Lecture 1

Asymptotic Analysis

Algorithms

- **Definition**:
  - A well-defined list of steps for solving a particular problem.

- **Objective**:
  - Develop efficient algorithm for the processing of our data.

- Two major **measures of the efficiency** of an algorithm are:
  - **Time**
  - **Space**

Algorithms

- The choice of data structure involves a time-space trade-off.

- ***Time-Space Trade-off:***
  - One algorithm may require less memory space but may take more time to complete its execution.

  - On the other hand, the other algorithm may require more memory space but may take less time to run to completion.

  - Thus, we have to sacrifice one at the cost of other.

  - In other words, there is **Space-Time trade-off** between algorithms.

Control Structures Used In Algorithms

- Broadly speaking, an algorithm may employ one of the following control structures:

  a) **Sequence:** By sequence, we mean that each step of an algorithm is executed in a specified order

  b) **Decision:** A decision statement can also be stated in the following manner –

```
IF condition
        Then process
```

or

```
IF condition
        Then process1
ELSE process2
```

  c) **Repetition:** It can be implemented using constructs such as `while`, `do-while`, and `for loops`.

Control Structures Used In Algorithms

```
Step 1: Input first number as A
Step 2: Input second number as B
Step 3: SET SUM = A+B
Step 4: PRINT SUM
Step 5: END
```

Algorithm to add two numbers

```
Step 1: Input first number as A
Step 2: Input second number as B
Step 3: IF A = B
              PRINT "EQUAL"
        ELSE
              PRINT "NOT EQUAL"
        [END OF IF]
Step 4: END
```

Algorithm to test the equality of two numbers

```
Step 1: [INITIALIZE] SET I = 1, N = 10
Step 2: Repeat Steps 3 and 4 while I<=N
Step 3: PRINT I
Step 4: SET I = I+1
        [END OF LOOP]
Step 5: END
```

Algorithm to print first 10 natural numbers

Control Structures Used In Algorithms

Try yourself

- Write an algorithm for swapping two values.
- Write an algorithm to find the larger of two numbers.
- Write an algorithm to find the sum of first N natural numbers.

- STEP 1: Input the two numbers as A and B
- STEP 2: Set C = A
- STEP 3: Set A = B
- STEP 4: Set B = C
- STEP 5: Print A and B
- STEP 6: END

Rate of Growth

- Consider the example of buying *Laptop* and *Earphone:*

  Cost = cost_of_laptop + cost_of_earphone

  Cost ≈ cost_of_laptop (approximation)

- Example: $\boldsymbol{n^2 + 4n + 6 \approx n^2}$

Rate of Growth

- The low order terms in a function are relatively insignificant for **large** *n*

- We say that $n^2 + 4n + 6$ and n^2 have almost the same **rate of growth**. Therefore, $\boldsymbol{n^2 + 4n + 6 = O(n^2)}$

n	n^2	$4n$	6	$n^2 + 4n + 6$
1	1	4	6	11
10	100	40	6	146
20	400	80	6	486
30	900	120	6	1026
40	1600	160	6	1766
50	2500	200	6	2706
60	3600	240	6	3846
70	4900	280	6	5186
80	6400	320	6	6726
90	8100	360	6	8466
100	10000	400	6	10406

Rate of Growth

- When we study algorithms, we are interested in characterizing them according to their efficiency.

- **We are usually interesting in the *order of growth* of the running time of an algorithm, not in the *exact running time.*** This is also referred to as the ***asymptotic running time*.**

- ***Asymptotic notation*** gives us a method for classifying functions according to their rate of growth.

Complexity of Algorithms

- The **complexity** of and algorithm is **the function f(n)** which gives the running time and/or storage space requirement of the algorithm in terms of the size **n** of the input data.

- Frequently, space complexity = **Cn**

- Unless otherwise state or implied, **the term "complexity" shall refer to the running time of the algorithm.**

Worst-case, Average-case, Best-case Time Complexity

- **Worst-case running time**
  - ✓ This denotes the behaviour of an algorithm with respect to the worst possible case of the input instance.
  - ✓ The worst-case running time of an algorithm is an upper bound on the running time for any input.
  - ✓ Therefore, having the knowledge of worst-case running time gives us an assurance that the algorithm will never go beyond this time limit.

Worst-case, Average-case, Best-case Time Complexity

- **Average-case running time**
  - ✓ The average-case running time of an algorithm is an estimate of the running time for an ‘average’ input.
  - ✓ It specifies the expected behaviour of the algorithm when the input is randomly drawn from a given distribution.
  - ✓ Average-case running time assumes that all inputs of a given size are equally likely.

Worst-case, Average-case, Best-case Time Complexity

- **Best-case running time**
  - ✓ The term ‘best-case performance’ is used to analyse an algorithm under optimal conditions.
  - ✓ For example, the best case for a simple linear search on an array occurs when the desired element is the first in the list.
  - ✓ However, while developing and choosing an algorithm to solve a problem, we hardly base our decision on the best-case performance.
  - ✓ It is always recommended to improve the average performance and the worst-case performance of an algorithm.

Expressing Time and Space Complexity

- The time and space complexity can be expressed using a function **f(n)** where **n** is the input size for a given instance of the problem being solved. Expressing the complexity is required when

  1. We want to predict the rate of growth of complexity as the input size of the problem increases.
  2. There are multiple algorithms that find a solution to a given problem and we need to find the algorithm that is most efficient.

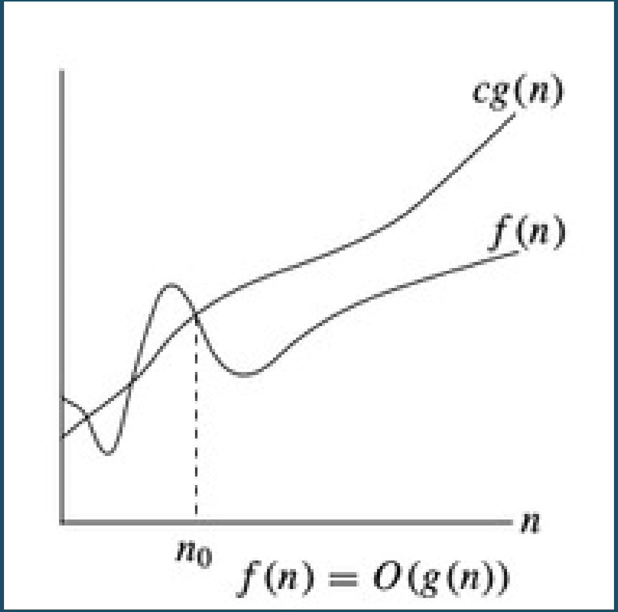
- The most widely used notation to express this function f(n) is the **Big O notation**. It provides the upper bound for the complexity.

Big O Notation

- Big-O is an Asymptotic Notation for the worst case, or ceiling of growth for a given function. It provides us with an asymptotic upper bound for the growth rate of the runtime of an algorithm.

- **f(n) = O(g(n)) that is, f of n is Big–O of g of n if there exist positive constants c and n_0 such that f(n) ≤ cg(n) for all $n \geq n_0$.**

- It means that for large amounts of data, f(n) will grow no more than a constant factor than g(n).



Big O Notation

- **Example:** Show that 30n+8 is O(n).

- Here we have $f(n)$ = 30n+8, and $g(n) = n$
- We need to prove that $\boldsymbol{30n + 8 \leq cn}$ for some constant c.
- Let, c = 31. Then we have

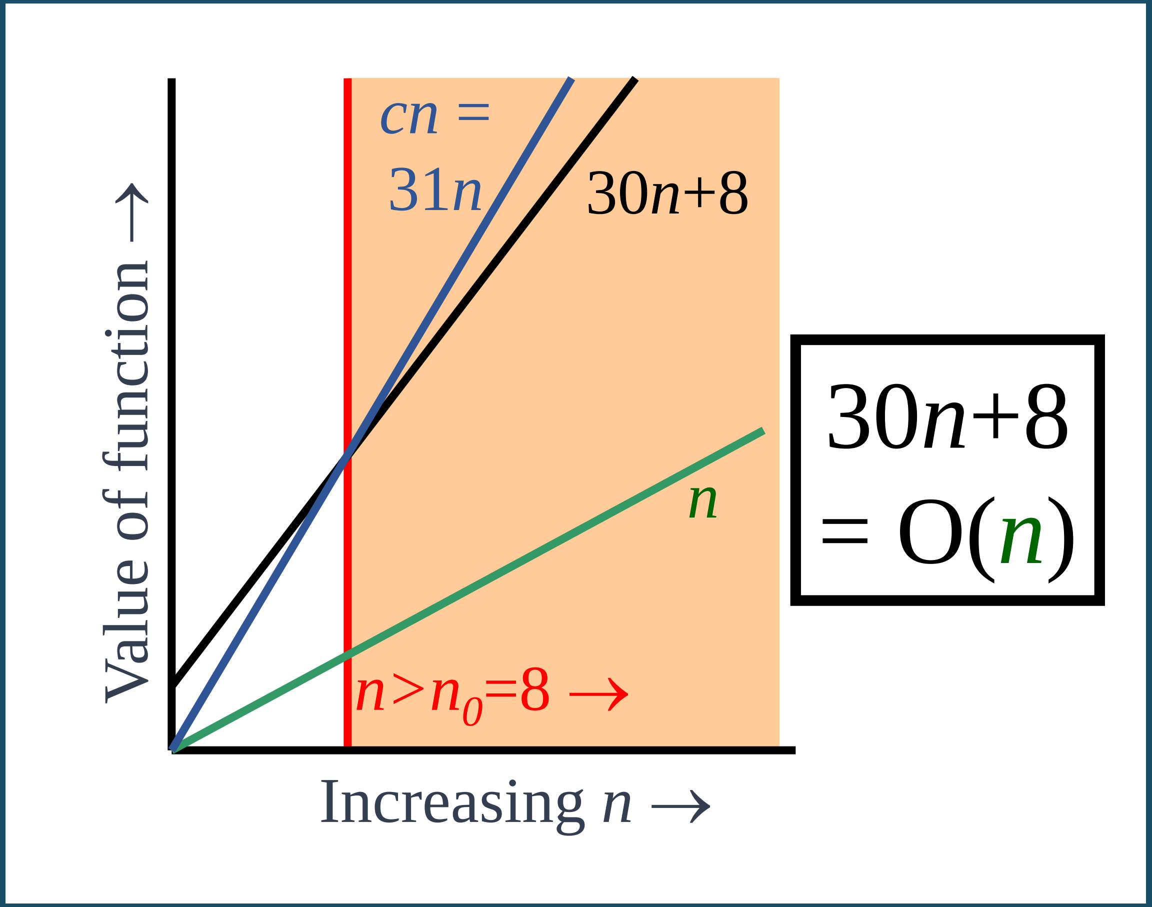
$$30n + 8 \leq 31n \text{ for all } n \geq 8.$$

- Therefore,

$$30n + 8 = \mathrm{O}(n) \text{ with } \mathrm{c} = 31 \text{ and } n_0 = 8$$ ***(Proved)***

Big O Notation

- Note 30*n*+8 isn't less than *n anywhere* (*n*>0).
- It isn't even less than 31*n* *everywhere*.
- But it *is* less than 31*n* everywhere to the right of *n*=8.



Big O Notation

- **Example:** Is $3n + 2 = O(n)$?

- Here we have $f(n) = 3n + 2$, and $g(n) = n$
- We need to prove that $\mathbf{3n + 2 \leq cn}$ for some constant c.

- We notice that when c = 4, we have $3n + 2 \leq 4n$ for all $n \geq 2$.
- Therefore,

$$f(n) = O(g(n))$$

Or, $3n + 2 = O(n)$

- with $\mathrm{c} = 4$ and $n_0 = 2$

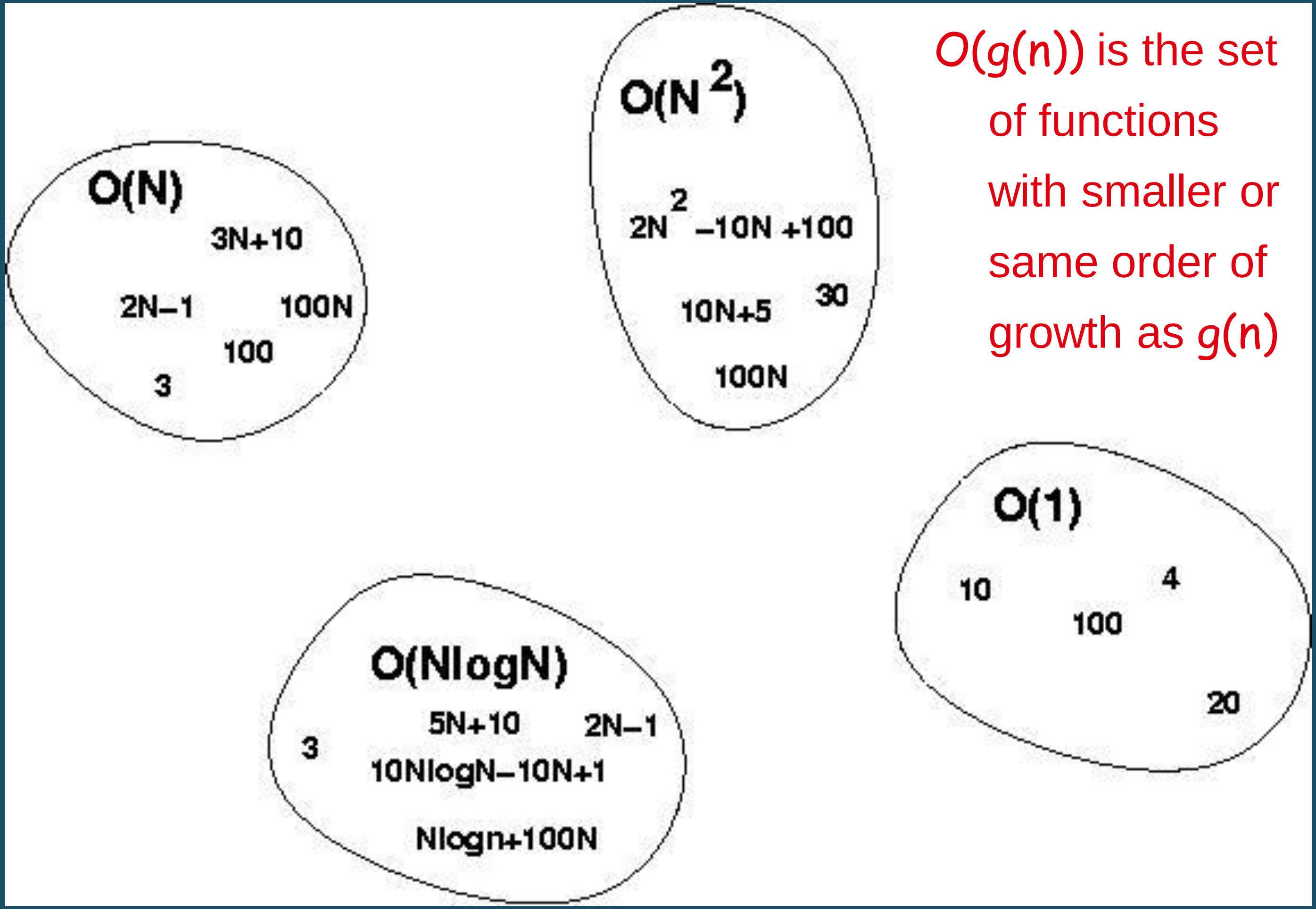
Big O Notation

- **Example:** Is $n^2 + n = O(n^3)$?

- Here we have $f(n) = n^2 + n$, and $g(n) = n^3$
- Notice that if $n \geq 1$, we have $n \leq n^3$.
- Also, notice that if $n \geq 1$, we have $\mathrm{n}^2 \leq n^3$
- Therefore,
$$n^2 + n \leq n^3 + n^3 = 2n^3$$
- We have just shown that
$$n^2 + n \leq 2n^3 \text{ for all } n \geq 1$$
- Thus, we have shown that $n^2 + n = O(n^3)$ with $\mathbf{c = 2 \text{ and } \boldsymbol{n_0} = 1}$

Big O Notation

Big O visualization



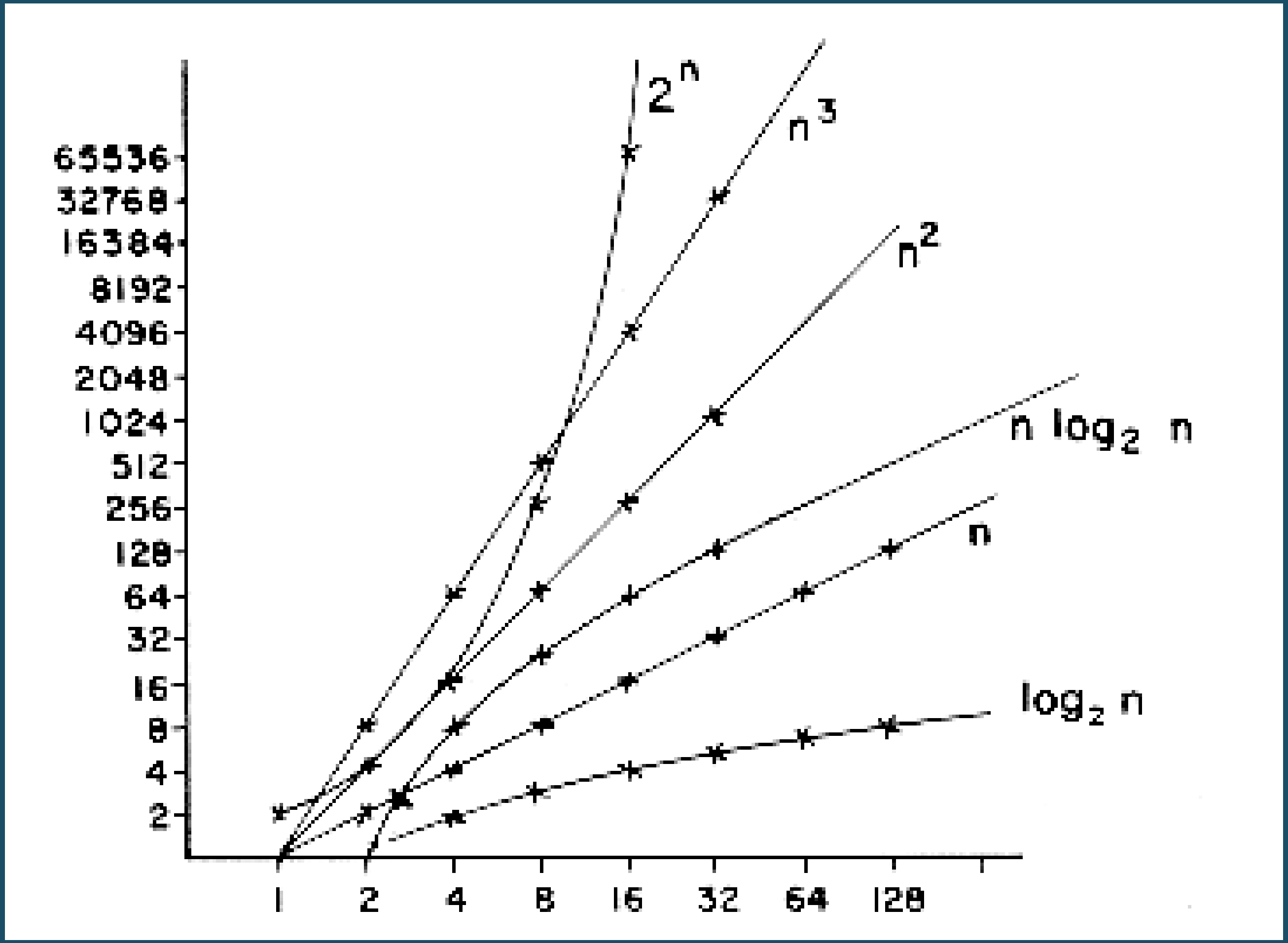
$O(g(n))$ is the set of functions with smaller or same order of growth as $g(n)$

Big O Notation

According to the Big O notation, we have five different categories of algorithms:

- **Constant time algorithm:** running time complexity given as O(1)
- **Linear time algorithm:** running time complexity given as O(n)
- **Logarithmic time algorithm:** running time complexity given as O(log n)
- **Polynomial time algorithm:** running time complexity given as O($\boldsymbol{n^k}$) where k > 1
- Ex**ponential time algorithm:** running time complexity given as O($\boldsymbol{2^n}$)

Common Rates of Growth



Related Readings

- ➢ Data Structures (Seymour Lischutz)
  - Chapter 2