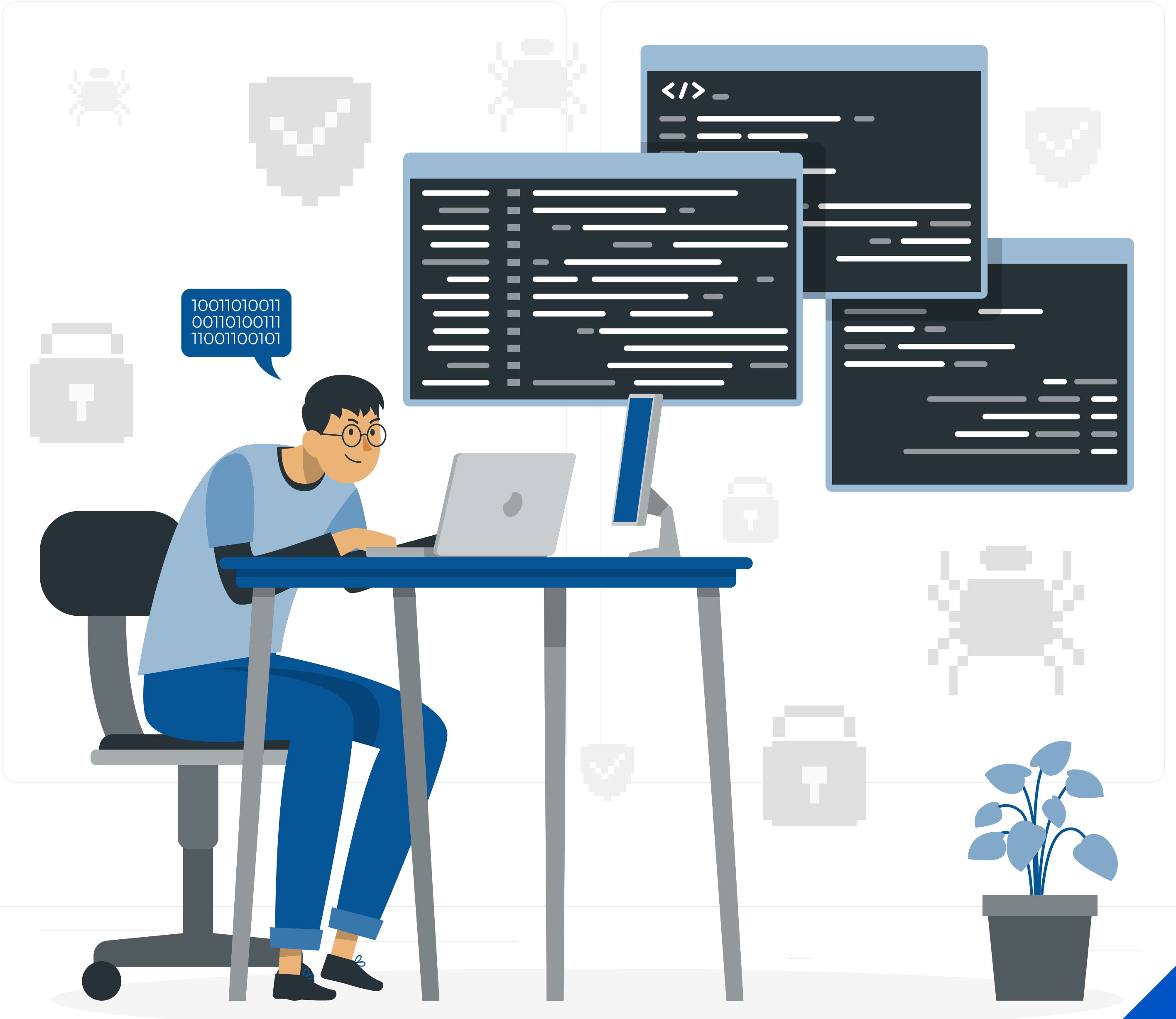
Association for
Computing Machinery
INSAT Student Chapter

Introduction to time and space complexity

Let’s solve a problem

- **Codeforces: 433B**



Motivation

Time limit exceeded on test 67

Time Limit Exceeded

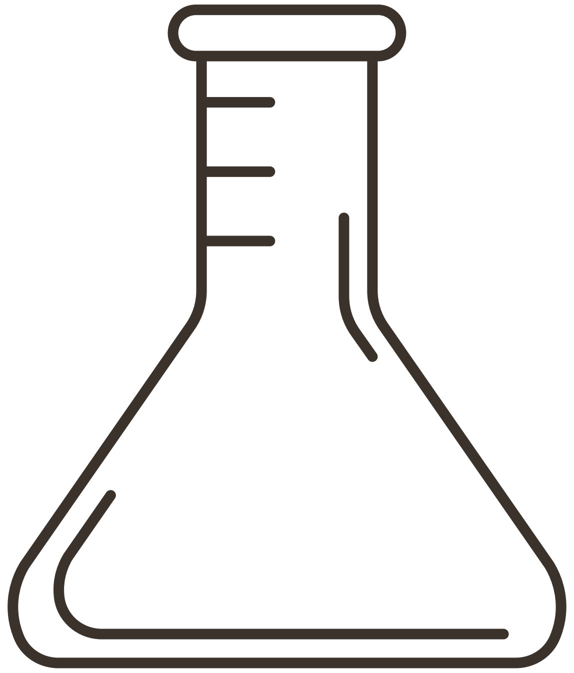
Time Limit Exceeded (1.01)

TLE

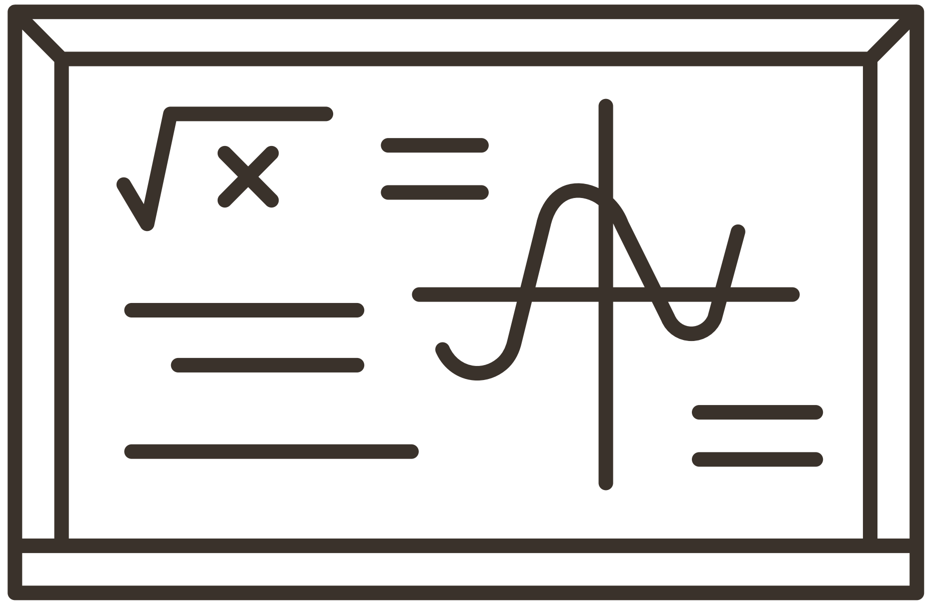
What is an algorithm?

- A finite set of instructions carried out in a specific order to perform a particular task
- Analyzing an algorithm has come to mean predicting the resources that the algorithm requires.
- More so than often, computational time is the main resource that we‘re interested in measuring as it gives us an idea about the performance of our algorithm.

How to measure computation time



Experimental method



Mathematical method

Experimental Method

- We simply measure the time the algorithm takes to finish (in different cases of the input)
  - Limitations:
    - Hardware dependant
    - Language dependant
    - Not practical (you can't predict performance and need to implement first)

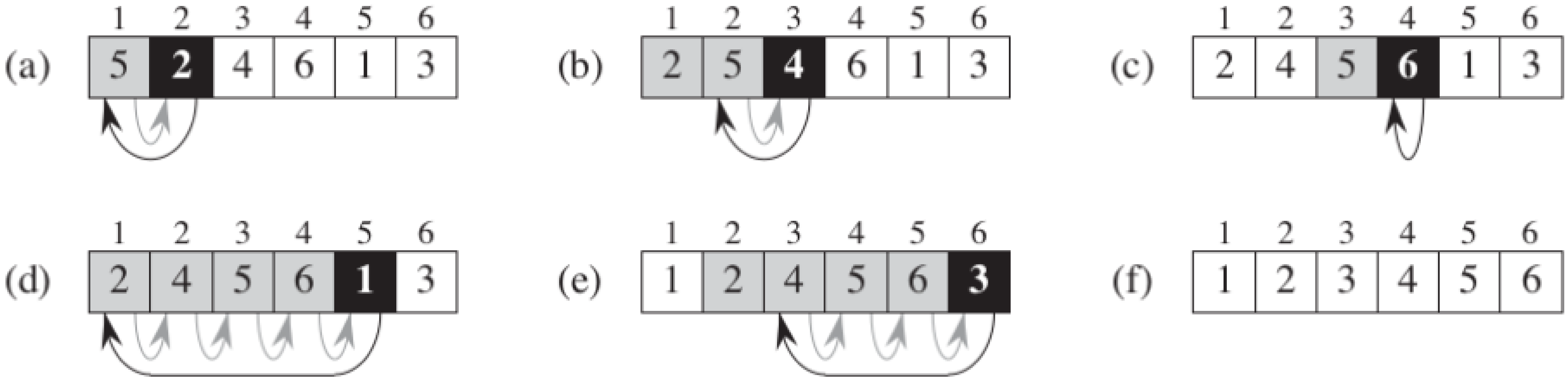
Mathematical Method

- We calculate the number of elementary operations an algorithm will take.
- 1 elementary operation = 1 unit of time
- We conclude the rate of growth of the computation time

Practical example

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

Practical example



Practical example

Observation: The number of steps depends on the input size

=> Time complexity is often described as a function of the input size f(n)

Observation: the algorithm can take drastically different number of steps depending on the input

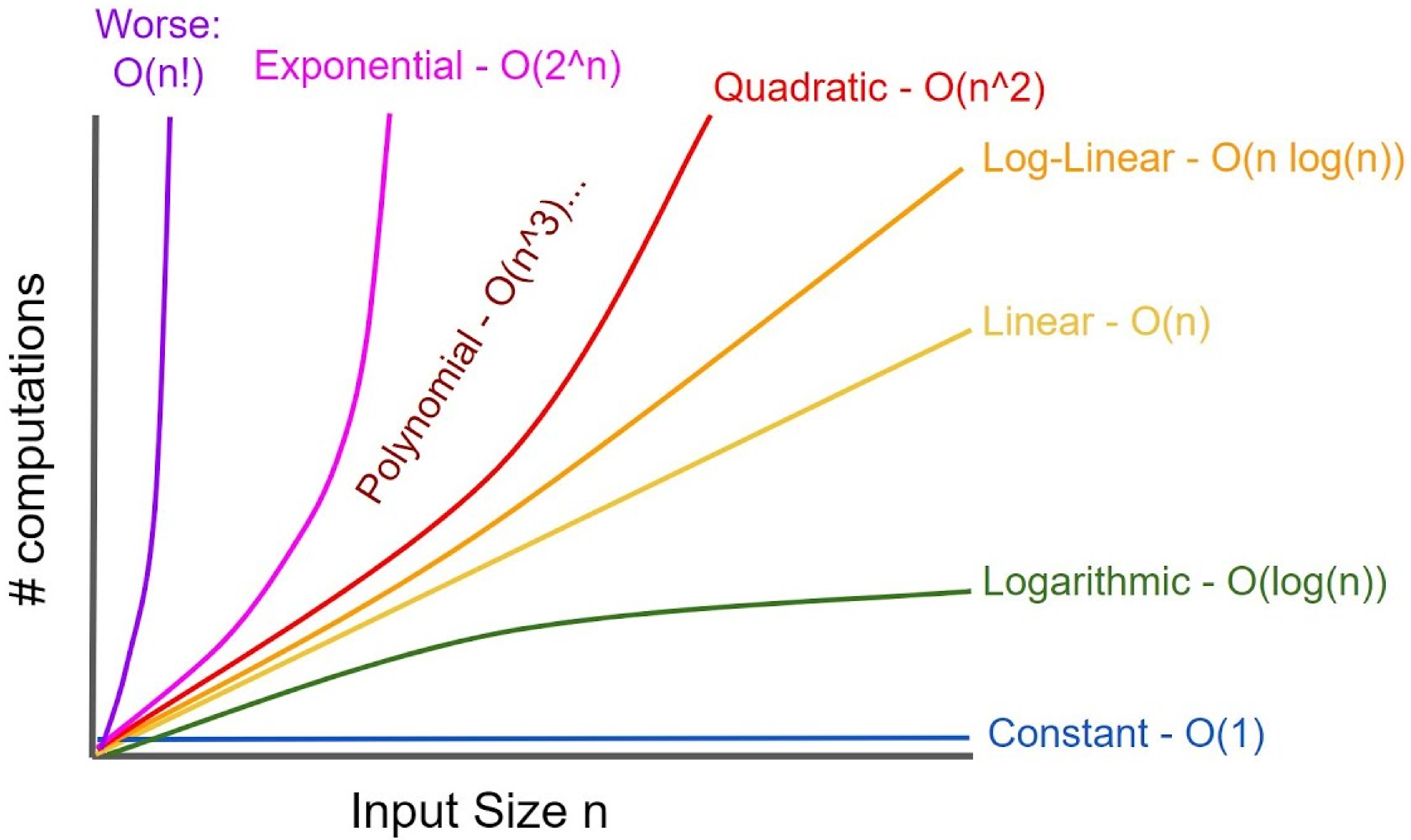
=> Time complexity is often calculated based on the worst case running time

Our worst case is having the array sorted in inverse

Practical example

- To better describe and classify algorithms, we use big-oh (O) to describe their rate of growth with respect to the input size.
- This gives us a great estimate of the behaviour of our algorithms as the input size gets bigger
- We can now conclude that our algorithm has a complexity of $O(n^2)$ (quadratic time complexity)

Usual time complexities



Rule of thumb

Input size	Required time complexity for 1s processing time
n ≤ 10	$O(n!)$
n ≤ 20	$O(2^n)$
n ≤ 500	$O(n^3)$
n ≤ 5000	$O(n^2)$
n ≤ 10^6	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

Let’s get back to our problem

- Let’s calculate the current time complexity: O(m*n)
- Is there a way to reduce the time complexity so that it fits our constraints?

Prefix Sum

original array	10	7	22	2	13	15	6
	0	1	2	3	4	5	6

prefix sums array	0	10	17	39	41	54	69	75
	0	1	2	3	4	5	6	7

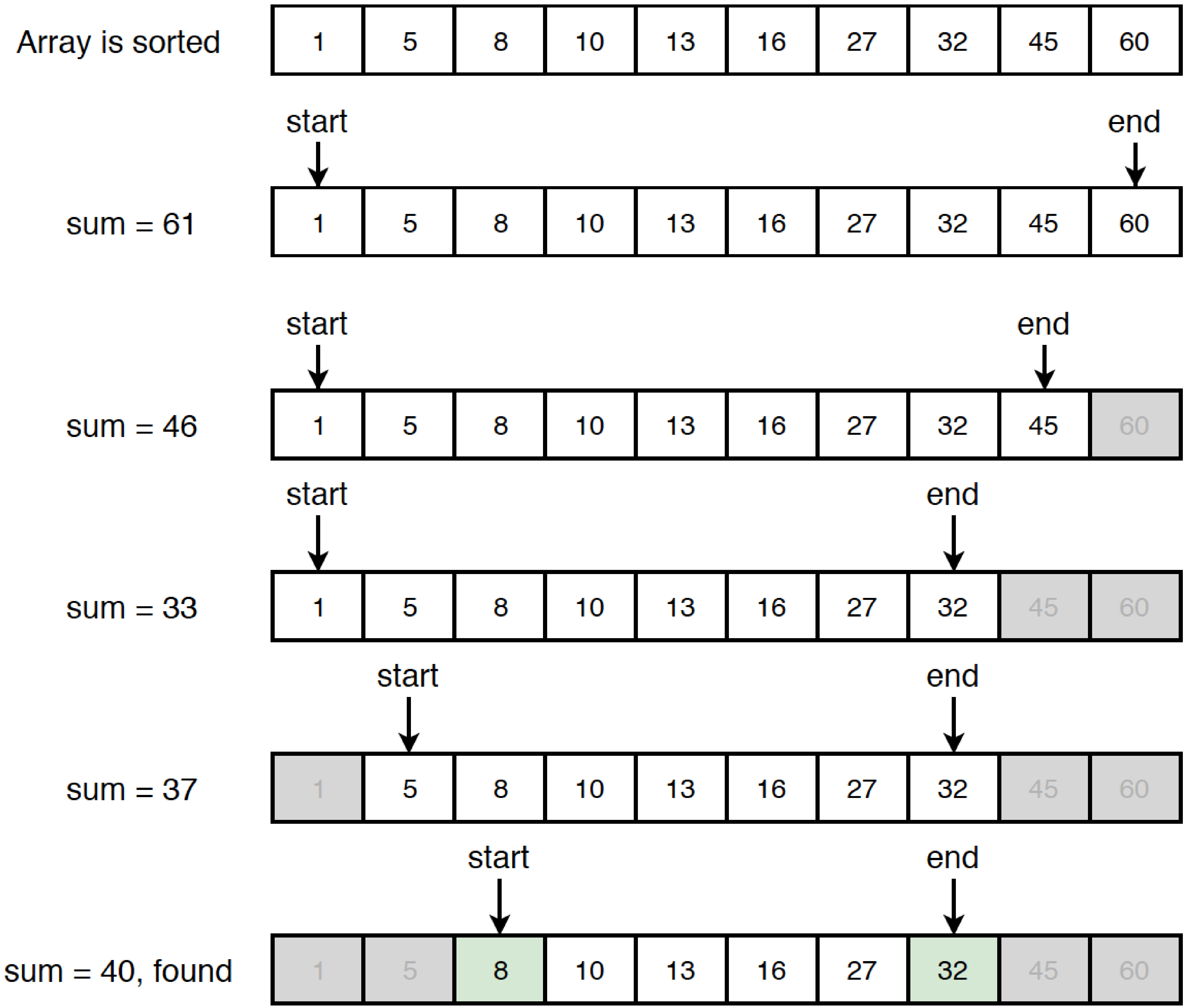
Let’s solve a problem

- **Codeforces: 102961G**

Analysis

- Using our rule of thumb, the problem calls for at least a log-linear solution O(n.log(n))
- A brute-force solution (checking every combination of two numbers) is $O(n^2)$ in time
  => Guaranteed to surpass the time limits
- Let's think of a better solution

Two pointers



Space complexity?

- Much like time complexity, the space Complexity of an algorithm is the total space taken by the algorithm with respect to the input size.
- We calculate the temporary space taken by the algorithm as a function of the input. (We use the worst case scenario)
- We then use big-oh notation to describe said function.
- Space complexity was a big deal back when computer memory was limited in size. Today, we generally don’t care much about it except for resource-constrained environment.
- Let’s calculate the space complexity of our two problems.

More Problems

- **Codeforces: 313B**
- **Codeforces: 1682A**

Thank you for your attention!