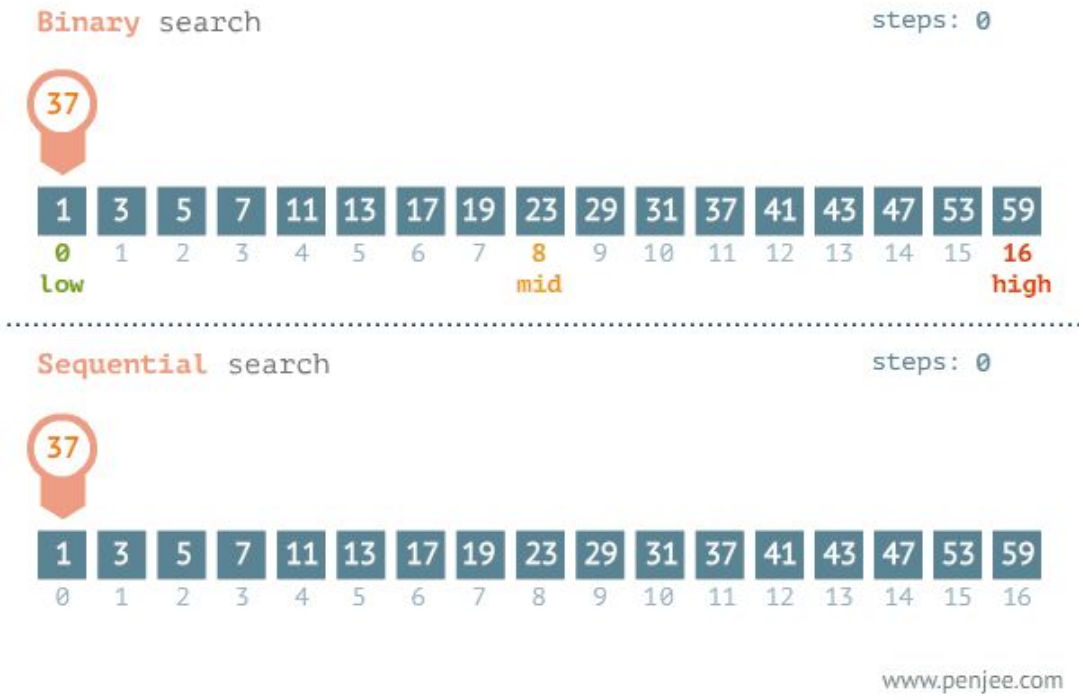
# Binary Search



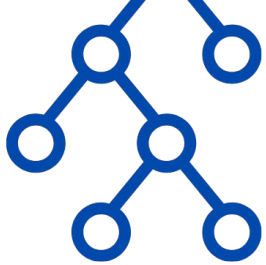
# Learning Objective

1. Understanding the Binary Search Algorithm
2. Understanding why the algorithm’s efficient
3. Understanding the Algorithm's Requirements
4. Exploring Applications and Use Cases.

# Lecture Flow

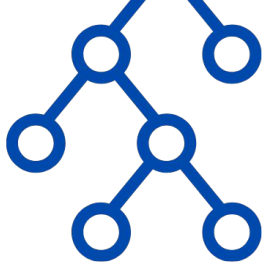
1. Prerequisites
2. Introduction
3. Naive Approach
4. Binary Search
5. Variants
6. Common Pitfalls
7. Quote of the day

# Pre-requisites



1. Array
2. Sorting
3. Loops and conditional statements

# Introduction

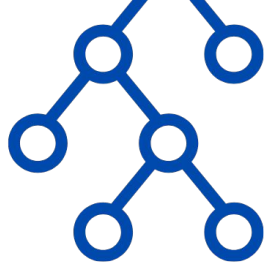


Let’s play a game.

# Introduction

- Binary Search is one of the most fundamental and useful algorithms in Computer Science.
- It describes the process of searching for a specific value in an ordered collection.

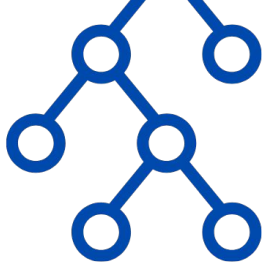
# Naive Approach



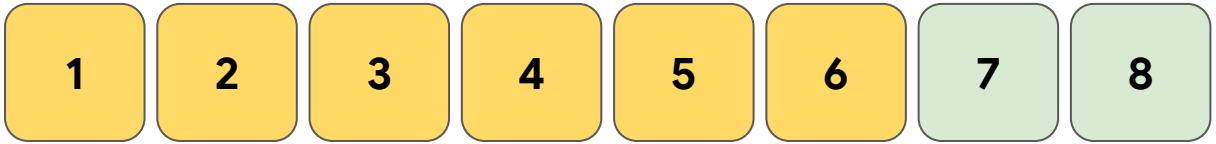
Given a sorted list of numbers search for 6

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

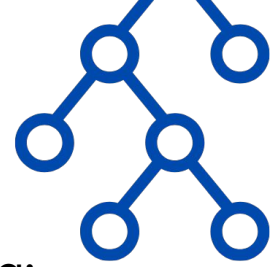
# Naive Approach



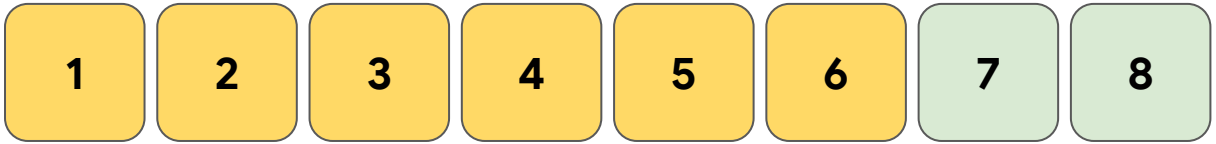
You iterate until we find the number. It takes 6 steps.



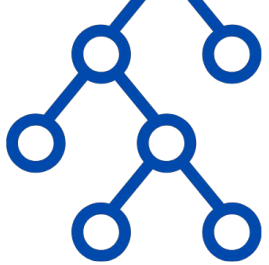

# Naive Approach - Time Complexity



Worst case scenario: when the element to be searched is at the end. Eg: when target is 8. O(n)




# Binary Search

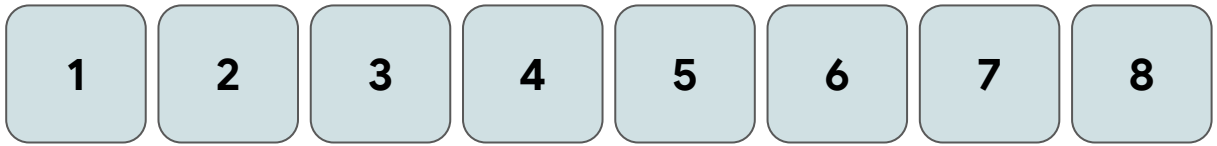


We know the numbers are sorted. Is there a room for optimization?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

# Binary Search

Let's call the region of the list we are looking for the number in the search space. We will start with the whole list as the search space.




# Binary Search

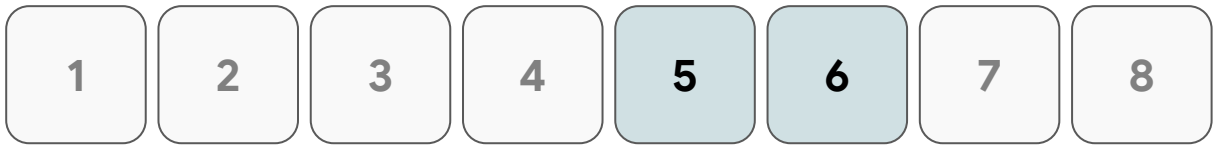
Let's pick a number around the middle. If the picked number is smaller than the target number, we make the search space after the picked number.

Otherwise, we make the search space until this number.


# Binary Search

Let's pick a number around the middle. If the picked number is smaller than the target number, we make the search space after the picked number.

Otherwise, we make the search space until this number.




# Binary Search

Let’s pick a number around the middle. If the picked number is smaller than the target number, we make the search space after the picked number.

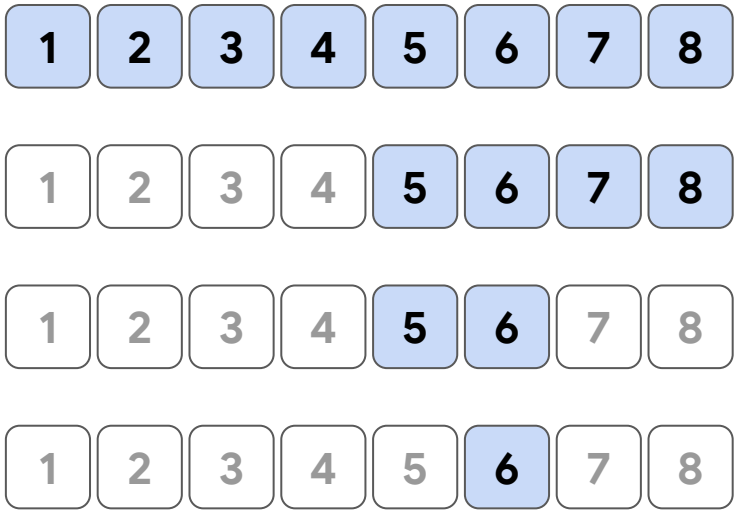
Otherwise, we make the search space until this number : 3 steps


# Practice

**Binary Search**

# Binary Search - Time Complexity

What is the number of steps needed to make the search space size exactly 1?
On each iteration we are halving it.

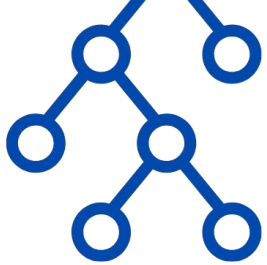



```
1*2*2* ... *2    =  problem size
Number of steps
     Times
```

```
2^(number of steps) = problem size
```

```
Number of steps = log2(problem size)
```

# Note



```
mid = (high + low) / 2
```

Unlike in python, this could result in an overflow, in lower-level languages. So use this instead.

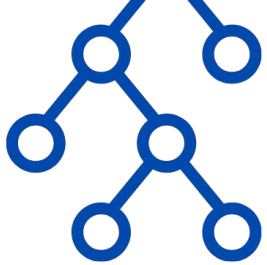
```
mid = low + (high - low) // 2
```

# A more general way of thinking

Not all binary search problems are about finding the right position in a sorted list.

Problem Link

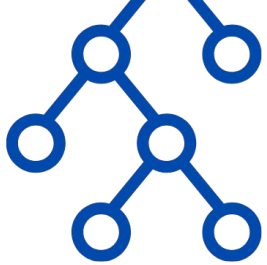
# A more general way of thinking



In this problem we are looking for the first time we get a defect.

| Not Defect | Not Defect | Not Defect | Not Defect | Not Defect | Defect | Defect | Defect |
|---|---|---|---|---|---|---|---|

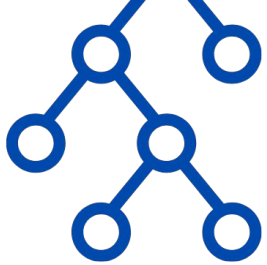
# A more general way of thinking



Cut the search space by half every time

| Not Defect | Not Defect | Not Defect | Not Defect | Not Defect | Defect | Defect | Defect |
|---|---|---|---|---|---|---|---|

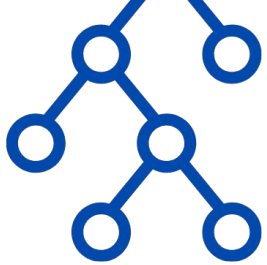
# A more general way of thinking



Cut the search space by half every time

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Not Defect | Not Defect | Not Defect | Not Defect | Not Defect | Defect | Defect | Defect |

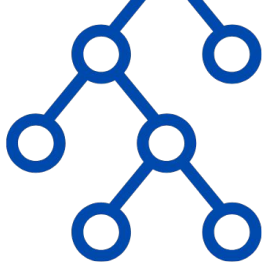
# A more general way of thinking



Cut the search space by half every time

| Not Defect | Not Defect | Not Defect | Not Defect | Not Defect | Defect | Defect | Defect |
|---|---|---|---|---|---|---|---|

# A more general template



```
def binarySearch(low=1, high=1, key = lambda x: True):

    while low < high:
        mid = low + (high - low)//2
        if key(mid):
            high = mid
        else:
            low = mid+1
    return low
```

# Variants

# Variant 1 - Over an input space

This is the variant we have seen, where we search for a particular value in the given input.

In other words, the search space is given explicitly by the problem.

# Variant 1 - Over an input space

## Pair Programming

Problem Link

# Variant 2 - Over an output space

The search is applied over a possible output range. For every choice there is usually a linear check to validate the choice.

This is trickier than the first variant.

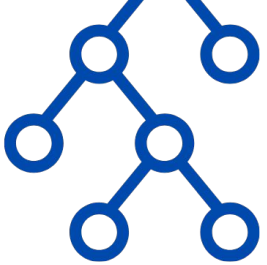
# Variant 2 - Over an output space

## Pair Programming

Problem Link

# Common Pitfalls

# Common Pitfalls



- Not checking the right condition for loop termination;
  `low < high` vs `low <= high` vs `low + 1 < high`
- Off-by-one on the search space
- Unreachable loop-termination condition

# Practice Problems

1. H-Index II - LeetCode
2. Search a 2D matrix
3. Heaters
4. The Meeting Place Cannot Be Changed

# Quote of the Day

“Believe you can and you're halfway there.”

~ ***Theodore Roosevelt***