Week 2 Lab – Arrays & Complexity Intuition

FM121: Programming & Data Structures
Spring Semester 2026

Objective

To understand arrays and searching techniques, and to analyze their time complexity.

Concepts Covered

- One-dimensional arrays
- Linear and binary search
- Time complexity basics

Key Notes

- An array is a linear data structure that stores elements of the same data type.
- Array elements are stored in contiguous memory locations.
- Arrays are 0-indexed, meaning the first element is accessed using index 0.
- The array name represents the base address of the first element.
- Accessing an array element using its index takes constant time, i.e., O(1).
- Linear search checks elements sequentially and has time complexity O(n).
- Binary search works only on sorted arrays and has time complexity O(log n).
- Finding minimum, maximum, or peak elements requires careful comparison of neighboring values.
- Understanding memory layout helps in deriving address calculation formulas for array elements.

Problem 1: Linear Search for Positive Integer Values

Problem Statement

Write a program to implement the Linear Search algorithm to search for a given positive integer in an unsorted list of positive integers.

Logic: The program should scan the array sequentially and report whether the element is found. If found, display the index position of the element; otherwise, display an appropriate message indicating that the element is not present.

Objectives

- To understand the working of the Linear Search algorithm
- To analyse sequential searching in an unsorted array
- To practice array traversal and analyze the number of comparisons in best and worst case

Input Specifications

- An integer n representing the number of elements ($n > 0$)
- An array of n positive integers
- A positive integer key to be searched

Constraints

- All array elements must be positive integers
- $1 \leq n \leq 1000$

Algorithm (Brief)

1. Read the value of n
2. Read n positive integers into an array
3. Read the search key
4. Compare the key with each element sequentially
5. If a match is found, display the index and terminate
6. If the loop ends without a match, display *Element not found*

Sample Test Cases

Test Case 1

- Input: $n = 5$, Array = 12 45 7 23 9, Key = 23
- Output: Element found at index 3

Test Case 2

- Input: $n = 6$, Array = 10 20 30 40 50 60, Key = 25
- Output: Element not found

Test Case 3

- Input: $n = 1$, Array = 15, Key = 15
- Output: Element found at index 0

Problem 2: Find Minimum and Maximum Elements in a Positive Integer Array

Problem Statement

Write a program to find the minimum and maximum elements present in a given array of integers. The program should scan the array elements and determine the smallest and largest values without modifying the original array.

Objectives

- To understand array traversal techniques
- To identify minimum and maximum values in a dataset
- To practice conditional comparison and iteration
- To analyze the efficiency of single-pass array processing

Input Specifications

- An integer n representing the number of elements in the array($n > 0$)
- An array of n integers

Output Specifications

- Display the minimum element in the array
- Display the maximum element in the array

Constraint

- All array elements must be integers
- $1 \leq n \leq 1000$

Algorithm (Single Pass)

1. Read the value of n
2. Read n integers into an array
3. Initialize:
   - min = array[0]
   - max = array[0]
4. Traverse the array from index 1 to $n - 1$
   - If array[i] < min, update min
   - If array[i] > max, update max
5. After traversal, display the minimum and maximum values

Sample Input

n = 5
Array = [2, 45, 7, 23, 9]

Sample Output

Minimum element = 2 Maximum element = 45

Problem 3: Binary Search for Sorted Positive Integer Values

Problem Statement

Write a program to implement the Binary Search algorithm to search for a given positive integer in a sorted array of positive integers.

Logic: The program should repeatedly divide the search interval in half until the element is found or the search interval becomes empty.

Objectives

- To understand divide-and-conquer searching techniques
- To implement Binary Search on sorted data
- To analyze the efficiency of Binary Search compared to Linear Search

Input Specifications

- An integer n representing the number of elements($n > 0$)
- A sorted array of n positive integers (ascending order)
- A positive integer key to be searched

Output Specifications

- If the element is found, display the index position
- If the element is not found, display "Element not found"

Constraint

- The array must be sorted in ascending order
- All elements must be positive integers
- $1 \leq n \leq 1000$

Algorithm (Brief)

1. Read the value of n
2. Read n sorted positive integers into an array
3. Read the search key
4. Set $low = 0$, $high = n - 1$
5. Repeat until $low \leq high$
   - Compute mid = (low + high) / 2
   - If array[mid] == key, display index and terminate
   - If key < array[mid], set high = $mid - 1$
   - Else set low = mid + 1
6. If the element is not found, display "Element not found"

Sample Test Cases

Test Case 1

- Input: $n = 7$, Array = [5, 10, 15, 20, 25, 30, 35], Key = 20
- Output: Element found at index 3

Test Case 2

- Input: $n = 5$, Array = [2, 4, 6, 8, 10], Key = 7

- Output: Element not found

Test Case 3

- Input: $n = 1$, Array = 100, Key = 100
- Output: Element found at index 0

Problem 4: Find Peak Element in an Integer Array

Problem Statement

Write a program to find a peak element in a given 0-indexed integer array. A peak element is defined as an element that is strictly greater than its immediate neighbors. The program should return the index of the peak element.

Objectives

- To understand the concept of peak elements in arrays
- To apply search techniques for identifying local maxima
- To analyze time complexity improvements over brute-force methods as well as using binary search.

Input Specifications

- An integer n representing the number of elements in the array ($n \geq 1$).
- A distinct positive integer array `nums` which is first sorted in ascending order and then in descending order.
- **Example:**

10	20	30	40	50	35	15

- The peak element is 50.

Output Specifications

- An integer representing the index of a peak element

Constraint

- $1 \leq n \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- The array is **0-indexed**.

Algorithm (Binary Search Based – Recommended)

1. Initialize low $= 0$ and high $= n - 1$.
2. While low $<$ high:
   - Compute mid $= \frac{\text{low}+\text{high}}{2}$.
   - If nums[mid] $<$ nums[mid $+ 1$], then a peak lies on the right side.
     - Set low $=$ mid $+ 1$.
   - Else, a peak lies on the left side or at mid.
     - Set high $=$ mid.
3. When low $==$ high, return low as the peak index.

Alternative Algorithm (Linear Scan – Basic Approach)

1. Traverse the array from index 0 to $n - 1$.
2. For each element, check:
   - It is greater than its left neighbor (if it exists).
   - It is greater than its right neighbor (if it exists).
3. Return the index of the first element satisfying the peak condition.

Sample Test Cases

Test Case 1

- Input: $n = 4$, Array = nums = [10, 20, 30, 5]
- Output: 2 (index of element 30)
- Explanation: Element 30 at index 2 is greater than both neighbours, 20 and 10.

Test Case 2

- Input: $n = 6$, Array = [15, 25, 41, 63, 5, 3]
- Output: 3
- Explanation: Element 63 (index 3) is greater than both neighbours, 41 and 5.

Test Case 3

- Input: $n = 1$, Array = [10]
- Output: 0
- Explanation: A single element is always considered a peak.

Reference

LeetCode 162: Find Peak Element