# COL202: Discrete Mathematical Structures

Ragesh Jaiswal, CSE, IIT Delhi

## Algorithms

# Introduction

- How do we describe an algorithm?
  - Using a pseudocode.
- What are the desirable features of an algorithm?
  1. It should be correct.
     - We use proof of correctness to argue correctness.
  2. **It should run fast.**
- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?

# Introduction

- Given two algorithms A1 and A2 for a problem, how do we decide which one runs faster?
  - Idea#1: Implement them on some platform, run and check.

# Introduction

- Given two algorithms A1 and A2 for a problem, how do we decide which one runs faster?
    - Idea#1: Implement them on some platform, run and check.
    - The speed of programs P1 (implementation of A1) and P2 (implementation of A2) may depend on various factors:
        - Input
        - Hardware platform
        - Software platform
        - Quality of the underlying algorithm

# Introduction

- Idea#1: Implement them on some platform, run and check.
- Let P1 denote implementation of A1 and P2 denote implementation of A2.
- Issues with Idea#1:
  - If P1 and P2 are run on different platforms, then the performance results are incomparable.
  - Even if P1 and P2 are run on the same platform, it does not tell us how A1 and A2 compare on some other platform.
  - There might be infinitely many inputs to compare the performance on.
  - Extra burden of implementing *both* algorithms where what we wanted was to first figure out which one is better and then implement just that one.
- So, what we need is a platform independent way of comparing algorithms.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution:
  - Any algorithm is expressed in terms of basic operations such as *assignment, method call, arithmetic, comparison*.
  - For a fixed input, we will count the number of these basic operations in our algorithm. Suppose the number of these operations is $b$.
  - We will assume that the amount of time required to execute these basic operations is at most some constant $T$ which is independent of the input size.
  - The running time of the algorithm will be at most $(b \cdot T)$.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution:
  - Any algorithm is expressed in terms of basic operations such as *assignment, method call, arithmetic, comparison*.
  - For a fixed input, we will count the number of these basic operations in our algorithm. Suppose the number of these operations is $b$.
  - We will assume that the amount of time required to execute these basic operations is at most some constant $T$ which is independent of the input size.
  - The running time of the algorithm will be at most $(b \cdot T)$.
  - **But, what about other inputs**? We are interested in measuring the performance of an algorithm and not performance of an algorithm on a given input.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- <u>Solution</u>: Count the number of basic operations.
  - How do we measure performance for all inputs?

**Example**

`FindPositiveSum`$(A, n)$
- $sum \leftarrow 0$
- For $i = 1$ to $n$
  - if $(A[i] > 0)$ $sum \leftarrow sum + A[i]$
- return$(sum)$

- Note that the number of operations grow with the array size $n$.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the number of basic operations.
  - How do we measure performance for all inputs?

**Example**

`FindPositiveSum`$(A, n)$

- $sum \leftarrow 0$
- For $i = 1$ to $n$
  - if $(A[i] > 0) sum \leftarrow sum + A[i]$
- return$(sum)$

- Note that the number of operations grow with the array size $n$.
- Even for all arrays of a fixed size $n$, the number of operations may vary depending on the numbers present in the the array.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the number of basic operations.
  - How do we measure performance for all inputs?

**Example**

`FindPositiveSum`$(A, n)$

- $sum \leftarrow 0$
- For $i = 1$ to $n$
  - if $(A[i] > 0) sum \leftarrow sum + A[i]$
- return$(sum)$

- Note that the number of operations grow with the array size $n$.
- Even for all arrays of a fixed size $n$, the number of operations may vary depending on the numbers present in the array.
- For inputs of size $n$, we will count the number of operations in the worst-case. That is, the number of operations for the worst-case input of size $n$.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the worst-case number of basic operations $b(n)$ for inputs of size $n$ and then analyse how this *function* $b(n)$ behaves as $n$ grows. This is known as worst-case analysis.

# Introduction

- How do we describe an algorithm?
    - Using a pseudocode.
- What are the desirable features of an algorithm?
    1. It should be correct.
        - We use proof of correctness to argue correctness.
    2. **It should run fast.**
- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the worst-case number of basic operations $b(n)$ for inputs of size $n$ and then analyse how this *function* $b(n)$ behaves as $n$ grows. This is known as worst-case analysis.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the worst-case number of basic operations $b(n)$ for inputs of size $n$ and then analyse how this *function* $b(n)$ behaves as $n$ grows. This is known as worst-case analysis.

**Example**

| | |
|---|---|
| `FindPositiveSum`$(A, n)$ | |
| - $sum \leftarrow 0$ | [1 assignment] |
| - For $i = 1$ to $n$ | [1 assignment + 1 comparison + 1 arithmetic]*$n$ |
| - if $(A[i] > 0) sum \leftarrow sum + A[i]$ | [1 assignment + 1 arithmetic + 1 comparison]*n |
| - return$(sum)$ | [1 return] |
| | **Total**: $6n + 2$ |

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the worst-case number of basic operations $b(n)$ for inputs of size $n$ and then analyse how this *function* $b(n)$ behaves as $n$ grows. This is known as worst-case analysis.
- Few observations:
  - Usually, the running time grows with the input size $n$.
  - Consider two algorithm `A1` and `A2` for the same problem. `A1` has a worst-case running time $(100n + 1)$ and `A2` has a worst-case running time $(2n^2 + 3n + 1)$. Which one is better?
    - `A2` runs faster for small inputs (e.g., $n = 1, 2$)
    - `A1` runs faster for all large inputs (for all $n \geq 49$)

# Introduction

- Given two algorithms A1 and A2 for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the worst-case number of basic operations $b(n)$ for inputs of size $n$ and then analyse how this *function* $b(n)$ behaves as $n$ grows. This is known as worst-case analysis.
- Few observations:
  - Usually, the running time grows with the input size $n$.
  - Consider two algorithm A1 and A2 for the same problem. A1 has a worst-case running time $(100n + 1)$ and A2 has a worst-case running time $(2n^2 + 3n + 1)$. Which one is better?
    - A2 runs faster for small inputs (e.g., $n = 1, 2$)
    - A1 runs faster for all large inputs (for all $n \geq 49$)
  - We would like to make a statement independent of the input size. What is a meaningful solution?

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Count the worst-case number of basic operations $b(n)$ for inputs of size $n$ and then analyse how this *function* $b(n)$ behaves as $n$ grows. This is known as worst-case analysis.
- Observations regarding worst-case analysis:
  - Usually, the running time grows with the input size $n$.
  - Consider two algorithm `A1` and `A2` for the same problem. `A1` has a worst-case running time $(100n + 1)$ and `A2` has a worst-case running time $(2n^2 + 3n + 1)$. Which one is better?
    - `A2` runs faster for small inputs (e.g., $n = 1, 2$)
    - `A1` runs faster for all large inputs (for all $n \geq 49$)
  - We would like to make a statement independent of the input size.
  - Solution: Asymptotic analysis
    - We consider the running time for large inputs.
    - `A1` is considered better than `A2` since `A1` will beat `A2` **eventually**.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Do an asymptotic worst-case analysis.

# Introduction

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- Solution: Do an asymptotic worst-case analysis.
- Observations regarding asymptotic worst-case analysis:
  - It is difficult to count the number of operations at an extremely fine level.
  - Asymptotic analysis means that we are interested only in the **rate of growth** of the running time function w.r.t. the input size. For example, note that the rates of growth of functions $(n^2 + 5n + 1)$ and $(n^2 + 2n + 5)$ is determined by the $n^2$ (*quadratic*) term. The lower order terms are insignificant. So, we may as well drop them.

- Given two algorithms `A1` and `A2` for a problem, how do we decide which one runs faster?
- What we need is a platform independent way of comparing algorithms.
- <u>Solution</u>: Do an asymptotic worst-case analysis.
- Observations regarding asymptotic worst-case analysis:
  - It is difficult to count the number of operations at an extremely fine level and keep track of these constants.
  - Asymptotic analysis means that we are interested only in the **rate of growth** of the running time function w.r.t. the input size. For example, note that the rates of growth of functions $(n^2 + 5n + 1)$ and $(n^2 + 2n + 5)$ is determined by the $n^2$ (*quadratic*) term. The lower order terms are insignificant. So, we may as well drop them.
  - The nature of growth rate of functions $2n^2$ and $5n^2$ are the same. Both are quadratic functions. It makes sense to drop these constants too when one is interested in the nature of the growth functions.
  - We need a notation to capture the above ideas.

### Definition (Big-O)

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ (or $f(n) = O(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

- Another short way of saying that $f(n) = O(g(n))$ is "$f(n)$ is **order of** $g(n)$".
- Show that: $8n + 5 = O(n)$.

**Definition (Big-O)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ (or $f(n) = O(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

- Another short way of saying that $f(n) = O(g(n))$ is "$f(n)$ is **order of** $g(n)$".
- Show that: $8n + 5 = O(n)$.
    - For constants $c = 13$ and $n_0 = 1$,we show that $\forall n \geq n_0, 8n + 5 \leq 13 \cdot n$. So, by definition of big-O, $8n + 5 = O(n)$.

## Definition (Big-O)

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ (or $f(n) = O(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

- Another short way of saying that $f(n) = O(g(n))$ is "$f(n)$ is **order of** $g(n)$".
- $g(n)$ may be interpreted as an upper bound on $f(n)$.
- Show that: $8n + 5 = O(n)$.
- Is this true $8n + 5 = O(n^2)$?

**Definition (Big-O)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ (or $f(n) = O(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

- Another short way of saying that $f(n) = O(g(n))$ is "$f(n)$ is **order of** $g(n)$".
- $g(n)$ may be interpreted as an upper bound on $f(n)$.
- Show that: $8n + 5 = O(n)$.
- Is this true $8n + 5 = O(n^2)$? Yes
- $g(n)$ may be interpreted as an *upper bound* on $f(n)$.
- How do we capture *lower bound*?

**Definition (Big-Omega)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ (or $f(n) = \Omega(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \geq c \cdot g(n)$$

**Definition (Big-Omega)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ (or $f(n) = \Omega(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \geq c \cdot g(n)$$

- Show that: $f(n) = \Omega(g(n))$ iff $g(n) = O(f(n))$.

**Definition (Big-O)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ (or $f(n) = O(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

**Definition (Big-Omega)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ (or $f(n) = \Omega(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \geq c \cdot g(n)$$

- How do we say that $g(n)$ is both an upper bound and lower bound for a function $f(n)$? In other words, $g(n)$ is a **tight bound** on $f(n)$.

**Definition (Big-O)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ (or $f(n) = O(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

**Definition (Big-Omega)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ (or $f(n) = \Omega(g(n))$ in short) **iff** there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that:

$$\forall n \geq n_0, f(n) \geq c \cdot g(n)$$

**Definition (Big-Theta)**

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $\Theta(g(n))$ (or $f(n) = \Theta(g(n))$) **iff** $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

- Question: Show that $3n \log n + 4n + 5 \log n$ is $\Theta(n \log n)$.

End