Introduction

09114319: Data Structures and Algorithms

R. Promkam, Dr. rer. nat.

Department of Mathematics and Computer Science, RMUTT

Outline

- What is an Algorithm?
- Introduction to Binary Search
- Running Time and Big O Notation
- Comparing Algorithms and Growth Rates
- Exercises

What is an Algorithm?

- A set of instructions for accomplishing a task.
- Algorithms are chosen based on efficiency and problem-solving capability.
- Example highlights:
    - Binary Search: Speeds up code by reducing the number of steps.
    - Graph Algorithms: Used for GPS navigation.
    - Dynamic Programming: Write AI algorithms, such as for games.

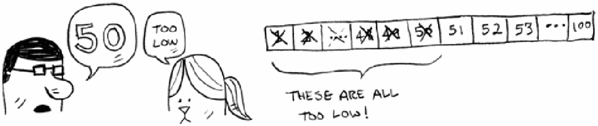
Binary Search

- Suppose you need to find a person in the phone book whose name starts with 'K'.
- Instead of starting from 'A', you start in the middle.
- Binary Search is an efficient algorithm for searching sorted lists.
- It eliminates half the list in each step.

Binary Search Example

- Imagine guessing a number between 1 and 100.

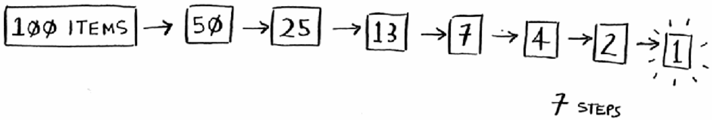
- Start at 50 and adjust based on feedback (too high or too low).



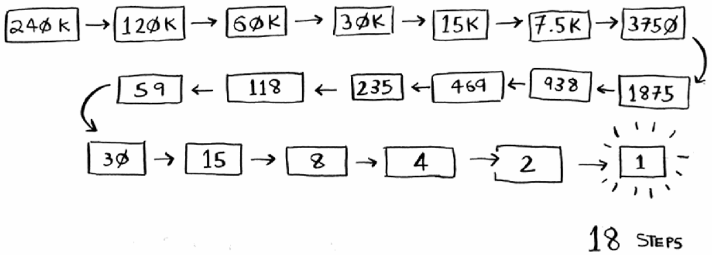
- This process continues until the number is found.

Binary Search Example

- ✎ Maximum number of guesses: 7 for 100 elements.



- ✎ Maximum number of guesses: 7 for 240K elements.



Binary Search in Python

```python
def binary_search(list, item):
    low = 0
    high = len(list) - 1

    while low <= high:
        mid = (low + high) // 2
        guess = list[mid]
        if guess == item:
            return mid
        if guess > item:
            high = mid - 1
        else:
            low = mid + 1

    return None
```

Big O Notation

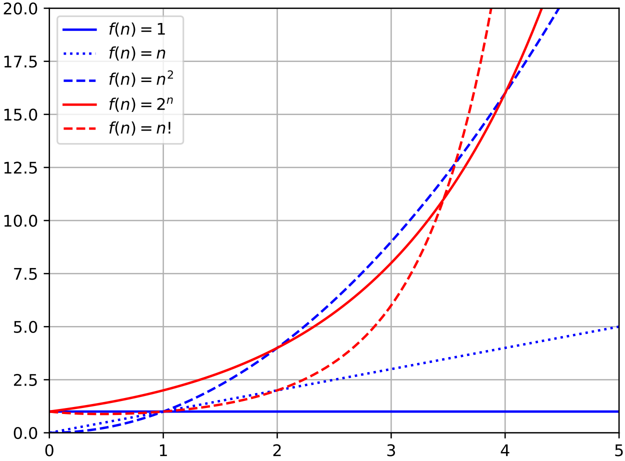
Definition 1

Let f and g be functions defined on some subset of the real numbers. We say that $f \in O(g(n))$ if there exist positive constants C and N such that

$$|f(n)| \leq C \cdot g(n) \quad \text{for all } n \geq N$$

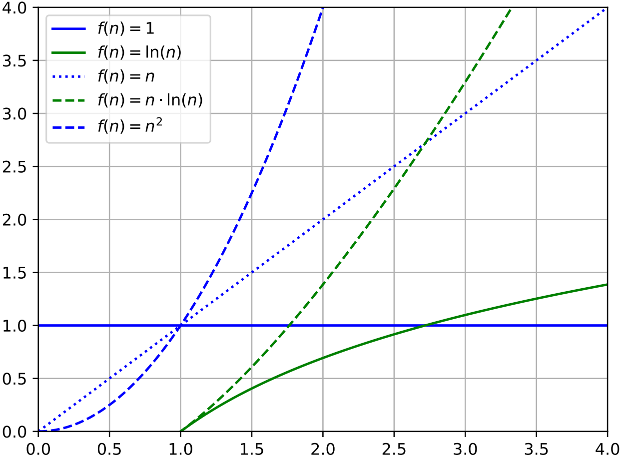
- ✎ The notation $f \in O(g(n))$ means that the growth rate of $f(n)$ is asymptotically bounded above by a constant multiple of $g(n)$.
- ✎ The constants C and N provide a threshold beyond which the inequality holds, showing that $g(n)$ is an upper bound on $f(n)$ as n becomes sufficiently large.

$$1 \prec n \prec n^2 \prec n^3 \prec \ldots \prec 2^n \prec n!$$



Big O Notation

$$1 \prec \ln(n) \prec n \prec n\ln(n) \prec n^2$$



Example 2 (Linear Function)

Let

$$f(n) = 3n + 5, \text{ and } g(n) = n.$$

Show that $f(n) \in O(g(n))$.

Consider the following

$$\begin{aligned} |3n+5| &\leq |3n| + 5 \\ &\leq 3n + n \quad (\text{if } n \geq 5) \\ &= 4n. \end{aligned}$$

So, $N = 5$, and $C = 4$ work.

In conclusion, there exist $C = 4$ and $N = 5$ such that $|f(n)| \leq C \cdot g(n)$ for all $n \geq N$. That is $f(n) = 3n + 5 \in O(n)$.

Example 3 (Quadratic Function)

Let
$$f(n) = 2n^2 - 3n + 4, \text{ and } g(n) = n^2.$$
Show that $f(n) \in O(g(n))$.

Consider the following

$$\begin{aligned} \left|2n^2 - 3n + 4\right| \le \left|2n^2\right| + |3n| + 4 &= 2n^2 + 3\,|n| + 4 \\ &\le 2n^2 + 3n^2 + 4 \qquad (\text{if } n \ge 2) \\ &\le 2n^2 + 3n^2 + n^2 \quad (\text{if } n \ge 2) \\ &= 6n^2. \end{aligned}$$

In conclusion, there exist $C = 6$ and $N = 2$ such that
$|f(n)| \le C \cdot g(n)$ for all $n \ge N$.
That is $f(n) = 2n^2 - 3n + 4 \in O(n^2)$.

Example 4 (Logarithm Function)

Let
$$f(n) = 2\ln(n) + 5n\ln(n), \text{ and } g(n) = n\ln(n).$$
Show that $f(n) \in O(g(n))$.

Consider the following

$$\begin{aligned} |2\ln(n) + 5n\ln(n)| &\leq 2n\ln(n) + 5n\ln(n) \quad (\text{if } n \geq 1) \\ &= 7n\ln(n). \end{aligned}$$

In conclusion, there exist $C = 7$ and $N = 1$ such that
$|f(n)| \leq C \cdot g(n)$ for all $n \geq N$.
That is $f(n) = 2\ln(n) + 5n\ln(n) \in O(n\ln(n))$.

Example 5 (Constant Time – $O(1)$)

```
def first_element(arr):
    return arr[0]
```

The function accesses the first element of the list and returns it. This operation is $O(1)$ because it takes the same amount of time, irrespective of the list size.

Example 6 (Linear Time – $O(n)$)

```
def simple_search(arr, target):
    i = 0
    while i < len(arr):
        if arr[i] == target:
            return i
        else:
            i += 1
    return None
```

This function iterates through the list to find a target element. In the worst case, it has to check every element in the list, which makes it $O(n)$.

Example 7 (Quadratic Time – $O(n^2)$)

```python
def print_pairs(arr):
    for i in arr:
        for j in arr:
            print(f'Pair: ({i}, {j})')
```

This function uses two nested loops to print all possible pairs from the list. If the list has n elements, the number of iterations will be $n \times n = n^2$, making the complexity $O(n^2)$.

Big O Notation

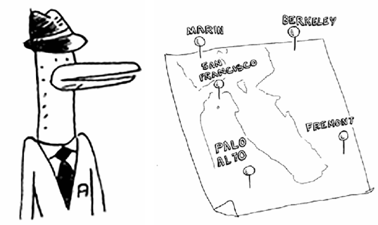
Example 8 (Logartirhm Time – $O(\log_2 n)$)

```python
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return None
```

The binary search algorithm divides the search space by half each time. This divide-and-conquer strategy leads to a running time of $O(\log_2 n)$.

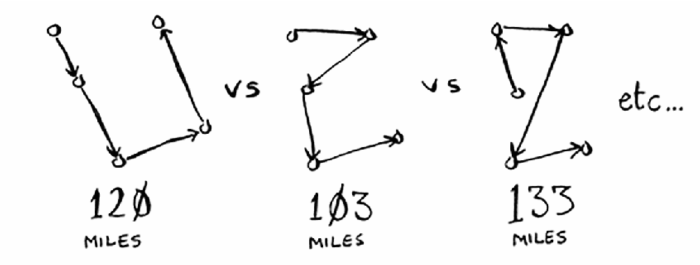
Example 9 (Traveling Salesman Problem: TSP)

TSP involves finding the shortest route that visits a set of cities exactly once and returns to the starting point.



To solve this problem, one approach is to calculate the total distance for all possible orders of visiting the cities and choose the shortest one.

Big O Notation

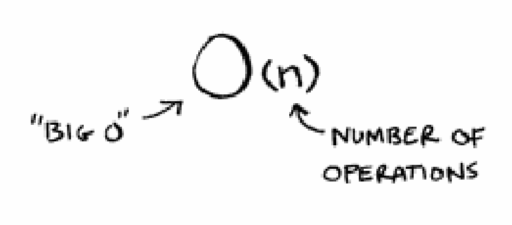


- For n cities, there are $n!$ (factorial) possible permutations.
  - For 5 cities, there are $5! = 120$ possible routes.
  - For 6 cities, there are $6! = 720$ possible routes.
  - As n grows, the number of permutations increases rapidly.

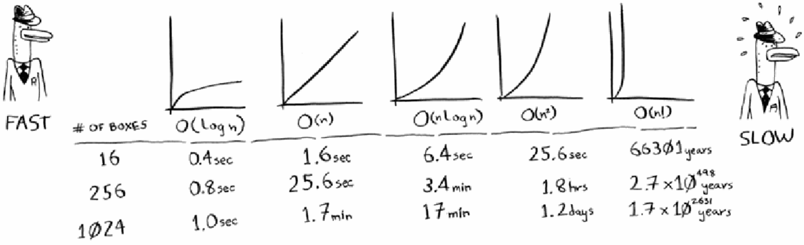
CITIES	OPERATIONS
6	720
7	5040
8	40320
...	...
15	1,307,674,368,000
...	...
30	265,252,859,812,191,058,636,308,480,000,000

- ✎ The complexity of this brute-force solution is $O(n!)$, which makes it impractical for large values of n.
- ✎ The TSP is a well-known example of a problem with exponential growth, making it very computationally expensive as the number of cities increases.

Big O Notation

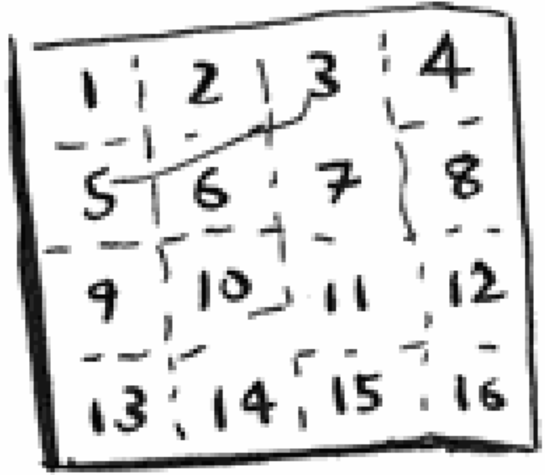


- ✎ Big O notation describes the performance or complexity of an algorithm.
- ✎ It expresses how the running time grows with the input size.

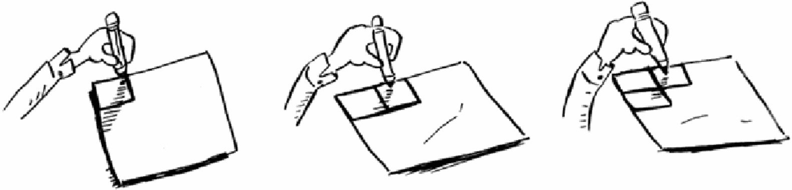


- ✎ Algorithm running times grow at different rates.
- ✎ Simple Search vs. Binary Search:
  - ✎ Simple Search takes $O(n)$ time.
  - ✎ Binary Search takes $O(\log n)$ time.
  - ✎ As the input size grows, Binary Search becomes much faster.

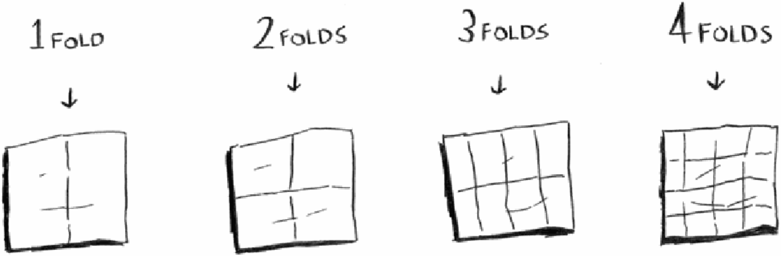
Comparing Growth Rates



Comparing Growth Rates



vs.



Exercises

1. Suppose you have a sorted list of 128 names, and you are using binary search. What is the maximum number of steps it would take?
2. Suppose you double the size of the list. What is the maximum number of steps now?

For the following, what is the running time (in terms of Big O)?

3. You have a name, and you want to find the person's phone number in the phone book.
4. You have a phone number, and you want to find the person's name in the phone book.
5. You want to read the numbers of every person in the phone book.
6. You want to read the numbers of just the people whose last names start with 'A'.

Recap

- ✎ Binary search is much faster than simple search for large datasets.
- ✎ Big O notation helps understand algorithm efficiency.
- ✎ Different algorithms have different growth rates, which significantly impact their performance as the input size grows.