

ACOL106: Data Structures and Algorithms

Ragesh Jaiswal, IITD

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

Sorting: Given an integer array A with n elements, sort it.

Algorithm

BubbleSort(A, n)

- for $i = 1$ to $(n - 1)$
 - for $j = 1$ to $(n - i)$
 - if($A[j] > A[j + 1]$) Swap($A, j, j + 1$)

- What is an appropriate loop-invariant for the above algorithm?

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 $\text{Big-}(O, \Omega, \Theta)$ notation and use it to compare algorithms.

Problem

Sorting: Given an integer array A with n elements, sort it.

Algorithm

BubbleSort(A, n)

- for $i = 1$ to $(n - 1)$
 - for $j = 1$ to $(n - i)$
 - if($A[j] > A[j + 1]$) Swap($A, j, j + 1$)

- What is running time of the above algorithm?

- How do Data Structures play a part in making computational tasks efficient?

Introduction

- How do Data Structures play a part in making computational tasks efficient?

Example problem

Maintain a record of students and their scores on some test so that queries of the following nature may be answered:

- Insert: Insert a new record of a student and his/her score.
 - Search: Find the score of a given student.
- Suppose we maintain the information in a 2-dimensional array.
 - How much time does each insert operations take?
 - How much time does each search operation take?

Introduction

- How do Data Structures play a part in making computational tasks efficient?

Example problem

Maintain a record of students and their scores on some test so that queries of the following nature may be answered:

- Insert: Insert a new record of a student and his/her score.
 - Search: Find the score of a given student.
-
- Suppose we maintain the information in a 2-dimensional array.
 - How much time does each insert operations take? $O(1)$
 - How much time does each search operation take? $O(n)$
 - So, if the majority of the operations performed are search operations, then this data structure is perhaps not the right one.

Introduction

- How do Data Structures play a part in making computational tasks efficient?

Example problem

Maintain a record of students and their scores on some test so that queries of the following nature may be answered:

- Insert: Insert a new record of a student and his/her score.
 - Search: Find the score of a given student.
-
- Suppose we maintain the information in a 2-dimensional array such that the array is sorted based on the names (dictionary order).
 - How much time does each insert operations take?
 - How much time does each search operation take?

Introduction

- How do Data Structures play a part in making computational tasks efficient?

Example problem

Maintain a record of students and their scores on some test so that queries of the following nature may be answered:

- Insert: Insert a new record of a student and his/her score.
 - Search: Find the score of a given student.
- Suppose we maintain the information in a 2-dimensional array such that the array is sorted based on the names (dictionary order).
 - How much time does each insert operations take? $O(n)$
 - How much time does each search operation take? $O(\log n)$
using **Binary Search**
 - In this case, if the majority of the operations performed are insert operations, then the previous one is better.

Introduction

Digression: Binary Search

Problem

Given a sorted array A containing n integers and an integer x , check if x is present in A .

Introduction

Digression: Binary Search

Problem

Given a sorted array A containing n integers and an integer x , check if x is present in A .

Algorithm

BinarySearch-v1(x, A, n)

- If A has no elements, then return("not present")
- Let mid denote the *middle* index of the array (i.e., $mid = \lfloor n/2 \rfloor$)
- If $(A[mid] = x)$, then return("present")
- Let A_L denote the *left-half* of the array and A_R denote the *right-half* of the array
- If $(x < A[mid])$
 - Search x in A_L
- else
 - Search x in A_R

Introduction

Digression: Binary Search

Problem

Given a sorted array A containing n integers and an integer x , check if x is present in A .

Algorithm

`BinarySearch-v2( $x, A, n$ )`

- If ($n \leq 0$), then return("not present")
- $mid \leftarrow \lfloor n/2 \rfloor$
- If ($A[mid] = x$), then return("present")
- $A_L \leftarrow A[1 \dots (mid - 1)]$
- $A_R \leftarrow A[(mid + 1) \dots n]$
- If ($x < A[mid]$)
 - Search x in A_L return(`BinarySearch-v2( $x, A_L, mid - 1$ )`)
- else
 - Search x in A_R return(`BinarySearch-v2( $x, A_R, n - mid$ )`)

Introduction

Digression: Binary Search

Problem

Given a sorted array A containing n integers and an integer x , check if x is present in A .

Algorithm

BinarySearch-v2(x, A, n)

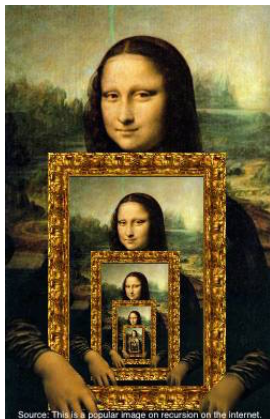
- If ($n \leq 0$), then return(“not present”)
- $mid \leftarrow \lfloor n/2 \rfloor$
- If ($A[mid] = x$), then return(“present”)
- $A_L \leftarrow A[1 \dots (mid - 1)]$
- $A_R \leftarrow A[(mid + 1) \dots n]$
- If ($x < A[mid]$)
 - ~~Search x in A_L~~ return(BinarySearch-v2($x, A_L, mid - 1$))
- else
 - ~~Search x in A_R~~ return(BinarySearch-v2($x, A_R, n - mid$))

- The above function calls marked in red are called *recursive* function calls.
- The function BinarySearch-v2 is called a *recursive function*.

Introduction

Digression: Binary Search → Recursive Functions

- Recursion: Self reference.



- In our context, we talk about recursive functions.

Introduction

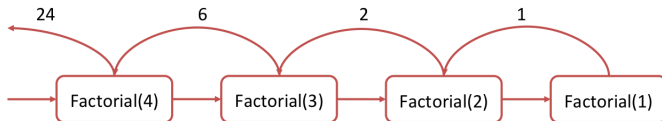
Digression: Binary Search → Recursive Functions

- Recursive function: A function that makes a call to itself.

Algorithm

`Factorial( $n$ )`

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)



Introduction

Digression: Binary Search → Recursive Functions

- Recursive function: A function that makes a call to itself.

Algorithm

Factorial(n)

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)

- Base case: Returns result for small value of inputs. Defines the recursion termination condition.
- Reduction step: Assuming that the function returns correct value for smaller inputs use function calls on smaller inputs to compute the result on the given input.

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions?

Algorithm

Factorial(n)

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions? **Induction**

Algorithm

Factorial(n)

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)

End