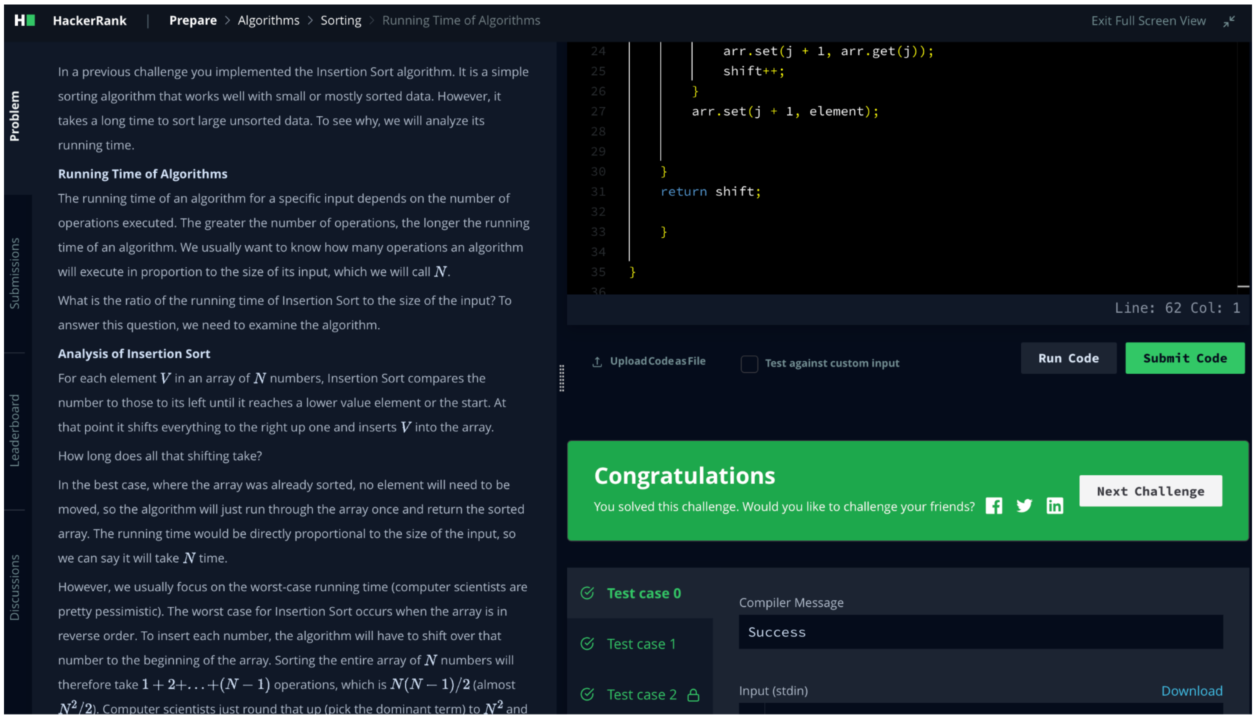
Running Time algorithm



HackerRank | Prepare > Algorithms > Sorting > Running Time of Algorithms

In a previous challenge you implemented the Insertion Sort algorithm. It is a simple sorting algorithm that works well with small or mostly sorted data. However, it takes a long time to sort large unsorted data. To see why, we will analyze its running time.

Running Time of Algorithms

The running time of an algorithm for a specific input depends on the number of operations executed. The greater the number of operations, the longer the running time of an algorithm. We usually want to know how many operations an algorithm will execute in proportion to the size of its input, which we will call N.

What is the ratio of the running time of Insertion Sort to the size of the input? To answer this question, we need to examine the algorithm.

Analysis of Insertion Sort

For each element V in an array of N numbers, Insertion Sort compares the number to those to its left until it reaches a lower value element or the start. At that point it shifts everything to the right up one and inserts V into the array.

How long does all that shifting take?

In the best case, where the array was already sorted, no element will need to be moved, so the algorithm will just run through the array once and return the sorted array. The running time would be directly proportional to the size of the input, so we can say it will take N time.

However, we usually focus on the worst-case running time (computer scientists are pretty pessimistic). The worst case for Insertion Sort occurs when the array is in reverse order. To insert each number, the algorithm will have to shift over that number to the beginning of the array. Sorting the entire array of N numbers will therefore take $1+2+\ldots+(N-1)$ operations, which is $N(N-1)/2$ (almost $N^2/2$). Computer scientists just round that up (pick the dominant term) to N^2 and

```
                arr.set(j + 1, arr.get(j));
                shift++;
            }
            arr.set(j + 1, element);

        }
        return shift;

    }
}
```

Line: 62 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Congratulations

You solved this challenge. Would you like to challenge your friends?

Next Challenge

Test case 0

Test case 1

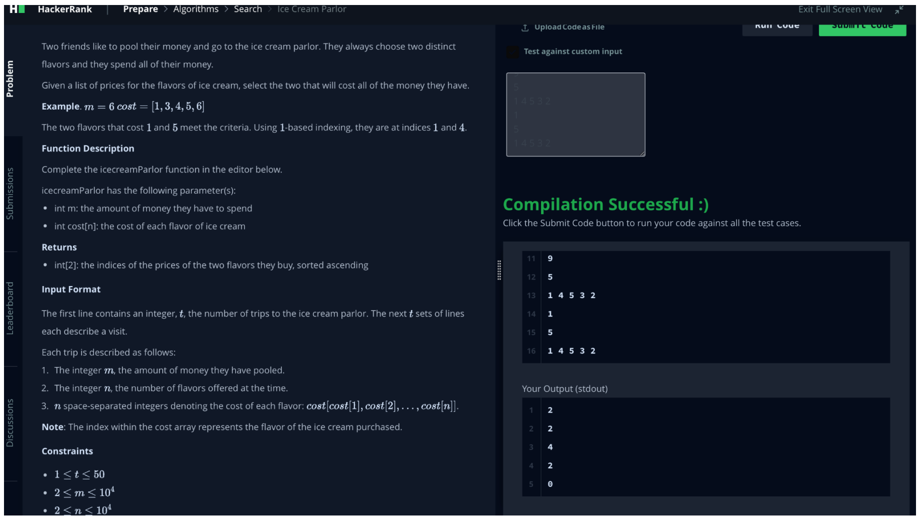
Test case 2

Compiler Message

```
Success
```

Input (stdin) Download

The time and space complexity:
Space complexity is constante O(1) since it is using fixed variables, and the size of the arrays is not dependent on that.
Combining the outer and inner for loops you get a time complexity of O(n^2) since it will run in a descending order series, 1 + 2 + 3 (n-1). If the array is sorted it will run with a best time of O(n) since the array is sorted.



HackerRank | Prepare > Algorithms > Search > Ice Cream Parlor

Two friends like to pool their money and go to the ice cream parlor. They always choose two distinct flavors and they spend all of their money.

Given a list of prices for the flavors of ice cream, select the two that will cost all of the money they have.

Example. $m = 6$ $cost = [1, 3, 4, 5, 6]$

The two flavors that cost 1 and 5 meet the criteria. Using 1-based indexing, they are at indices 1 and 4.

Function Description

Complete the icecreamParlor function in the editor below.

icecreamParlor has the following parameter(s):

- int m: the amount of money they have to spend
- int cost[n]: the cost of each flavor of ice cream

Returns

- int[2]: the indices of the prices of the two flavors they buy, sorted ascending

Input Format

The first line contains an integer, t, the number of trips to the ice cream parlor. The next t sets of lines each describe a visit.

Each trip is described as follows:

1. The integer m, the amount of money they have pooled.
2. The integer n, the number of flavors offered at the time.
3. n space-separated integers denoting the cost of each flavor: $cost[cost[1], cost[2], \ldots, cost[n]]$.

Note: The index within the cost array represents the flavor of the ice cream purchased.

Constraints

- $1 \le t \le 50$
- $2 \le m \le 10^4$
- $2 \le n \le 10^4$

Upload Code as File | Run Code | Submit Code

Test against custom input

```
5
1 4 5 3 2
1
5
1 4 5 3 2
```

Compilation Successful :)

Click the Submit Code button to run your code against all the test cases.

```
9
5
1 4 5 3 2
1
5
1 4 5 3 2
```

Your Output (stdout)

```
2
2
4
2
0
```

The way i tried to do it was by including and excluding combinations that sum up to the money, i got the amount of combinations almost right, i could not find a way to include already used prices that was for me the tricky part of the code.