves.

GCD:

The greatest common divisor of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.

Pangrams:

Pangram is a sentence or expression that uses all the letters of the alphabet.

9.2 Coding

```
def gcd(a, b):
    while b != 0:
        a, b = b, a%b
    return a

T = int(input())
for t in range(T):
    N, L = map(int, input().split())
    a = list(map(int, input().split()))
    p = [0]*(L+1)
    for i in range(L-1):
        c = gcd(a[i], a[i+1])
        if c != a[i] and c != a[i+1]:
            p[i+1] = c
    for i in range(L):
        if p[i] != 0 and p[i+1] == 0:
            p[i+1] = a[i] // p[i]
    for i in range(L, 0, -1):
        if p[i] != 0 and p[i-1] == 0:
            p[i-1] = a[i-1] // p[i]
    b = sorted(set(p))
    d = dict()
    for i in range(26):
        d[b[i]] = chr(ord('A') + i)
    s = ""
    for i in p:
        s += d[i]
    print("Case #" + str(t+1) + ": " + s)
```

Fig 9.2.1

9.3 Output

```
2
103 31
217 1891 4819 2291 2987 3811 1739 2491 4717 445 65 1079 8383 5353 901 187 649 1003 697 3239 7663 291 123 779 1007 3551 1943 211
7 1679 989 3053
Case #1: CJQUIZKNOWBEVYOFDPFLUXALGORITHMS
10000 25
3292937 175597 18779 50429 375469 1651121 2102 3722 2376497 611683 489059 2328901 3150061 829981 421301 76409 38477 291931 7302
41 959821 1664197 3057407 4267589 4729181 5335543
Case #2: SUBDERMATOGLYPHICFJKNQVWXZ
```

Fig 9.3.1

SOFTWARE REQUIREMENTS

10.1 Hardware Requirements

This project can be executed in any system or android phone without prior to any platform.

We can use any online compiler and interpreter.

10.2 Software Requirements

There are two ways to execute this project

1) Online compilers
2) Software applications for execution i.e.., IDE's (DEV C++, ANACONDA…..)

Online Compilers require only an internet connection. We have many free compilers with which we can code. IDEs need to be installed based on the user's system specification. These help us to completely execute the project. These softwares applications are based on the platforms. Many of them are free of cost.

Based on the availability we can use anyone.

Some of the best Online Compilers are:

OnlineGDB, Tutorial point, Programiz, Code chef and many more.

Some of the best Softwares applications for execution(IDE's) are:

For C:

Turbo C, DEV C++ etc.

For Python:

Jupyter Notebook, Python IDLE, Atom etc.

References:

- https://www.geeksforgeeks.org
- https://codingcompetitions.withgoogle.com/kickstart
- https://www.codechef.com
- https://static.googleusercontent.com/media/services.google.com/en//blog_resources/Google_CodeJam_Practice.pdf
- https://codingcompetitions.withgoogle.com/codejam