

ACOL106: Data Structures and Algorithms

Ragesh Jaiswal, IITD

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 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.

Introduction

Big-O Notation

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: $8n + 5 = O(n)$.

Introduction

Big-O Notation

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, $8n + 5 = O(n)$.

Introduction

Big-O Notation

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 $8n + 5 = O(n^2)$?

Introduction

Big-O Notation

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*?

Introduction

Big-O Notation

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:

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

Introduction

Big-O Notation

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

Introduction

Big-O Notation

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 $f(n)$.

Introduction

Big-O Notation

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 $\Theta(n \log n)$.

Introduction

Big-O Notation

- Growth rates:
 - Arrange the following functions in ascending order of growth rate:
 - n
 - $2^{\sqrt{\log n}}$
 - $n^{\log n}$
 - $2^{\log n}$
 - $n / \log n$
 - n^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: Do an **asymptotic worst-case analysis** recording the running time using Big- (O, Ω, Θ) notation.

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.
 - We do an **asymptotic worst-case analysis** noting the running time in Big- (O, Ω, Θ) notation and use it to compare 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.
 - We do an **asymptotic worst-case analysis** noting the running time in Big- (O, Ω, Θ) notation and use it to compare algorithms.

Problem

Given an integer array A with n elements output another array B such that for all i , $B[i] = \sum_{j=1}^i A[j]$. (That is find cumulative sum of elements in A .)

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.
 - We do an **asymptotic worst-case analysis** noting the running time in Big- (O, Ω, Θ) notation and use it to compare algorithms.

Problem

Given an integer array A with n elements output another array B such that for all i , $B[i] = \sum_{j=1}^i A[j]$. (That is find cumulative sum of elements in A .)

Algorithm

CumulativeSum(A, n)

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

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.
 - We do an **asymptotic worst-case analysis** noting the running time in Big- (O, Ω, Θ) notation and use it to compare algorithms.

Problem

Given an integer array A with n elements output another array B such that for all i , $B[i] = \sum_{j=1}^i A[j]$. (That is find cumulative sum of elements in A .)

Algorithm

CumulativeSum(A, n)

- for $i = 1$ to n

- $sum \leftarrow 0$

- for $j = 1$ to i

- $sum \leftarrow sum + A[j]$

- $B[i] \leftarrow sum$

- return(B)

$3n$ operations

n operations

$3 \cdot (1 + 2 + 3 + \dots + n)$ operations

$2 \cdot (1 + 2 + 3 + \dots + n)$ operations

n operations

1 operation (assuming that only reference to the array is returned)

Total: $\frac{1}{2} \cdot (5n^2 + 15n + 2)$ operations

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.
 - We do an **asymptotic worst-case analysis** noting the running time in Big- (O, Ω, Θ) notation and use it to compare algorithms.

Problem

Given an integer array A with n elements output another array B such that for all i , $B[i] = \sum_{j=1}^i A[j]$. (That is find cumulative sum of elements in A .)

Algorithm

CumulativeSum(A, n)	
- for $i = 1$ to n	$2n$ operations
- $sum \leftarrow 0$	n operations
- for $j = 1$ to i	$2 \cdot (1 + 2 + 3 + \dots + n)$ operations
- $sum \leftarrow sum + A[j]$	$2 \cdot (1 + 2 + 3 + \dots + n)$ operations
- $B[i] \leftarrow sum$	n operations
- return(B)	1 operation (assuming that only reference to the array is returned)
Total: $\frac{1}{2} \cdot (5n^2 + 15n + 2)$ operations	

- So, the asymptotic worst-case running time of the above algorithm is $O(n^2)$. Note that we can also say the running time is $\Omega(n^2)$ and $\Theta(n^2)$.

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.
 - We do an **asymptotic worst-case analysis** noting the running time in Big- (O, Ω, Θ) notation and use it to compare algorithms.

Problem

Given an integer array A with n elements output another array B such that for all i , $B[i] = \sum_{j=1}^i A[j]$. (That is find cumulative sum of elements in A .)

Algorithm

CumulativeSum(A, n)	
- for $i = 1$ to n	$2n$ operations
- $sum \leftarrow 0$	n operations
- for $j = 1$ to i	$2 \cdot (1 + 2 + 3 + \dots + n)$ operations
- $sum \leftarrow sum + A[j]$	$2 \cdot (1 + 2 + 3 + \dots + n)$ operations
- $B[i] \leftarrow sum$	n operations
- return(B)	1 operation (assuming that only reference to the array is returned)
	Total: $\frac{1}{2} \cdot (5n^2 + 15n + 2)$ operations

- So, the asymptotic worst-case running time of the above algorithm is $O(n^2)$. Note that we can also say the running time is $\Omega(n^2)$ and $\Theta(n^2)$.
- Can you design a better $O(n)$ (linear-time) algorithm for this problem?

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.
 - We do an **asymptotic worst-case analysis** noting the running time in Big- (O, Ω, Θ) notation and use it to compare algorithms.

Problem

Given an integer array A with n elements output another array B such that for all i , $B[i] = \sum_{j=1}^i A[j]$. (That is find cumulative sum of elements in A .)

Algorithm

BetterCumulativeSum(A, n)	
- $sum \leftarrow 0$	$O(1)$
- for $i = 1$ to n	$O(n)$
- $sum \leftarrow sum + A[i]$	$O(n)$
- $B[i] \leftarrow sum$	$O(n)$
- return(B)	$O(1)$
Total: $O(n)$	

End