Mastering Algorithms

Greedy Algorithms

Created by Venkatesh

Agenda

01

Intro to Greedy

Definition

02

Basic Problem

Greedy problem

03

Warning

You can't use random greedy algos

04

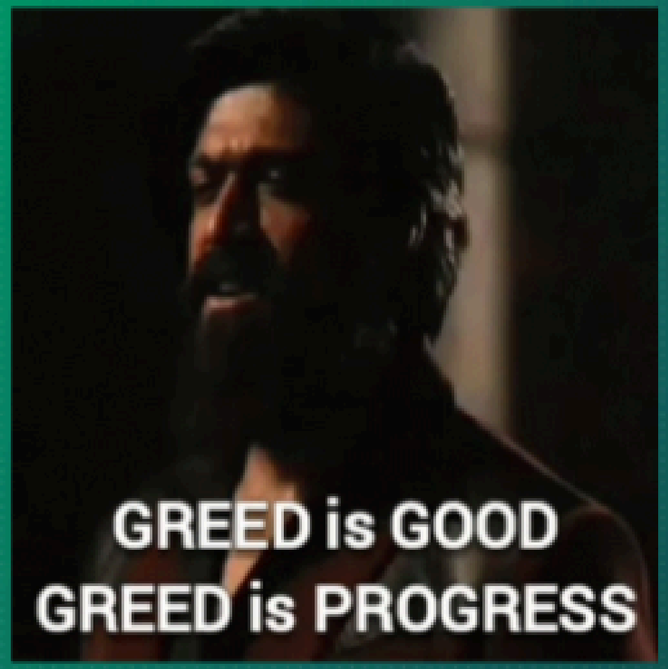
More Problems

Lets try a question

Add body text

Greedy

Be greedy



What is it ?

- Greedy algorithms make a locally optimal choice at at step in hope of finding the final optimal solution

(In other words, a greedy algorithm chooses the best possible option at each
step, without considering the consequences of that choice on future steps.)

- The algorithms are generally derived by intuition and proved later on

What is it ?

- Do note that in some problems , all that is required is some bit of logic , without some big algorithm

Problem

- n positive integers are given
- What is the maximum of them which can be chosen such that the total sum of the selected numbers is less than or equal to m
- **Example : First line n and m. Next line contains n integers.**

  7 100

  10 10 20 30 40 50 60
- Here the greedy algorithm is simple
- Keep selecting the smallest number and add it to the total sum until the total sum would exceed 100 if another is added

Problem

- Choose 10 ... then 10 ... then 20 ... then 30
- If we choose 40 next , the sum would exceed 100 so we do not choose it
- We can choose at max 4 numbers whose sum stays equal to or below 100

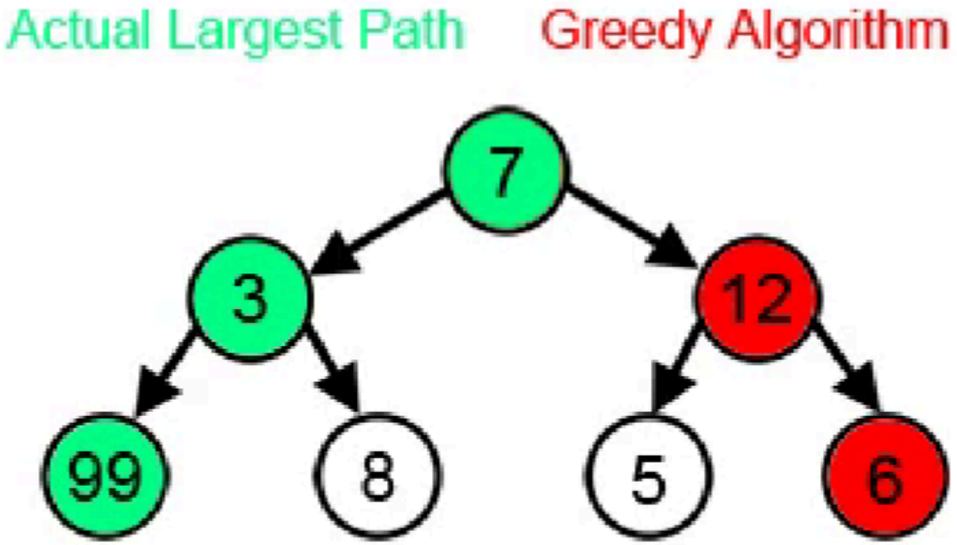
Code it

```cpp
int n,m;
cin >> n >> m;

int a[n];
for(int i=0;i<n;i++){
    cin >> a[i];
}

sort(a,a+n);

int max = 0;
int current_sum = 0;
for(int i=0;i<n;i++){
    if(a[i] + current_sum <= m) max++;
    current_sum = current_sum + a[i];
}

cout << max ;
```

Not Always

kill the order bro

- Find the path starting from 7 through nodes such that the sum of values on the nodes is maximum
- Here greedy algorithm might say , choose the largest value node at each step , but that would not work in all cases



- Greedy algorithms are derived by intuition so they may not always work
- Greedy approaches consider only locally optimal choices , and do not consider the broader picture , so they may not always work

Not Always

kill the order bro

More Problems

Discussions

G1. Teleporters (Easy Version)

1791 G1

The only difference between the easy and hard versions are the locations you can teleport to.

Consider the points $0, 1, \ldots, n$ on the number line. There is a teleporter located on each of the points $1, 2, \ldots, n$. At point i, you can do the following:

- Move left one unit: it costs 1 coin.
- Move right one unit: it costs 1 coin.
- Use a teleporter at point i, if it exists: it costs a_i coins. As a result, you teleport to point 0. Once you use a teleporter, you **can't** use it again.

You have c coins, and you start at point 0. What's the most number of teleporters you can use?

Input

The input consists of multiple test cases. The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases. The descriptions of the test cases follow.

The first line of each test case contains two integers n and c ($1 \le n \le 2 \cdot 10^5$; $1 \le c \le 10^9$) — the length of the array and the number of coins you have respectively.

The following line contains n space-separated integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the costs to use the teleporters.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output the maximum number of teleporters you can use.

Example

input Copy

```
10
5 6
1 1 1 1 1
8 32
100 52 13 6 9 4 100 35
1 1
5
4 5
4 3 2 1
5 9
2 3 1 4 1
5 8
2 3 1 4 1
4 3
2 3 4 1
4 9
5 4 3 3
2 14
7 5
5 600000000
500000000 400000000 300000000 200000000 100000000
```

output Copy

```
2
2
0
1
2
2
1
1
1
2
```

Problem

- In essence to use a teleporter at some position we should travel to it , then pay the price of the teleporter
- for the teleporter at point i, the total cost to use it would be $i + a_i$
- recall the first problem , This is essentially now the same thing.
- We have to choose maximum number of teleporters without spending more than c coins
- Where the cost to use each teleporter is essentially $i + a_i$

Code it

this code is for one test case

int may not work here for sum
What datatype instead ?

```cpp
int n,m;
cin >> n >> m;

int a[n];
for(int j=0;j<n;j++){
    int i = j + 1;
    cin >> a[j];
    a[j] = a[j] + i ;
}

sort(a,a+n);

int max = 0;
int current_sum = 0;
for(int i=0;i<n;i++){
    if(a[i] + current_sum <= m) max++;
    current_sum = current_sum + a[i];
}

cout << max ;
```

A. Hit the Lottery

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Allen has a LOT of money. He has n dollars in the bank. For security reasons, he wants to withdraw it in cash (we will not disclose the reasons here). The denominations for dollar bills are 1, 5, 10, 20, 100. What is the minimum number of bills Allen could receive after withdrawing his entire balance?

Input

The first and only line of input contains a single integer n ($1 \leq n \leq 10^9$).

Output

Output the minimum number of bills that Allen could receive.

.

Examples

input	Copy
125	
output	**Copy**
3	

input	Copy
43	
output	**Copy**
5	

input	Copy
1000000000	
output	**Copy**
10000000	

Note

In the first sample case, Allen can withdraw this with a 100 dollar bill, a 20 dollar bill, and a 5 dollar bill. There is no way for Allen to receive 125 dollars in one or two bills.

In the second sample case, Allen can withdraw two 20 dollar bills and three 1 dollar bills.

In the third sample case, Allen can withdraw 100000000 (ten million!) 100 dollar bills.

Code it

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long int
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    ll n, sum = 0;
    int v[] = {100, 20, 10, 5, 1};
    cin >> n;
    int i = 0;
    while (n != 0)
    {
        sum += n / v[i];
        n %= v[i];
        i++;
    }
    cout << sum;
    return 0;
}
```