Competitive Programming

Arrays

B.Bhuvaneswaran, AP (SG) / CSE

9791519152

bhuvaneswaran@rajalakshmi.edu.in

RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution

Affiliated to ANNA UNIVERSITY, Chennai

Introduction

- An array is a collection of elements (values or variables), each identified by atleast one index, stored in contiguous memory locations.

Empty Array

0	1	2	3	4	5	6	7	8	9

Strengths

- Fast lookups
  - Retrieving the element at a given index takes *O(1)* time, regardless of the length of the array.
- Fast appends
  - Adding a new element at the end of the array takes *O(1)* time.

Weaknesses

- Fixed size
  - You need to specify how many elements you're going to store in your array ahead of time. (Unless you're using a fancy dynamic array.)
- Costly inserts and deletes
  - You have to "scoot over" the other elements to fill in or close gaps, which takes worst-case *O(n)* time.

Operations

- Insertion
- Deletion
- Update
- Sorting
- Searching

Examples

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90	

Inserting

- If we want to insert something into an array, first we have to make space by "scooting over" everything *starting at* the index we're inserting into:

0	1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90	
		↑ 25							

0	1	2	3	4	5	6	7	8	9
10	20	25	30	40	50	60	70	80	90

- In the worst case we're inserting into the 0^{th} index in the array (prepending), so we have to "scoot over" *everything* in the array. That's *O(n)* time.

Deleting

- Array elements are stored adjacent to each other. So when we remove an element, we have to fill in the gap - "scooting over" all the elements that were *after* it:

0	1	2	3	4	5	6	7	8	9
10	20	25	30	40	50	60	70	80	90
		↓							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	20	30	40	50	60	70	80	90	

- In the worst case we're deleting the 0^{th} item in the array, so we have to "scoot over" *everything else* in the array. That's O(n) time.

Why not just leave the gap?

- Because the quick lookup power of arrays depends on everything being sequential and uninterrupted.
- This lets us predict exactly how far from the start of the array the 138^{th} or $9{,}203^{rd}$ item is.
- If there are gaps, we can no longer predict exactly where each array item will be.

Updation

- Update the value 60 to 55 at index 5 - Time Complexity?

0	1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90	

Sorting

Sorting Algorithm	Time Complexity			Space Complexity	Stable
	Best	Average	Worst	Worst	
Quicksort	O(n log n)	O(n log n)	O(n^2)	O(log n)	No
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes
Heapsort	O(n log n)	O(n log n)	O(n log n)	O(1)	No
Timsort	O(n)	O(n log n)	O(n log n)	O(n)	Yes
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	No *
Shellsort	O(n log n)	-- (depends on gap sequence)	O(n^2)	O(1)	No
Bucket Sort	O(n + k)	O(n + k)	O(n^2)	O(nk)	Yes
Radix Sort	O(nk)	O(nk)	O(nk)	O(nk)	Yes
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(k)	Yes

*Selection sort can be implemented as a stable sort if, rather than swapping the minimum value with its current value, the minimum value is inserted into the first position and the intervening values shifted up. However, this modification either requires a data structure that supports efficient insertions or deletions, such as a linked list, or it leads to O(n^2) writes.

Searching

- Sorted array
  - O(log n)
- Unsorted array
  - O(n)
- Note: Sort + Search → O(n log n)

Finding Index / Finding Element

- Find index of the element 60 in the array - Time Complexity?
- Find element 60 is in the array or not - Time Complexity?
- Find element 60 and update to 55 in the array - Time Complexity?

0	1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90	

Data structures built on arrays

- Arrays are the building blocks for lots of other, more complex data structures.
- Don't want to specify the size of your array ahead of time? One option: use a dynamic array.
- Want to look up items by something other than an index? Use a dictionary.

Worst case

- Space
  - O(n)
- Lookup
  - O(1)
- Append
  - O(1)
- Insert
  - O(n)
- Delete
  - O(n)

Queries?

Thank You…!