Data Structure

Array → Algorithms

↓ Data structure

↓ Divide & conquer

↳ Recursion

1) Linear

2) Non-Linear

collections of items in a contiguous manner.

Student marks (100) array → index

1000 → Base address 1012

arr[8] ↳ 47

85	75	70	65	98	99	50	33	47	77
0	1	2	3	4	5	6	7	8	9
↓ index	1004	1008		1016	-	-	-	-	-

n = 10

int → 4 Bytes

Last index = 9

Address of 9th index (i)

$$1000 + (9-0) * 4$$

$$1000 + 36 = 1036$$

→ Base address + (i - lower Bound) * Size of each element

Hexadecimal → (0-9, A-F) (16 Digits)

array[7] ↳ 33 ⟹ Random access

↓

O(1) → constant time complexity

Binary Search ⇓ Random access

Why we need different Data Structures??

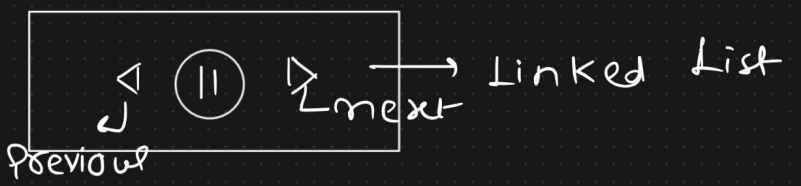
- Array → Searching operation
- Linked List → insertion & deletion
- Hashing (Dictionary) → Mapping of key-value
- Graph → Networking (BFS & DFS) → Traversal Algorithm
- Tree → Non-Linear (Networking)
- Stack → LIFO (Last In First Out) ⇒ function calling (Recursion)
- Queue → FIFO (first in first out)
- Heap → maxima & minima (K-closest)

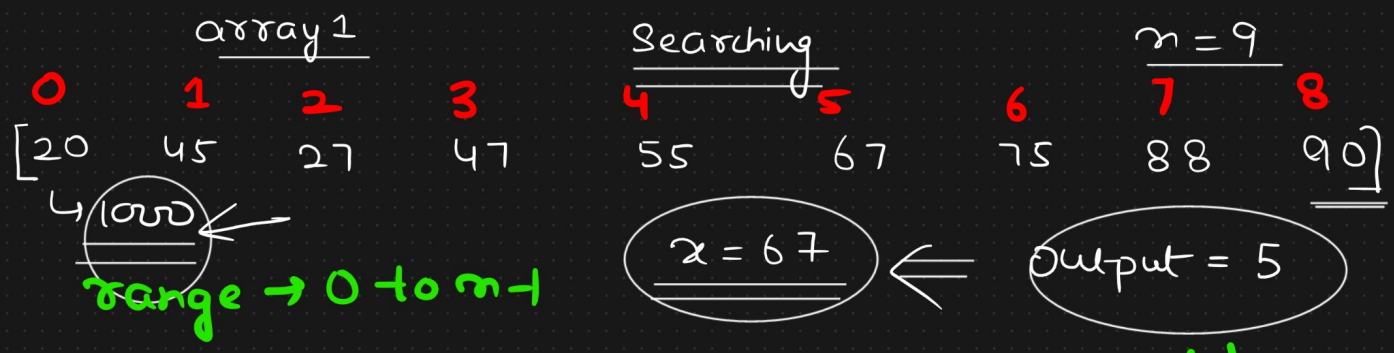
Time complexity

LinkedIn

↳ Post → 7600 followers

5%





Linear Search

```
for i in range(0,n):
    if arr(i) == x:
        return i
return -1
```

match

Pseudocode

$T(n) = O(n)$

↳ Worst case (Element present near to last index)

Best case → $O(1)$

↳ (Element present near to the first index)

Average case → $\frac{1}{2} \cdot \underbrace{O(n)}_{\text{Worst case}} + \frac{1}{2} \cdot \underbrace{O(1)}_{\text{Best case}}$

$\Rightarrow O(n)$

Binary Search

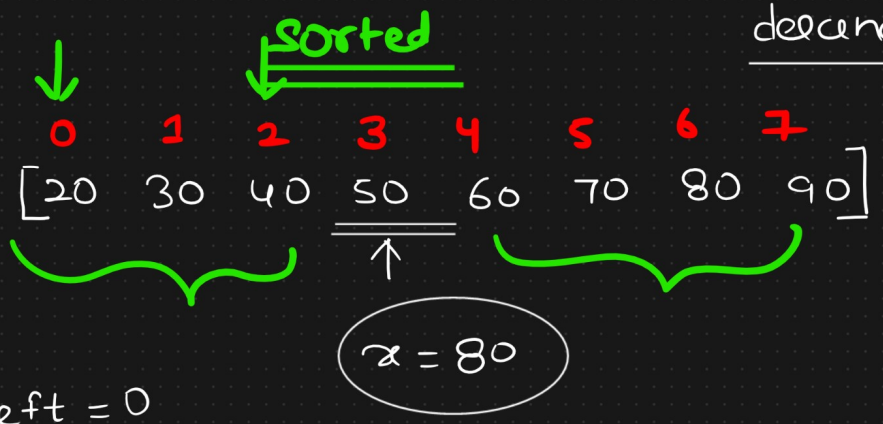
↳ Sorted array

①

⇩

ascending order or

descending order



arr(mid) = 50

x = 80

x = 30

left = 0

right = 7

$mid = \frac{0+7}{2} = 3$ (Lower Bound)

```
binarySearch(arr, left, right):
    while left <= right:
        if arr(mid) == x:
            return mid
        elif arr(mid) < x:        # 50 < 80
            left = mid+1          # binarySearch(arr, mid+1, right)
        else arr(mid) > x:
            right = mid-1         # binarySearch(arr, left, mid-1)
    return -1
```

Recursion

Recursion

↳ calling the same function again inside the method definition

Note

↳ 1) Sorted array *

(Logically)

x = 50

x = 20

2) 20 (0), 30 (1), 40 (2), 50 (3), 70 (4)

Sorted array

{ i = 0, J = 4

mid = (i + J)/2 = 2

binarySearch(arr, 0, 4) 50 == 50

True

while i <= J:

arr(mid) == x :

40 == 50 X

return mid

arr(mid) < x :

40 < 50

right

binarySearch(arr, 3, 4)

arr(mid) > x :

binarySearch(arr, 0, 1)

left

Psuedocode

Binary

Left

Right

mid = (3+4)/2 = 3

i → starting index, J → ending index

Preferrable → mid = i + (J - i)/2

= 3 + 1/2

i = 3

J = 4

⇒ 3

mid = (i + J)/2

mid = (3 + 4)/2

= 7/2 = 3

binarySearch(arr, 0, 4)

↓

binarySearch(arr, 3, 4)

mid = 3

result → 3

i = 1,00,000

J = 4,00,000 → 4,00,000

(i+J)/2

mid = 5,00,000 / 2

mid = 1,00,000 + 3,00,000/2

if i == j: ⟹ only one element

if arr(i) == x:

return i

else:

return -1 ⟹ Terminating Condition

else:

Recursive code

Recurrence Relation

$T(n)$

↳ binarySearch(arr, i, j, x):

while i <= j:

mid = i + (j-i)//2 → c

if arr(mid) == x: → c

return mid

elif arr(mid) < x:

⇒ binarySearch(arr, mid+1, j, x) → $T(n/2)$

else: OR

⇒ return -1 ← binarySearch(arr, i, mid-1, x) → $T(n/2)$

Binary Search Algorithm

$$T(n) = T\left(\frac{n}{2}\right) + c$$

Substitution

Master's

Master's Theorem

~~$T(n) = 2T\left(\frac{n}{2}\right) + c$~~

$a = 1$ $k = 0$

$b = 2$ $p = 0$

$$\log_b^a = \log_2^1 = 0$$

case 2 $\Rightarrow$ $\log_b^a = k$

$p > -1 \Rightarrow \Theta(n^k \log^{p+1} n)$

$\Theta(\log n)$

Substitution

$$T(n) = T(n/2) + c$$

$$= T(n/2^2) + c + c = T(n/2^2) + 2c$$

$$= T(n/2^3) + 3c$$

$\frac{n}{2^k} = 1$ $\Big\downarrow$ k times

$k = \log_2 n$ $\Rightarrow T(n/2^k) + k \cdot c$

$$= T\left(\frac{n}{2^{\log_2 n}}\right) + c \cdot \log_2 n$$

$$= T\left(\frac{n}{n^{\log_2 1}}\right) + c \cdot \log_2 n$$

$\Rightarrow O(\log_2 n)$