COL106: Data Structures and Algorithms

Ragesh Jaiswal, IITD

- Data Structure: Systematic way of organising and accessing data.
- Algorithm: A step-by-step procedure for performing some task.

Introduction

- How do we describe an algorithm?
  - Using a pseudocode.
- What are the desirable features of an algorithm?
  1. **It should be correct.**
  2. It should run fast.
- How do we argue that an algorithm is correct?

- How do we argue that an algorithm is correct?
  - Proof of correctness: An argument that the algorithm works correctly for **all** inputs.
    - Proof: A valid argument that establishes the truth of a mathematical statement.
- Consider the following algorithm that is supposed to output the sum of elements of an integer array of size n.

Algorithm

`FindSum`(A, n)

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

Introduction

- How do we argue that an algorithm is correct?
  - Proof of correctness: An argument that the algorithm works correctly for **all** inputs.
    - Proof: A valid argument that establishes the truth of a mathematical statement.
- Consider the following algorithm that is supposed to output the sum of elements of an integer array of size n.

Algorithm

FindSum(A, n)
- $sum \leftarrow 0$
- for $i = 1$ to n
  - $sum \leftarrow sum + A[i]$
- return(sum)

- To prove the algorithm correct, let us define the following loop-invariant:
  $P(i)$: At the end of the i^{th} iteration, the variable sum contains the sum of first i elements of the array A.

Introduction

- How do we argue that an algorithm is correct?
  - Proof of correctness: An argument that the algorithm works correctly for **all** inputs.
    - Proof: A valid argument that establishes the truth of a mathematical statement.
- Consider the following algorithm that is supposed to output the sum of elements of an integer array of size n.

Algorithm

FindSum(A, n)
- $sum \leftarrow 0$
- for $i = 1$ to n
  - $sum \leftarrow sum + A[i]$
- return(sum)

- To prove the algorithm correct, let us define the following loop-invariant:
  $P(i)$: At the end of the i^{th} iteration, the variable sum contains the sum of first i elements of the array A.
- How do we prove statements of the form $\forall i, P(i)$?

Introduction

- How do we argue that an algorithm is correct?
  - Proof of correctness: An argument that the algorithm works correctly for **all** inputs.
    - Proof: A valid argument that establishes the truth of a mathematical statement.
- Consider the following algorithm that is supposed to output the sum of elements of an integer array of size n.

Algorithm

FindSum(A, n)
- $sum \leftarrow 0$
- for $i = 1$ to n
  - $sum \leftarrow sum + A[i]$
- return(sum)

- To prove the algorithm correct, let us define the following loop-invariant:
  $P(i)$: At the end of the i^{th} iteration, the variable *sum* contains the sum of first i elements of the array A.
- How do we prove statements of the form $\forall i, P(i)$?Induction

- Proof: A valid argument that establishes the truth of a mathematical statement.
  - The statements used in a proof can include axioms, definitions, the premises, if any, of the theorem, and previously proven theorems and uses rules of inference to draw conclusions.
- A proof technique very commonly used when proving correctness of Algorithms is *Mathematical Induction*.

Definition (Strong Induction)

To prove that $P(n)$ is true for all positive integers, where $P(.)$ is a propositional function, we complete two steps:

- Basis step: We show that $P(1)$ is true.
- Inductive step: We show that for all k, if $P(1), P(2), ..., P(k)$ are true, then $P(k+1)$ is true.

Introduction

Definition (Strong Induction)

To prove that $P(n)$ is true for all positive integers, where $P(.)$ is a propositional function, we complete two steps:

- Basis step: We show that $P(1)$ is true.
- Inductive step: We show that for all k, if $P(1), P(2), ..., P(k)$ are true, then $P(k+1)$ is true.

- Question: Show that for all $n > 0$, $1 + 3 + ... + (2n-1) = n^2$.

- Question: Show that for all $n > 0$, $1 + 3 + ... + (2n - 1) = n^2$.

Proof

- Let $P(n)$ be the proposition that $1 + 3 + 5 + ... + (2n - 1)$ equals n^2.
- Basis step: $P(1)$ is true since the summation consists of only a single term 1 and $1^2 = 1$.
- Inductive step: Assume that $P(1), P(2), ..., P(k)$ are true for any arbitrary integer k. Then we have:

$$\begin{aligned} 1 + 3 + ... + (2(k+1) - 1) &= 1 + 3 + ... + (2k - 1) + (2k + 1) \\ &= k^2 + 2k + 1 \quad \text{(since } P(k) \text{ is true)} \\ &= (k+1)^2 \end{aligned}$$

  This shows that $P(k + 1)$ is true.
- Using the principle of Induction, we conclude that $P(n)$ is true for all $n > 0$. □

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.**

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

- 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.

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

- 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.

End