

--	--	--

**Ex. No: 3**

**Date: 26.08.24**

**Register No.: 230701339**

**Name: Sri Vignesh.P**

## **Greedy Algorithm**

### **3.a. 1-G-Coin Problem**

**Aim:** 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.

#### **Algorithm:**

```
Int main() {
```

```
    initialize amt
```

```
    initialize count to 0
```

```
    read amt from user
```

```
    // array of currency denominations
```

--	--	--

--	--	--

initialize arr as {1, 2, 5, 10, 20, 50, 100, 500, 1000}

// loop through currency denominations from highest to lowest

for i from 8 down to 0 {

    count = count + (amt divided by arr[i]) // calculate number of notes of current denomination

    amt = amt modulo arr[i] // update amt to the remaining amount

}

print count // output the total count of notes

}

### Program:

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int amt,count=0;
```

```
    scanf("%d",&amt);
```

```
    int arr[]={ 1, 2, 5, 10, 20, 50, 100, 500, 1000} ;
```

```
    for (int i=8;i>=0;i--)
```

```
    {
```

```
        count+=amt/arr[i];
```

```
        amt%=arr[i];
```

```
    }
```

```
    printf("%d",count);
```

```
}
```

### Output:

	Input	Expected	Got	
✓	49	5	5	✓

--	--	--

--	--	--

--	--	--

--	--	--

### 3.b. 2-G-Cookies Problem

#### Aim:

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  $s[j] \geq g[i]$ , 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:

$1 \leq g.length \leq 3 \cdot 10^4$

$0 \leq s.length \leq 3 \cdot 10^4$

$1 \leq g[i], s[j] \leq 2^{31} - 1$

#### Algorithm:

```
function main() {
```

```
    initialize n // number of children
```

```
    read n
```

```
    initialize greed array of size n // array to hold children's greed factors
```

--	--	--

--	--	--

```
// read greed factors for each child
```

```
for i from 0 to n-1 {
    read greed[i] from user
}
```

```
initialize c // number of cookie sizes
```

```
read c from user
```

```
initialize csize array of size c // array to hold cookie sizes
```

```
// read cookie sizes
```

```
for j from 0 to c-1 {
    read csize[j] from user
}
```

```
initialize count to 0 // counter for satisfied children
```

```
// check each child's greed against available cookie sizes
```

```
for i from 0 to n-1 {
    for j from 0 to c-1 {
        if csize[j] is greater than or equal to greed[i] {
            increment count by 1 // child is satisfied
            break // exit inner loop after satisfying this child
        }
    }
}
```

```
print count // output the total count of satisfied children
```

```
}
```

### Program:

```
#include<stdio.h>
```

```
#include<string.h>
```

--	--	--

--	--	--

```

int main(){
    int n;
    scanf("%d",&n);
    int greed[n];
    for(int i=0;i<n;i++){
        scanf("%d",&greed[i]);
    }

    int c;
    scanf("%d",&c);
    int csize[c];
    for(int i=0;i<c;i++){
        scanf("%d",&csize[i]);
    }

    int count=0;
    for(int i=0;i<n;i++){
        for(int j=0;j<c;j++){
            if (csize[j]>=greed[i]){
                count++;
                break;
            }
        }
    }
    printf("%d",count);

}

```

**Output:**

--	--	--

--	--	--

	Input	Expected	Got	
✓	2	2	2	✓
	1 2			
	3			
	1 2 3			

--	--	--

--	--	--

### 3.c. 3-G-Burger Problem

#### Aim:

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
```

#### Algorithm:

```
int main() {
    initialize n // number of elements
    read n from user

    initialize cal array of size n // array to hold integers

    // read values into the cal array
    for i from 0 to n-1 {
        read cal[i] from user
    }
```

--	--	--



--	--	--

```
// sorting the array using bubble sort
```

```
for i from 0 to n-2 {
```

```
    for j from 0 to n-i-2 {
```

```
        if cal[j] is greater than cal[j+1] {
```

```
            // swap cal[j] and cal[j+1]
```

```
            initialize temp as cal[j]
```

```
            cal[j] = cal[j+1]
```

```
            cal[j+1] = temp
```

```
        }
```

```
    }
```

```
}
```

```
initialize mulfact // variable to hold power value
```

```
initialize sum to 0 // variable to hold the final sum
```

```
initialize h to n-1 // index for the last element
```

```
// compute the weighted sum
```

```
for i from 0 to n-1 {
```

```
    mulfact = n raised to the power of i // compute  $n^i$ 
```

```
    sum = sum + (mulfact * cal[h]) // accumulate the weighted sum
```

```
    h = h - 1 // move to the next element
```

```
}
```

```
print sum // output the final result
```

```
}
```

### Program:

```
#include<stdio.h>
```

```
#include<math.h>
```

```
int main(){
```

```
    int n;
```

--	--	--

--	--	--

```
scanf("%d",&n);
int cal[n];
for(int i=0;i<n;i++){
    scanf("%d",&cal[i]);
}
//sorting the array
int i, j, temp;
for (i = 0; i < n-1; i++) {
    for (j = 0; j < n-i-1; j++) {
        if (cal[j] > cal[j+1]) {
            temp = cal[j];
            cal[j] = cal[j+1];
            cal[j+1] = temp;
        }
    }
}
int mulfact;
int sum=0;
int h=n-1;
for(int i=0;i<n;i++)
{
    mulfact=pow(n,i);
    sum+=mulfact*cal[h];
    h--;
}
printf("%d",sum);
}
```

--	--	--

--	--	--

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	✓

--	--	--

--	--	--

### 3.d. 4-G-Array Sum Max Problem

#### Aim:

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

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

#### Algorithm:

```
function main() {
    initialize n // number of elements
    read n from user

    initialize arr array of size n // array to hold integers

    // read values into the arr array
    for i from 0 to n-1 {
        read arr[i] from user
    }

    // sorting the array using bubble sort
    for i from 0 to n-2 {
        for j from 0 to n-i-2 {
```

--	--	--

--	--	--

```

        if arr[j] is greater than arr[j+1] {
            // swap arr[j] and arr[j+1]
            initialize temp as arr[j]
            arr[j] = arr[j+1]
            arr[j+1] = temp
        }
    }
}

initialize prod to 0 // variable to hold the weighted sum

// compute the weighted sum
for i from 0 to n-1 {
    prod = prod + (arr[i] * i) // accumulate the weighted sum
}

print prod // output the final result
}

```

### 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-1;i++){
        for(int j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){

```

--	--	--

--	--	--

```

        int temp=arr[j];
        arr[j]=arr[j+1];
        arr[j+1]=temp;
    }
}
}
int prod=0;
for(int i=0;i<n;i++){
    prod+=(arr[i]*i);
}
printf("%d",prod);
}

```

**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	✓

--	--	--

--	--	--

--	--	--

--	--	--

### 3.e. 5-G-Product of Array Elements-Minimum

#### Aim:

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  $\text{SUM } (A[i] * B[i])$  for all i is minimum.

#### Algorithm:

```
function main() {
    initialize n // number of elements
    read n from user

    initialize array_One of size n // first array
    initialize array_Two of size n // second array

    // read values into array_One
    for i from 0 to n-1 {
        read array_One[i] from user
    }

    // read values into array_Two
    for i from 0 to n-1 {
        read array_Two[i] from user
    }

    // sorting both arrays
    for i from 0 to n-2 {
        for j from 0 to n-i-2 {
            // sort array_One in ascending order
            if array_One[j+1] is less than array_One[j] {
```

--	--	--



--	--	--

```

        // swap array_One[j] and array_One[j+1]
        initialize temp as array_One[j]
        array_One[j] = array_One[j+1]
        array_One[j+1] = temp
    }

```

```

        // sort array_Two in descending order
        if array_Two[j+1] is greater than array_Two[j] {
            // swap array_Two[j] and array_Two[j+1]
            initialize temp as array_Two[j]
            array_Two[j] = array_Two[j+1]
            array_Two[j+1] = temp
        }
    }
}

initialize sum to 0 // variable to hold the final sum

// calculate the sum of products of corresponding elements
for i from 0 to n-1 {
    sum = sum + (array_One[i] * array_Two[i]) // accumulate the product
}

print sum // output the final result
}

```

### Program:

```
#include<stdio.h>
```

```
int main(){
```

```
    int n;
```

--	--	--

--	--	--

```
scanf("%d",&n);
```

```
int array_One[n];
```

```
int array_Two[n];
```

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

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

```
}
```

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

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

```
}
```

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

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

```
        if(array_One[j+1]<array_One[j]){
```

```
            int temp=array_One[j];
```

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

```
            array_One[j+1]=temp;
```

```
        }
```

```
        if(array_Two[j+1]>array_Two[j]){
```

```
            int temp=array_Two[j];
```

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

```
            array_Two[j+1]=temp;
```

```
        }
```

```
    }
```

```
}
```

```
int sum=0;
```

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

```
    sum+=(array_One[i]*array_Two[i]);
```

```
}
```

--	--	--

--	--	--

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

**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	✓

--	--	--