Ex:3 **Date**:22.08.24

Name: Veronica Regina Paul **Register number**:230701377

Greedy Algorithms-1-G-Coin Problem

AIM:

To find the minimum number of coins and/or notes needed to make the change for a given value using Indian currency denominations.

ALGORITHM:

1. Start with a given value V and initialize a variable c to keep track of the count of coins/notes.
2. Create an array arr[] with the available denominations in decreasing order: {1000, 500, 100, 50, 20, 10, 5, 2, 1}.
3. Iterate over the denominations starting from the largest:
    - Divide the remaining value V by the current denomination to determine how many coins/notes of that denomination are needed.
    - Add the result to c.
    - Update V by taking the remainder of the division (V % arr[i]).
4. Repeat the process for all denominations until the remaining value V becomes zero.
5. Print the count c as the minimum number of coins/notes required.

1. Print count, which represents the maximum number of content children.

PROBLEM

Write a program to take value V and we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change.

Input Format:

Take an integer from stdin.

Output Format:

print the integer which is change of the number.

Example Input :

64

Output:

4

Explanaton:

We need a 50 Rs note and a 10 Rs note and two 2 rupee coins.

PROGRAM

```
#include<stdio.h>
int main(){
   int v,c=0;
   int i;
   scanf("%d",&v);
   int arr[]={ 1, 2, 5, 10, 20, 50, 100, 500, 1000};
   for(i=8;i>=0;i--){
      c+=v/arr[i];
      v%=arr[i];

   }
   printf("%d",c);
}
```

OUTPUT

	Input	Expected	Got	
✔	49	5	5	✔

2-G-Cookies Problem

AIM:

To maximize the number of children who are content with the given cookies based on their greed factor and the size of available cookies.

ALGORITHM:

1. Input the number of children m and their greed factors g[].
2. Input the number of cookies n and their sizes s[].
3. Sort the greed factors g[] in increasing order.
4. Sort the cookie sizes s[] in increasing order.
5. Initialize a variable count to 0 to track the number of children made content.
6. Use a two-pointer approach:
    - Pointer i to traverse the greed factors and pointer j to traverse the cookie sizes.
    - If the current cookie can satisfy the child (i.e., s[j] >= g[i]), increase count and move both pointers i and j.
    - If the current cookie cannot satisfy the child, move only pointer j to try the next cookie.
7. Print count, which represents the maximum number of content children.

PROBLEM

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Example 1:

Input:

3

1 2 3

2

1 1

Output:

1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Constraints:

1 <= g.length <= 3 * 10^4

0 <= s.length <= 3 * 10^4

1 <= g[i], s[j] <= 2^31 - 1

PROGRAM

```
#include <stdio.h>
int main(){
   int m,n,count=0;
   scanf("%d",&m);
   int g[m];
   for (int i=0;i<m;i++)
   {
      scanf("%d",&g[i]);
   }
   scanf("%d",&n);
   int s[n];
   for (int i=0;i<n;i++)
   {
      scanf("%d",&s[i]);
   }
```

```
    for (int i=0;i<m;i++)
    {
        for (int j=0;j<n;j++)
        {
            if (g[i]<=s[i])
            {
                count++;
                i++;
            }
        }
    }
    printf("%d",count-1);
}
```

OUTPUT

	Input	Expected	Got	
✔	2 1 2 3 1 2 3	2	2	✔

3-G-Burger Problem

AIM:

To determine the minimum distance needed to burn the calories from the burgers by consuming them in an optimal order.

ALGORITHM:

1. Input the number of burgers n and their respective calorie counts a[].
2. Sort the array a[] of calories in descending order.
3. Initialize a variable k to 0 to keep track of the total distance to run.
4. For each burger, calculate the distance using the formula 3 * (i + 1) * a[i], where i is the index of the burger in the sorted list.
5. Add the computed distance for each burger to k.
6. Print the total distance k, which is the minimum distance to run.

PROBLEM

A person needs to eat burgers. Each burger contains a count of calorie. After eating the burger, the person needs to run a distance to burn out his calories. If he has eaten i burgers with c calories each, then he has to run at least $3^i * c$ kilometers to burn out the calories. For example, if he ate 3 burgers with the count of calorie in the order: [1, 3, 2], the kilometers he needs to run are $(3^0 * 1) + (3^1 * 3) + (3^2 * 2) = 1 + 9 + 18 = 28$.
But this is not the minimum, so need to try out other orders of consumption and choose the minimum value. Determine the minimum distance he needs to run. Note: He can eat burger in any order and use an efficient sorting algorithm.Apply greedy approach to solve the problem.

Input Format

First Line contains the number of burgers
Second line contains calories of each burger which is n space-separate integers

Output Format

Print: Minimum number of kilometers needed to run to burn out the calories

Sample Input

```
3
5 10 7

Sample Output
76
```

For example:

Test	Input	Result
Test Case 1	3 1 3 2	18

PROBLEM

PROGRAM

```
#include<stdio.h>

int main()

{

   int n;

   scanf("%d",&n);

   int a[n];

   for(int i = 0;i < n;i++)

   {

      scanf("%d",&a[i]);

   }

   int k = 0;


   for(int i = 0;i < n-1;i++)

   {

      for(int j = 0;j < n-i-1;j++)

      {

         if(a[j] < a[j+1])

         {

            int t = a[j];

            a[j] = a[j+1];
```

```
            a[j+1] = t;
          }
        }
    }
    for(int i = 0;i < n;i++)
    {
        int p = 1;
        if(i == 0)
        k += (p*a[0]);
        else
        {
            for(int j = 1;j <= i;j++)
            {
                p *= n;
            }
            k += (p * a[i]);
        }

    }
    printf("%d",k);
}
```

OUTPUT

	Test	Input	Expected	Got	
✔	Test Case 1	3 1 3 2	18	18	✔
✔	Test Case 2	4 7 4 9 6	389	389	✔
✔	Test Case 3	3 5 10 7	76	76	✔

4-G-Array Sum max problem

AIM

To maximize the sum of the array where each element is multiplied by its index.

ALGORITHM:

1. Input the number of elements n and the array arr[].
2. Sort the array arr[] in ascending order.
3. Initialize a variable sum to 0.
4. For each element in the sorted array, multiply the element by its index i and add to sum.
5. Print the total sum, which is the maximum array sum.

PROBLEM

Given an array of N integer, we have to maximize the sum of arr[i] * i, where i is the index of the element (i = 0, 1, 2, ..., N).Write an algorithm based on Greedy technique with a Complexity O(nlogn).

Input Format:

First line specifies the number of elements-n

The next n lines contain the array elements.

Output Format:

Maximum Array Sum to be printed.

Sample Input:

5

2 5 3 4 0

Sample output:

40

PROGRAM

```
#include <stdio.h>
int main(){
   int n;
   scanf("%d",&n);
   int arr[n];
   for(int i=0;i<n;i++){
```

```
        scanf("%d",&arr[i]);
    }
     for (int i = 0; i < n; ++i){
       for (int j = i + 1; j < n; ++j){
        if (arr[i] > arr[j]){
          int a = arr[i];
          arr[i] = arr[j];
           arr[j] = a;
        }
       }
     }
    int sum=0;
    for(int i=0;i<n;i++){
       sum+=arr[i]*i;
    }
    printf("%d",sum);
}
```

OUTPUT

	Input	Expected	Got	
✔	5 2 5 3 4 0	40	40	✔
✔	10 2 2 2 4 4 3 3 5 5 5	191	191	✔
✔	2 45 3	45	45	✔

Product of Array elements-Minimum

AIM:

To rearrange two arrays A[] and B[] in such a way that the sum of the products of corresponding elements is minimized.

ALGORITHM:

1. Input the number of elements N and the two arrays A[] and B[].
2. Sort array A[] in ascending order and array B[] in descending order.
3. Initialize a variable s to 0 to keep track of the sum of products.
4. For each index i, multiply the corresponding elements A[i] and B[i] and add the product to s.
5. Print s, which is the minimum sum of the product of pairs from the two arrays.

PROBLEM

Given two arrays array_One[] and array_Two[] of same size N. We need to first rearrange the arrays such that the sum of the product of pairs(1 element from each) is minimum. That is SUM (A[i] * B[i]) for all i is minimum.

For example:

Input	Result
3 1 2 3 4 5 6	28

PROGRAM

```
#include<stdio.h>

int main()

{

   int N;

   scanf("%d",&N);

   int A[N],B[N];

   for(int i = 0;i < N;i++)
```

```
{
   scanf("%d",&A[i]);
}
for(int i = 0;i < N;i++)
{
   scanf("%d",&B[i]);
}
for(int i = 0;i < N-1;i++)
{
   for(int j = 0;j < N-i-1;j++)
   {
      if(A[j] > A[j+1])
      {
         int t = A[j];
         A[j] = A[j+1];
         A[j+1] = t;
      }
      if(B[j] < B[j+1])
      {
         int t = B[j];
         B[j] = B[j+1];
         B[j+1] = t;
      }
   }
}
int s = 0;
for(int i = 0;i < N;i++)
{
   s += (A[i] * B[i]);
}
printf("%d",s);
```

}

OUTPUT

	Input	Expected	Got	
✔	3 1 2 3 4 5 6	28	28	✔
✔	4 7 5 1 2 1 3 4 1	22	22	✔
✔	5 20 10 30 10 40 8 9 4 3 10	590	590	✔