Phase 1: Data Structures & Algos:

3rd DSA, 4th DSA] C Language — C++ Pointers

int a = 10; → value

key word → identifier &

→ 1000, 1000,

int * ptr = &a → 1000

Arrays: → Same Data Type — C, C++, Java

Collection of homogeneous data — position

example: → int arr[] = {2, 6, 9, 1, 0};

position: 1 2 3 4 5

index: 0 1 2 3 4

Relation: index = pos − 1 ; pos = index + 1

Python: → List → Heterogeneous (different data)

[1, 1.2, "SJCIT", True]

Collections: Java

STL — C++

Standard Template Library

Stack, queue, map, list, vector

Introduction to DSA: → [Logic] → DAA

* Why do we need DS & A? [Time] [Data]

Time complexity

Searching 4–6 LPA

* Linear Search
* Binary Search
* Recursive Binary Search

10 LPA

* Jump Search
* Interpolation Search

Sorting

* Bubble Sort
* Selection Sort
* Insertion Sort
* Merge Sort
* Quick Sort
* Heap Sort
* Count Sort
* Radix Sort
* Wave Sort
* Shell Sort

Big O Notation.

Linear Search Alg: → (6) target or key given

int arr[] = {2, 4, 8, 1, 7, 6} (5)

If the key is found, return the position or the index.

If not then, return −1.

BTC → O(1) → constant (invalid) index = 5, pos = 5+1 — O(1000)

Time Complexity → O(n)

O(n) is very slow for larger size arrays, so we use Binary Search. Prerequisite → Sorted Array — BS + Recursion

key = 3 | key = 12

Sorted Array: | 2 | 3 | 5 | 8 | 9 | 12 | 15 |

index: 0 1 2 3 4 5 6 — mid = (s+e)/2

s=0, e=6, m=3

① if (arr[mid] == key) { return mid; } — 8 == 12 = (0+6)/2, 12 == 12, 8 == 3 = 3

else if (arr[mid] < key) 8 < 12 — go to right → s = m+1; — m = (4+6)/2 = 5

s=0, e=2, else { e = m−1; (go to left) } — (0+2)/2 = 1 — 5 = mid

Time Complexity of Binary Search

Initial Size N

N → N/2 → N/2^2 → N/2^3 → ... → N/2^k

TC ↓

k = log_2 N ⇒ O(log N)

where k = constant 0, 1, 2, 3 . . . .