GREEDY ALGORITHM

PROBLEM-1

AIM: 1G COIN PROBLEM

ALGORITHM:

1. Read the integer `V` (amount to be broken down into coins).

2. Set `count` to 0 and define an array `d[]` with coin denominations: `{1000, 500, 100, 50, 20, 5, 2, 1}`.

3. For each denomination `d[i]` in `d[]`, while `V >= d[i]`, subtract `d[i]` from `V` and increment `count`.

4. Repeat until `V` becomes 0 or all denominations are used.

5. Print the value of `count`, which is the minimum number of coins needed.

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

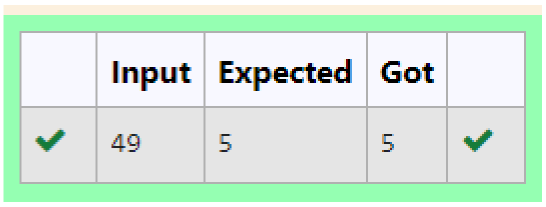
Explanaton:

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

PROGRAM:

```
#include<stdio.h>

int main()

{

    int V,count=0;

    scanf("%d",&V);

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

    int n = sizeof(d) / sizeof(d[0]);

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

    {

        while(V>=d[i])

        {

            V -= d[i];

            count++;

        }

    }

    printf("%d\n", count);

    return 0;

}
```

OUTPUT:



	Input	Expected	Got	
✓	49	5	5	✓

PROBLEM-2

AIM: 2G COOKIE PROBLEM

ALGORITHM:

1. Read the integer `ch` (number of children) and their greed factors into array `g[]`.
2. Read the integer `co` (number of cookies) and their sizes into array `s[]`.
3. Initialize a `used[]` array to track assigned cookies and `childrennum` to count assigned children.
4. For each child, find the smallest unassigned cookie that satisfies the child's greed factor and mark it as used.
5. Print the total number of children who receive cookies (`childrennum`).

PROBLEM:

ssume 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

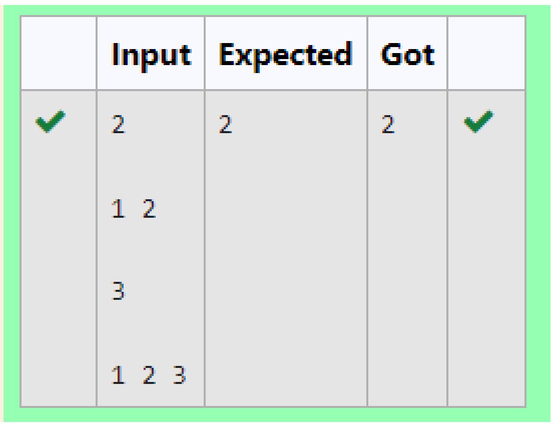
0 <= s.length <= 3 * 10^4

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

PROGRAM:

```
#include<stdio.h>
#define MAX_SIZE 30000
int main()
{
    int ch;
    int g[MAX_SIZE];
    scanf("%d", &ch);
    for (int i = 0; i < ch; i++)
    {
        scanf("%d", &g[i]);
    }

    int co;
    int s[MAX_SIZE];
    int used[MAX_SIZE] = {0};

    scanf("%d", &co);
    for (int j = 0; j < co; j++) {
        scanf("%d", &s[j]);
    }

    int childrennum = 0;

    for (int i = 0; i < ch; i++) {
        for (int j = 0; j < co; j++) {
            if (!used[j] && s[j] >= g[i]) {
                used[j] = 1;
                childrennum++;
                break;
            }
        }
    }

    printf("%d\n", childrennum);

    return 0;
}
```

OUTPUT:



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

PROBLEM-3

AIM: 3-G BURGER PROBLEM

ALGORITHM:

1. Read integer `m` (number of elements) and the array `a[]` of size `m`.

2. Perform a bubble sort on the array `a[]` in descending order.

3. Initialize `k` to 0 and `power` to 1. For each element in the sorted array, update `k` by adding `a[i] * power`, and update `power` by multiplying it by `m`.

4. Print the value of `k`.

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 3i * 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 (30 * 1) + (31 * 3) + (32 * 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 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 3i * 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 (30 * 1) + (31 * 3) + (32 * 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

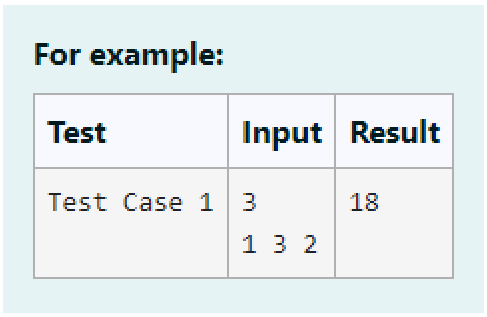
Sample Input

3

5 10 7

Sample Output

76



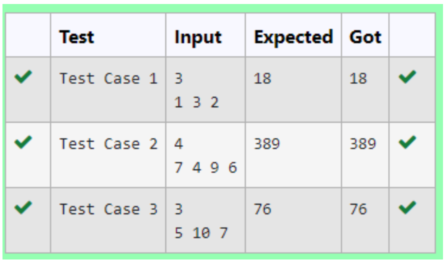
For example:

Test	Input	Result
Test Case 1	3 1 3 2	18

PROGRAM:

```
#include <stdio.h>

int main() {

    int m;
    scanf("%d", &m);
    int a[m];
    for (int i = 0; i < m; ++i) {
        scanf("%d", &a[i]);
    }
    for (int i = 0; i < m - 1; ++i) {
        for (int j = 0; j < m - i - 1; ++j) {
            if (a[j] < a[j + 1]) {
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
    int k = 0, power = 1;
    for (int i = 0; i < m; ++i) {
        k += a[i] * power;
        power *= m;
    }
    printf("%d\n", k);
    return 0;
}
```

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	✓

PROBLEM-4

AIM: 4-G ARRAY SUM MAX PROBLEM

ALGORITHM:

1. Read integer `m` (number of elements) and the array `arr[]` of size `m`.

2. Perform a bubble sort on the array `arr[]` in ascending order.

3. Initialize `sum` to 0. For each element in the sorted array, update `sum` by adding `arr[i] * i`.

4. Print the value of `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

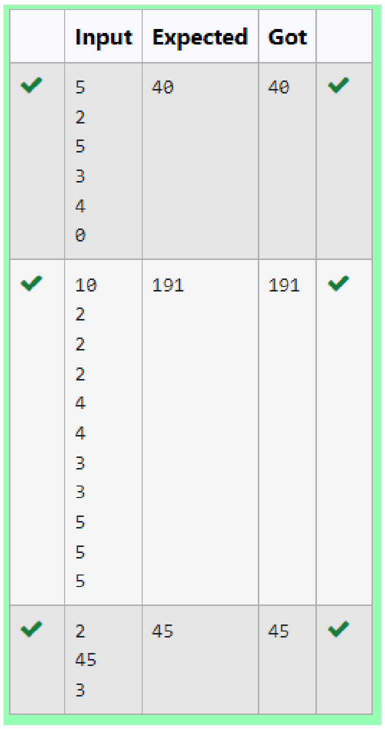
Sample output:

40

PROGRAM:

```
#include <stdio.h>

int main()

{

    int m;

    scanf("%d", &m);

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

OUTPUT:



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

PROBLEM-5

AIM: 5-G PRODUCT OF ARRAY ELEMENTS-MINIMUM

ALGORITHM:

1. Read integer `n` (number of elements) and two arrays `x[]` and `y[]` of size `n`.

2. Sort array `x[]` in ascending order using bubble sort.

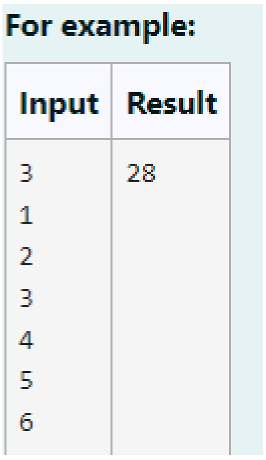
3. Sort array `y[]` in descending order using bubble sort.

4. Initialize `sum` to 0. For each element in the sorted arrays, update `sum` by adding `x[i] * y[i]`.

5. Print the value of `sum`.

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	28
1	
2	
3	
4	
5	
6	

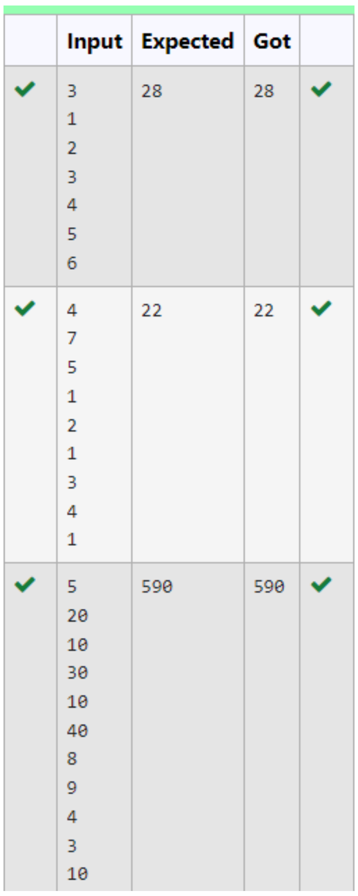
PROGRAM:

```
#include <stdio.h>

int main()

{

    int n;

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

```
int x[n], y[n];
for (int i = 0; i < n; i++)
{
    scanf("%d", &x[i]);
}
for (int i = 0; i < n; i++)
{
    scanf("%d", &y[i]);
}
for (int i = 0; i < n - 1; i++)
{
    for (int j = 0; j < n - i - 1; j++)
    {
        if (x[j] > x[j + 1])
        {
            int temp = x[j];
            x[j] = x[j + 1];
            x[j + 1] = temp;
        }
    }
}
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++)
    {
        if (y[j] < y[j + 1])
        {
            int temp = y[j];
            y[j] = y[j + 1];
```

```
                y[j + 1] = temp;
            }
        }
    }

    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += x[i] * y[i];
    }
    printf("%d\n", sum);
    return 0;
}
```

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	✓