Newton School of Technology

3

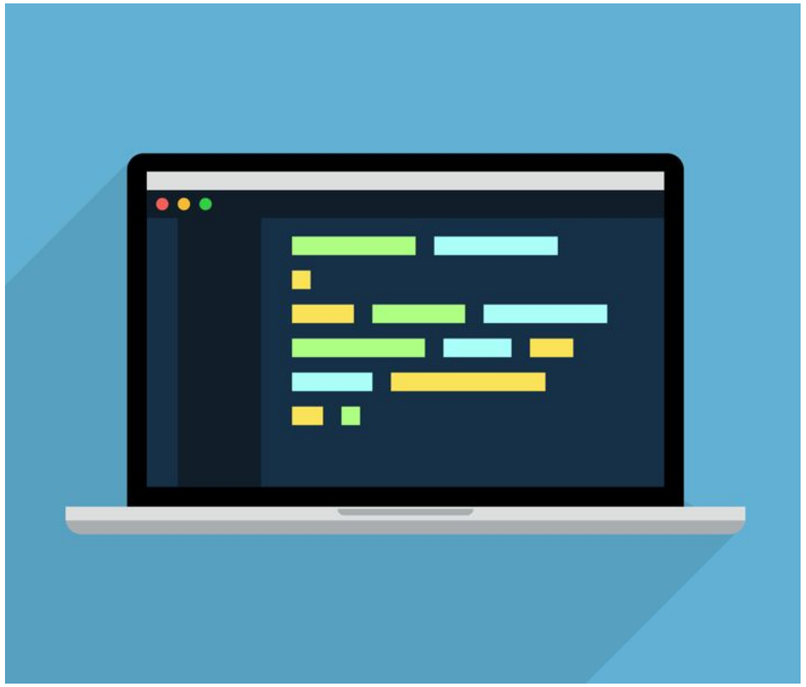
Time complexity

Newton School of Technology

Join the lecture online on your dashboard

Quick Recap :

- 2D List
- Taking input
- Traversing 2D List
- Matrix Multiplication



Newton School of Technology

Pre Read Quiz

Why time complexity?

Newton School of Technology

Why time complexity?

Newton School of Technology

- **To Compare**: Helps choose the best algorithm for a task.
- **For Scale**: Ensures scalability of software with larger inputs.

What is time complexity?

Newton School of Technology

Algo 1:

Old windows machine

n = 10, T = 2 secs

n = 20, T = 4 secs,

n = 30, T = 6 secs

Algo 2:

New windows machine

n = 10, T = 1 secs

n = 20, T = 2 secs,

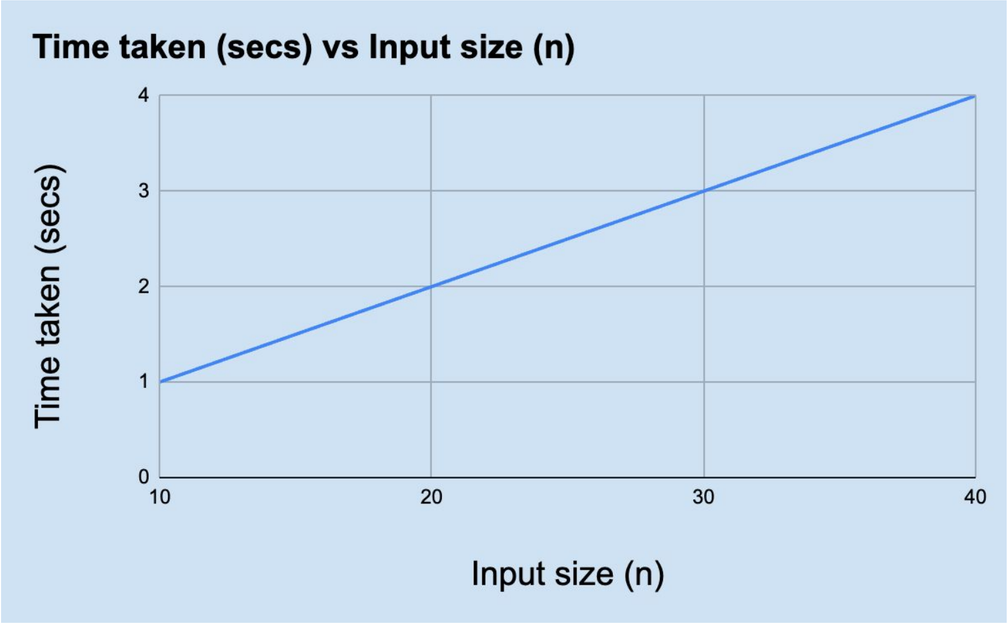
n = 30, T = 3 secs

Which algorithm is better?

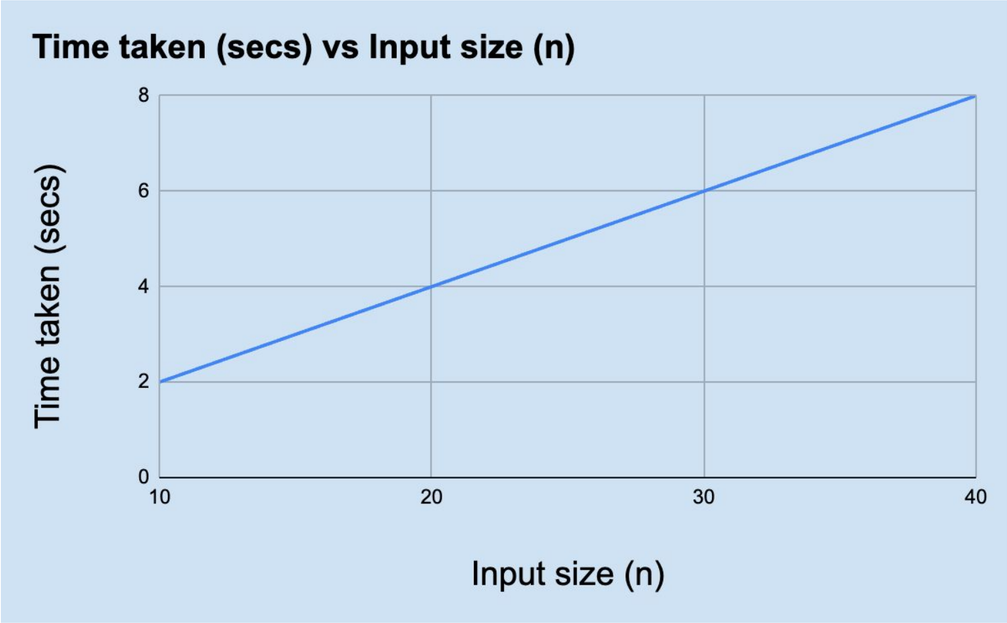
What is time complexity?

Newton School of Technology

Algo 1



Algo 2



Both are equally good.

What is time complexity?

Newton School of Technology

Time complexity != Time taken

How to measure and compare then?

Big O Notation

What is big O?

Newton School of Technology

What is big O?

Newton School of Technology

Big O Notation is a **mathematical concept** used to describe the **upper bound** of an algorithm's **time or space complexity**. It provides a way to express how the **runtime** or space requirements of an algorithm **grow relative to the input size n** as n approaches infinity.

In simple terms, Big O focuses on the **worst-case scenario** and **disregards constant factors or lower-order terms** to analyze scalability.

Example 1:

```
for _ in range(5):
    print("Newton School of Technology")
```

Number of Operations : 5
Time complexity : O(5)

Example 2:

```
for _ in range(500):
    print("Newton School of Technology")
```

Number of Operations : 500
Time complexity : O(500)

Example 3:

Newton School of Technology

```
for _ in range(n):
    print("Newton School of Technology")
```

Number of Operations : n
Time complexity : O(n)

Example 4:

```
# Loop from 0 to n
for i in range(0,n):
    sum += i
```

Number of Operations : n
Time complexity : O(n)

Example 5:

```
def add(a,b):
    return a+b
```

Number of Operations : 1
Time complexity : O(1)

What is time complexity?

Newton School of Technology

Time complexity is a measure used to describe the amount of time an algorithm takes to complete as a function of the size of the input. **It provides an estimate of how an algorithm's runtime grows with the input size**, helping us understand its efficiency.

Example 6:

Newton School of Technology

```
# First loop: Calculates the sum of numbers from 0 to n
for i in range(n):
    sum1 += i

# Second loop: Calculates the sum of squares of no's from 0 to n
for i in range(n):
    sum2 += i * i
```

Time complexity :

Example 6:

Newton School of Technology

```
# First loop: Calculates the sum of numbers from 0 to n
for i in range(n):
    sum1 += i

# Second loop: Calculates the sum of squares of no's from 0 to n
for i in range(n):
    sum2 += i * i
```

Time complexity : O(n) + O(n) => O(2n)

Important Points :

Newton School of Technology

- **Big - O** notation is used to represent time complexity in Worst case

- **Ignore constants** while finding the time complexity

Example 6:

Newton School of Technology

```
# First loop: Calculates the sum of numbers from 0 to n
for i in range(n):
    sum1 += i

# Second loop: Calculates the sum of squares of no's from 0 to n
for i in range(n):
    sum2 += i * i
```

Time complexity : O(n) + O(n) => O(2n)

Time complexity : O(n) + O(n) => O(2n) => O(n)

Example 8:

Newton School of Technology

```python
n = int(input("Enter a number: "))
sum1 = 0
sum2 = 0

# First loop: Calculates the sum of numbers from 0 to n
for i in range(n + 1):
    sum1 += i

# Second loop: Calculates the sum of squares of numbers from 0 to n
for i in range(n + 1):
    sum2 += i * i

print(sum1)
print(sum2)
```

Time complexity :

Example 9:

Newton School of Technology

```
# First loop: Calculates the sum of numbers from 0 to n
for i in range(n + 1):
    sum1 += i

# Second loop: Calculates the sum of squares of numbers from 0 to n
for i in range(n + 1):
    sum2 += i * i

# Third  loop: Calculates the sum of cubes of numbers from 0 to n
for i in range(n + 1):
    sum3 += i * i * i
```

Time complexity : O(n) + O(n) + O(n) => O(3n)
Time Complexity : O(n) Ignore constants

Newton School of Technology

Let’s go bit more deep!

Example 10:

Newton School of Technology

```
# First loop: Calculates the sum of numbers from 0 to n
for i in range(n + 1):
    sum1 += i

# Second loop: Calculates the sum of squares of numbers from 0 to n
for i in range(n + 1):
    sum2 += i * i

# Third  loop: Calculates the sum of cubes of numbers from 0 to n
for i in range(n + 1):
    sum3 += i * i * i
```

Time complexity : O(n) + O(n) + O(n) => O(3n)
Time Complexity : O(n) Ignore constants

Example 11:

```python
def nested_loop_example(n):
    count = 0
    for i in range(n):
        for j in range(n):
            count += 1
    return count
```

Time Complexity :

Example 11:

```python
def nested_loop_example(n):
    count = 0
    for i in range(n):
        for j in range(n):
            count += 1
    return count
```

Time Complexity : $O(n^2)$

Example 12:

```
sum1 = 0
sum2 = 0

for i in range(n):
    for j in range(n):
        sum1 += i + j

for i in range(n):
    sum2 += i
```

Time Complexity :

Example 12:

```
sum1 = 0
sum2 = 0

for i in range(n):
    for j in range(n):
        sum1 += i + j

for i in range(n):
    sum2 += i
```

Time Complexity : $O(n^2) + O(n)$

Quadratic Time Complexity : $O(n^2)$

Newton School of Technology

Execution time grows proportional to the square of the size of the input (n^2)

Important Points :

Newton School of Technology

- **Big - O** notation is used to represent time complexity in Worst case

- **Ignore constants** while finding the time complexity

- Only **higher order terms** to be considered

Example 12:

```
sum1 = 0
sum2 = 0

for i in range(n):
    for j in range(n):
        sum1 += i + j

for i in range(n):
    sum2 += i
```

Time Complexity : $O(n^2) + O(n)$ => ?

Example 12:

```
sum1 = 0
sum2 = 0

for i in range(n):
    for j in range(n):
        sum1 += i + j

for i in range(n):
    sum2 += i
```

Time Complexity : $O(n^2) + O(n) => O(n^2)$

Example:

Newton School of Technology

```python
def count_triplets_and_sum(arr, target):
    n = len(arr)
    count = 0

    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if arr[i] + arr[j] + arr[k] == target:
                    count += 1

    total_sum = 0
    for i in range(n):
        total_sum += arr[i]

    return count, total_sum
```

```python
def count_triplets_and_sum(arr, target):
    n = len(arr)
    count = 0

    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if arr[i] + arr[j] + arr[k] == target:
                    count += 1

    total_sum = 0
    for i in range(n):
        total_sum += arr[i]

    return count, total_sum
```

Newton School of Technology

Number of operations: $c_1 * n^3 + c_2 * n$ where c_1, c_2 are constants

Time Complexity : $O(n^3)$

Newton School of Technology

Quiz: What is the time complexity?

What is the time complexity? - 1

Newton School of Technology

```python
def count_element(arr, target):
    count = 0
    for num in arr:
        if num == target:
            count += 1
    return count
```

Time Complexity : O(n)

What is the time complexity? - 2

Newton School of Technology

```python
def sum_of_first_n_natural_numbers(n):
    if n <= 0:
        return 0

    # Formula to calculate the sum of first 'n' natural numbers
    sum_n = (n * (n + 1)) // 2

    return sum_n
```

Time Complexity : O(1)

What is the time complexity? - 3

Newton School of Technology

```
def nested_loop_example(n):
    count = 0
    for i in range(n):
        for j in range(n):
            if j == 0:
                break
            count += 1
    return count
```

Time Complexity : O(n)

What is the time complexity? - 4

Newton School of Technology

```python
def find_time(n):
    total = 0
    for i in range(n):      # Outer loop
        for j in range(i, i + 2):      # Inner loop
            total += j
    return total
```

Time Complexity : O(n)

What is the time complexity? - 4

Newton School of Technology

- **Explanation:**
  - **Outer Loop:** Runs n times.
  - **Inner Loop:** Runs **exactly twice**
- **Time Complexity:**
  - Outer loop: n iterations.
  - Inner loop: 2 iterations per outer loop.
  - **Total iterations**: 2*n, which **simplifies** to O(n).

What is the time complexity? - 5

Newton School of Technology

```python
def divide_until_one(n):
    steps = 0
    while n > 1:
        n //= 2  # Divide n by 2
        steps += 1
    return steps
```

Time Complexity :

What is the time complexity? - 5

Newton School of Technology

```python
def divide_until_one(n):
    steps = 0
    while n > 1:
        n //= 2  # Divide n by 2
        steps += 1
    return steps
```

Time Complexity : O(logn)

What is the time complexity? - 5

Newton School of Technology

When you divide a number n by a constant (e.g., 2) repeatedly, the size of n decreases at an exponential rate. For example:

- First division: $n \to n/2$
- Second division: $n/2 \to n/4$
- Third division: $n/4 \to n/8$

At each step, the value of n is halved. This is equivalent to saying:

$$n \to \frac{n}{2^k}$$

where k is the number of times you've divided the number.

What is the time complexity? - 5

Newton School of Technology

The process stops when n becomes less than or equal to 1. To find how many steps (k) are required, solve:

$$\frac{n}{2^k} \leq 1$$

Simplify:

$$n \leq 2^k$$

Take the logarithm base 2 on both sides:

$$\log_2 n \leq k$$

Thus, the number of divisions (k) required is approximately $\log_2 n$.

What is the time complexity? - 6

Newton School of Technology

```
def triple_nested_loop_example(n):
    count = 0
    for i in range(n):
        for j in range(n):
            for k in range(n):
                count += 1
    return count
```

Time Complexity : $O(n^3)$

What is the time complexity? - 6

Newton School of Technology

```python
def fib(n):
    if n<=1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Time Complexity : $O(2^n)$

What is the time complexity? - 7

Newton School of Technology

```
j = 0
sum = 0
for i in range(n):
    while(j<n) :
        sum+=j
        j+=1
print(sum)
```

Time Complexity : O(n)

Newton School of Technology

Algorithm Analysis :

Algorithm Analysis :

Newton School of Technology

Algorithm 1 : O(n)

Algorithm 2 : O(n^2)

Algorithm 3 : O(1)

Ans : Algorithm ? is best

Algorithm Analysis :

Newton School of Technology

Algorithm 1 : O(n)

Algorithm 2 : O(n^2)

Algorithm 3 : O(1)

Ans : Algorithm 3 is best

Algorithm Analysis :

Newton School of Technology

Algorithm 1 : O(n^3)

Algorithm 2 : O(n!)

Algorithm 3 : O(log(n))

Ans : Algorithm 3 is best

Algorithm Analysis :

Newton School of Technology

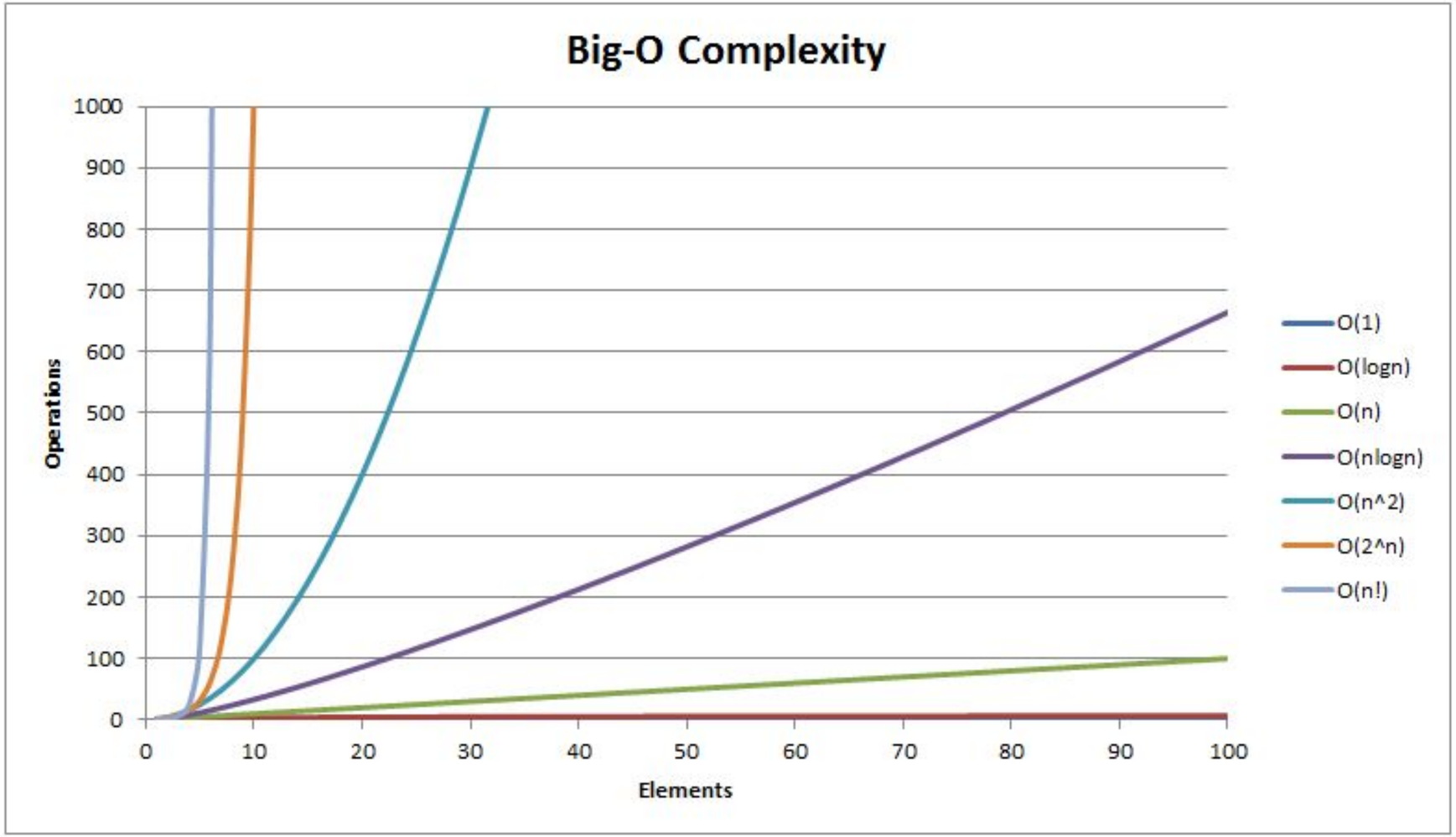
Algorithm 1 : O(n^3)

Algorithm 2 : O(n!)

Algorithm 3 : O(log(n))

Ans : Algorithm 3 is best

Algorithm Analysis :



Newton School of Technology

Finding Factors :

Finding pairs :

Newton School of Technology

1. **Factors Always Come in Pairs**

   For every number i that is a factor of n, there is a corresponding n/i such that:

   $$i \times (n/i) = n$$

   Examples for $n = 36$:

   - $i = 1, n/i = 36$ → Pair: $(1, 36)$
   - $i = 2, n/i = 18$ → Pair: $(2, 18)$
   - $i = 3, n/i = 12$ → Pair: $(3, 12)$
   - $i = 4, n/i = 9$ → Pair: $(4, 9)$
   - $i = 6, n/i = 6$ → Pair: $(6, 6)$ (special case: $i = n/i$)

   After $i = 6$, the pairs start **repeating** because the second factor in each pair (n/i) gets smaller than i.

Summary :

Newton School of Technology

Thank You!

Feedback Time!