Merge Sort

Saturday, 1 May 2021 6:23 PM

arr → { 90 70 20 60 50 30 40 10 }

Expectation → mergeSort (arr, st, end) → Return me a sorted array.

faith → mergeSort (arr, st, ?) → arr1

mergeSort (arr, ?, end) → arr2

Merging of faith and Expectation → mergeSort (arr, st, end) -

mid = $\frac{lo + (hi - lo)}{2}$

Figure out (1) Base case

(2) Flow of code

Sorted {

arr1 = mergeSort (arr, st, mid);

arr2 = mergeSort (arr, mid+1, end);

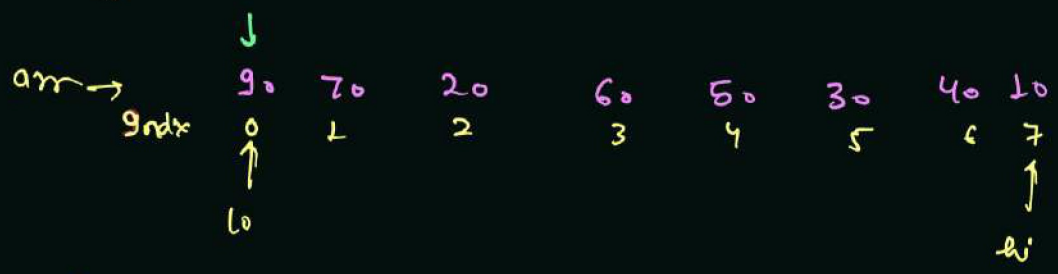
res = MergeTwoSortedArray (arr1, arr2);

return res;

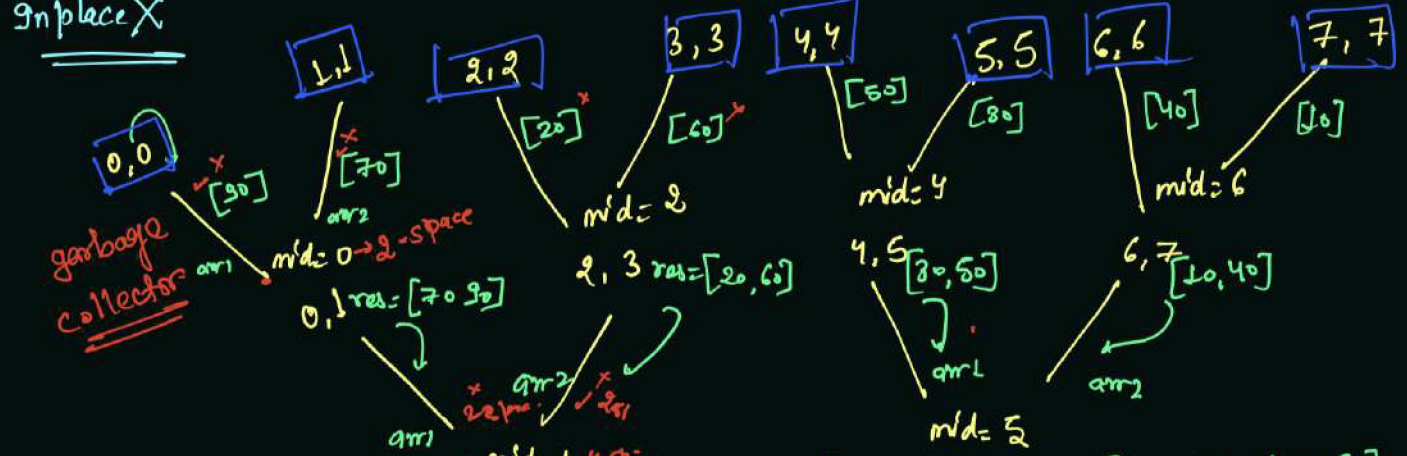
}

Base case for MergeSort →

MergeSort. t.c → O(nlogn)
Space Complex → O(n)



Inplace X



How of Code ↓

Space max through out the code

Inplace – Change in same array

Stack → Logn

Binary Search VS. Linear Search

Saturday, 1 May 2021 7:41 PM

Data is Random

size of data = 10^9 order

no. of query = 10^9 order

$\log n = \log 10^9 = 32$

$[\log_a a^b = b]$

cost = Complexity.

1 query = 10^9 iteration

cost

overall cost for Binary Search

Sort the data

→ cost = $n \log n$

$= 10^9 \log_2(10^9)$, $10^9 \equiv 2^{32}$

C_1 cost = $10^9 \log_2(2^{32}) = 32 \times 10^9$

32

→ cost for single data = $\log n$

$= \log_2 10^9 = \log_2 2^{32}$

$= 32$

→ cost for q seaerchy = $32 \times q$

cost C_2 = 32×10^9

overall cost = $C_1 + C_2$

$= 32 \times 10^9 + 32 \times 10^9$

$= 64 \times 10^9 = 10^{10}$ cost

overall cost for linear search

cost of finding of single data

$= 10^9$ 32×10^9

cost of finding dat q times

$= q \times 10^9$

$= 10^9 \times 10^9$

$= 10^{18}$

Cost = 10^{18} ||=

Prime Number with Queries

Saturday, 1 May 2021 8:00 PM

Array with Element → $1 \le arr[i] \le 40$ ($1 \le arr[i] \le 10^6$) (n)

size of array → 10,000 (size = 100) (m)

for Every Element, print if it prime, or not prime

Allowed time complexity $= m \times \sqrt{n}$

Normal Approach →

→ Check every element if it its prime or not

cost for single check → $\sqrt{n}$.

cost for m check → $m\sqrt{n}$

Sieve of eratosthenes →

n = 0 to 40 = $m \times \sqrt{n}$

2×10^9

m — queries: 38 O(1), 37 O(1), 35, 33, 37, 9, 8, 3, 2, 9, 7, 9, 3, 11, 3

2 ✓, 3 ✓, 4 ✗, 5 ✓, 6 ✗, 7

4 already marked

a × b, b × a

0: T	1: T	2: T	3: T (prime)	4: F	5: T	6: F	7: T	8: F	9: F (Not prime)
10: F	11: T	12: F	13: T	14: F	15: F	16: F	17: T	18: F	19: T
20: F	21: F	22: F	23: T	24: F	25: F	26: F	27: F	28: F	29: T
30: F	31: T	32: F	33: F	34: F	35: F	36: F	37: T (Prime #)	38: F	39: F
40: F									

upper Range → 40 → factor = unique

2 to $\le \sqrt{n}$

0, 1, $\sqrt{n}$ — a×b, b×a

mark false

9

3

3

40

F

upper Range → 40 → factor = unique

2 to ≤ √n

[1 … √n] a×b

made false

q >> Range

Generation → Cost: O(n) + [40/2 + 40/3 + 40/5] + 40/7

2 3 5

n = 10^6

Common] 2·3 pg

$\sum_{i=1}^{\sqrt{n}} \frac{n}{prime_i}$

complexity ⇒ [n/2 + n/3 + n/5 + n/7 + n/11 + …]

⇒ O(n log(log(n)))

10 → log(log 10^9) = log(log_2 2^32)

= log(32)

= log(2^5)

5×10^8 = O(n) 5n

O(n log log n) ≡ O(n)

n * 5

n + n + n … nn

n

n

n×n O(n)

1

cost for generation = n log log n ≤ O(m)

m >> n

complexity - O(1)

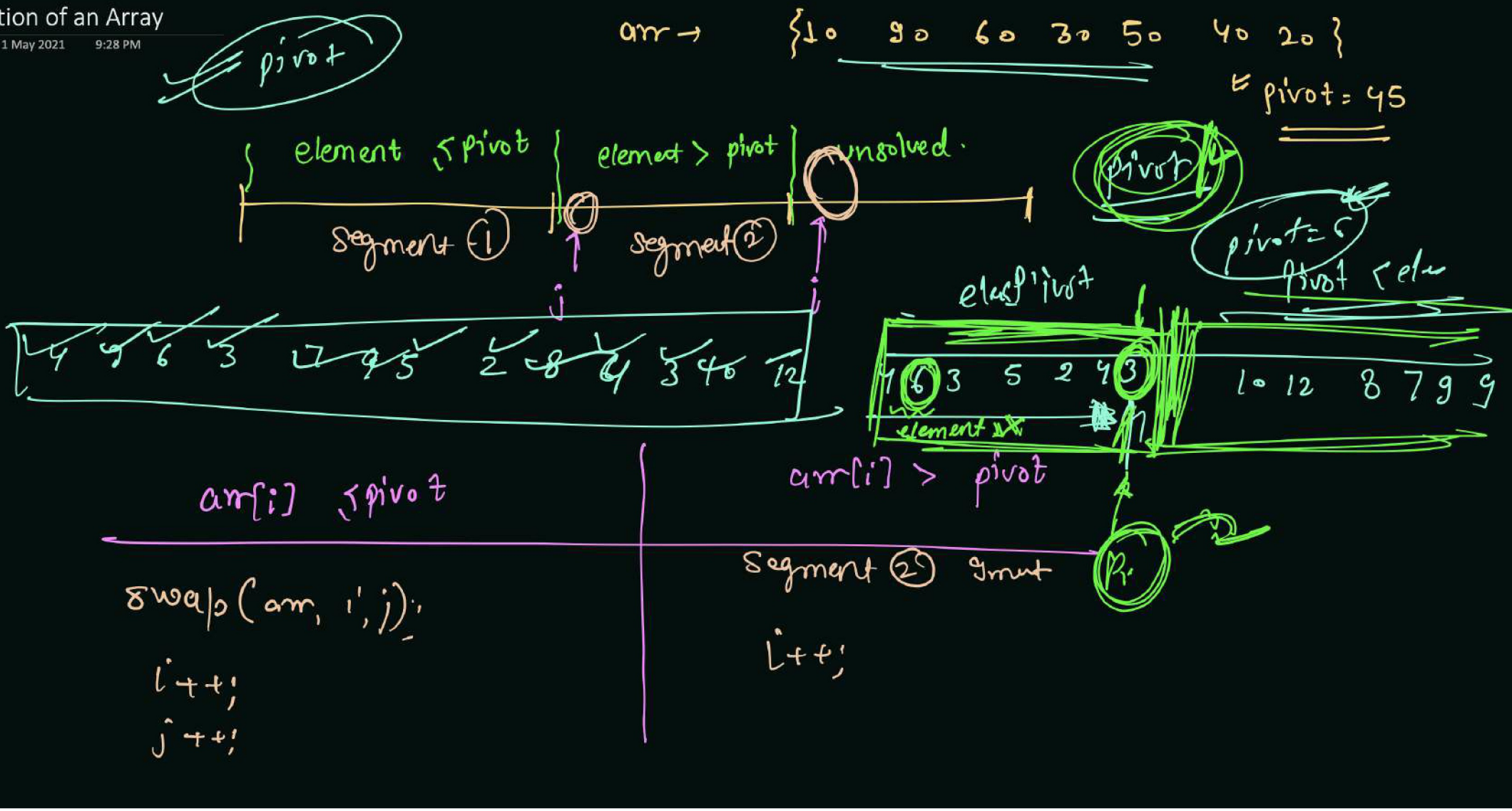
overall cost = O(n) + O(m) ≡ O(m)

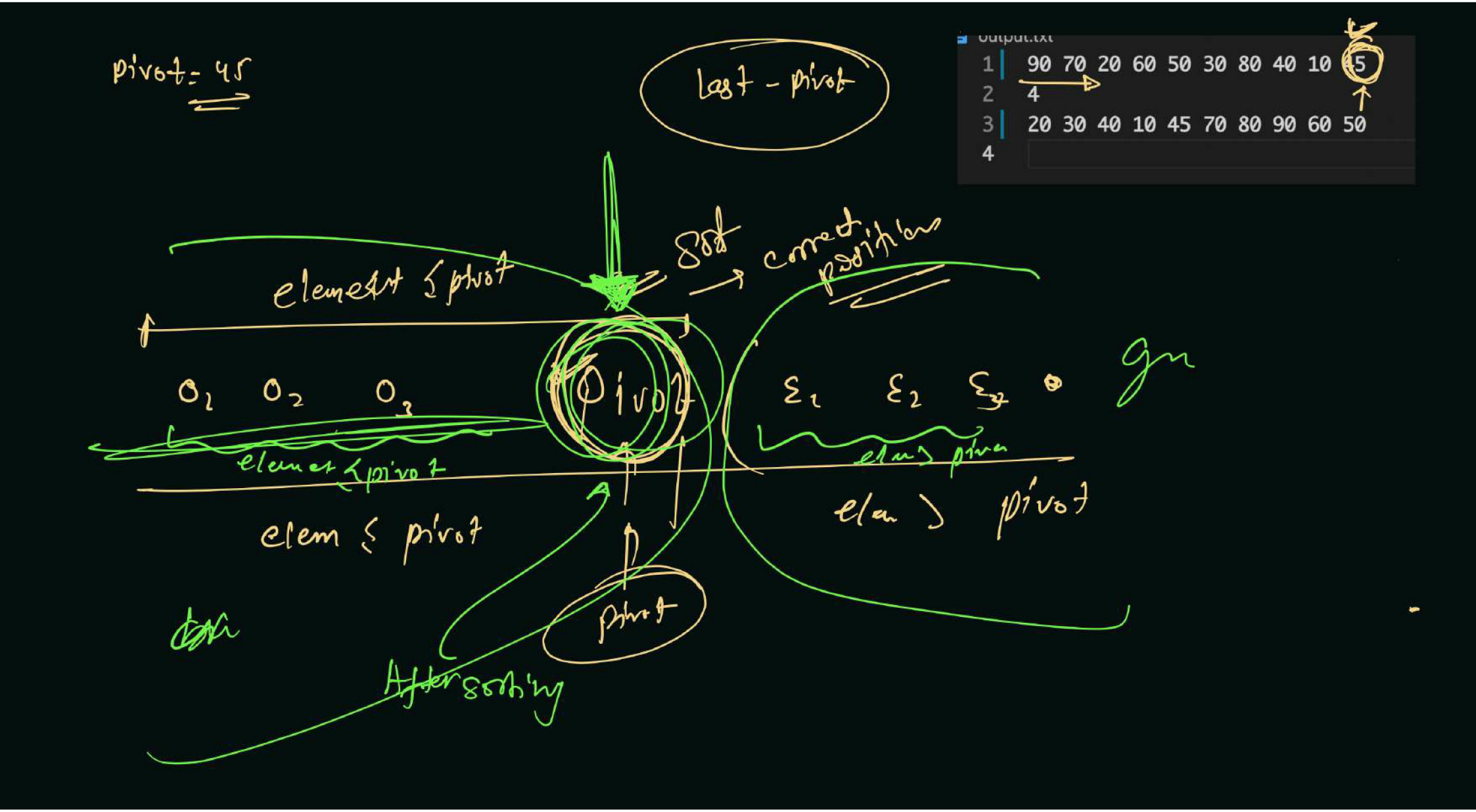
generation + queries Resolve

O(m)

Partition of an Array

Saturday, 1 May 2021 9:28 PM




pivot = 45
last - pivot
output.txt
1 90 70 20 60 50 30 80 40 10 45
2 4
3 20 30 40 10 45 70 80 90 60 50
4
elements < pivot
sort
correct position
O1 O2 O3
Opivot
E1 E2 E3
elem < pivot
elem > pivot
elem < pivot
elem > pivot
pivot
After sorting

Quick Sort

Saturday, 1 May 2021 9:44 PM

arr = { 90 60 40 30 50 20 40 }

solve for left

correct position after sort

pivot

10 30 2 40 90 60 50

part as well

Segregate. solve for

pi → partition index

right part as well

Expectation →

= quicksort(arr, lo, hi);

faith →

quicksort(arr, lo, ?); → pi-1

quicksort(arr, ?, hi) → pi+1

pivot = 40

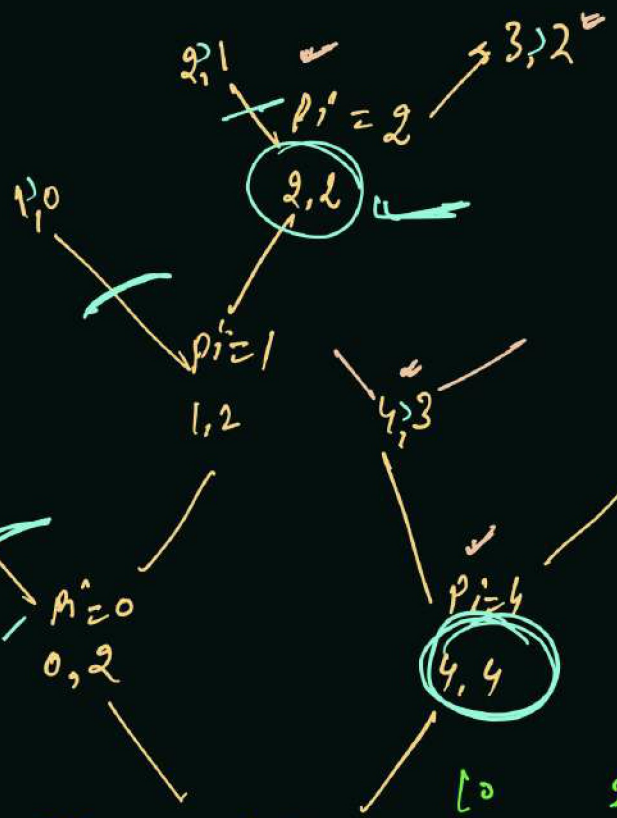
Merging of faith & Expectation →

```
pivot = arr[hi];
pi = partitionIndex(arr, lo, hi, pivot);
quick(arr, lo, pi-1);
quick(arr, pi+1, hi);
```

```java
public static void quickSort(int[] arr, int lo, int hi) {
    int pivot = arr[hi];
    int pi = partitionIndex(arr, lo, hi, pivot);
    quickSort(arr, lo, pi - 1); // left call
    quickSort(arr, pi + 1, hi); // Right call
}
```

Sorted array -

Pivot -



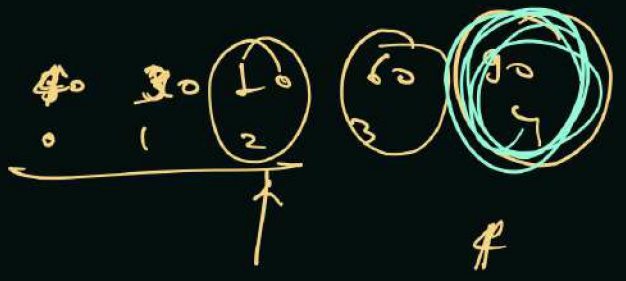
Base case → lo ≥ hi

Return

arr → 90, 70, 30, 10, 60

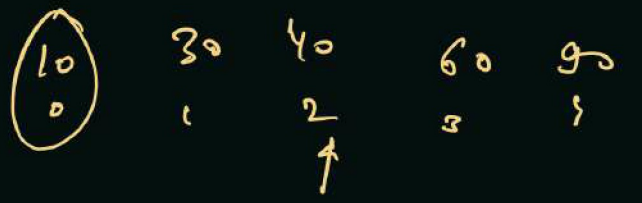
0 1 2 3 4

Batch -

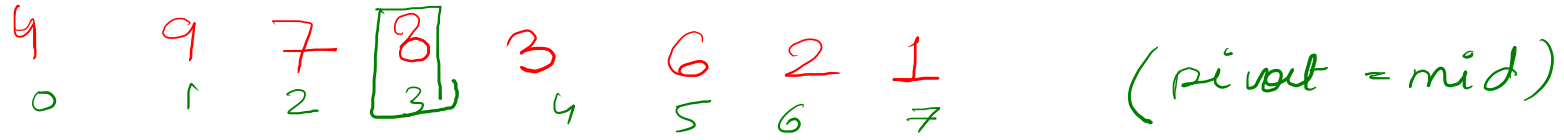


Random data → nlogn

10 20 30 40 50

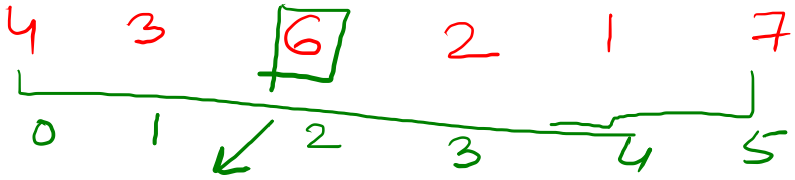


array - Sorted = $O(n^2)$



① $7/2 = 3$ pi ⇒ (4 3 6 2 1 7 8 9)

pindex = 6



$5/2 = 2$ pi = (4 3 2 6 7)

0 1 2 3 4

pi = 3

[4 3 2] [7]

0 1 2

(2 3 4)

[2] [4]

[9] $0/2 = 0$ (lo >= hi) ar[9]

```
quicksort(ar, lo, hi)
{
    if (lo > hi)
        return;
    int pivot = arr[hi];
    int pindx = partitionA(ar, pivot, lo, hi)
    Quicksort(ar, lo, pindx-1)
    Quicksort(ar, pindx+1, hi)
}
```