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);

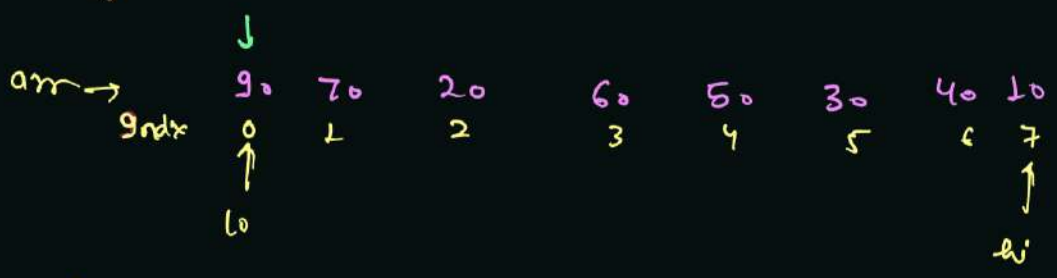
res = MergeTwoSortedArray (arr1, arr2);

return res;

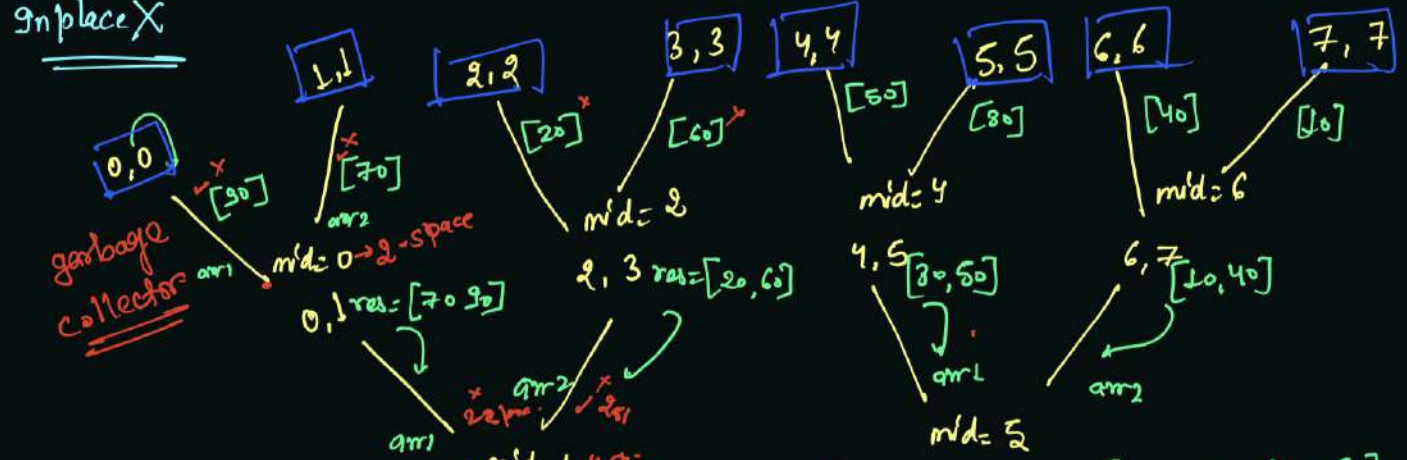
}

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

Prev

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 searchy = $32 \times q$

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

10

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 = 100 (m) size of array → 10,000

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

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

Normal Approach -

→ Check every element if it is prime or not
cost for single check → $\sqrt{n}$.
cost for m check → $m\sqrt{n}$

Sieve of eratosthenes →

n = 0 to 40 — $m \times \sqrt{n}$

10^6 4 already marked

m queries: 2 ✓, 3 ✓, 4 ×, 5 ✓, 6 ×, 7

Queries: 38, 37, 35, 33, 37, 9, 8, 3, 2, 9, 7, 9, 3, 11, 3 (15) — O(1)

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									

a × b, b × a

2×10^9

upper Range → 40 → factor = unique

2 to $\le \sqrt{n}$

0, 1, $\sqrt{n}$ — a×b, b×a

mark false

upper Range → 40 → factor = unique

2 to $\sqrt{n}$

$a \times b$

q >> Range

Generation → Cost: $O(n)$ + $\left[\frac{40}{2} + \frac{40}{3} + \frac{40}{5}\right] + \frac{40}{7}$

made false

$n = 10^6$

common 2, 3 page

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

complexity ⇒ $\left[\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} + \frac{n}{11} + \dots\right]$

⇒ $O(n \log(\log(n)))$

10^9 → $\log(\log 10^9) = \log(\log_2 2^{32})$

$= \log(32)$

$= \log(2^5)$

$5 \times 10^8 = O(n)$ → $5n$

$O(n \log \log n) \equiv O(n)$

$n = 5$

$n + n + n + \dots nn$

$n \times n$ $O(n)$

cost for generation = $n \log \log n \equiv O(m)$

$m >> n$

complexity - $O(1)$

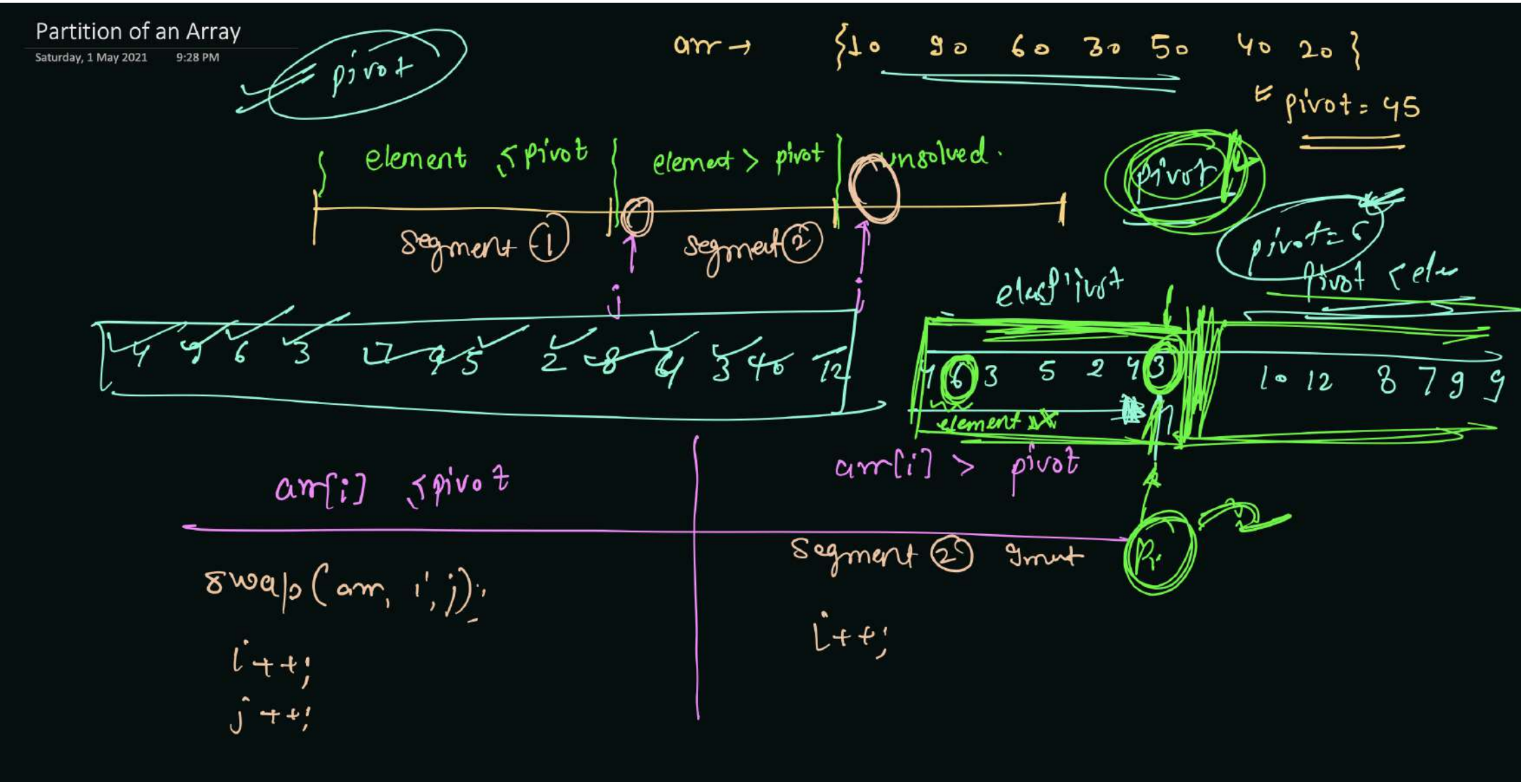
overall cost = $O(n)$ + $O(m)$ $\equiv$ $O(m)$

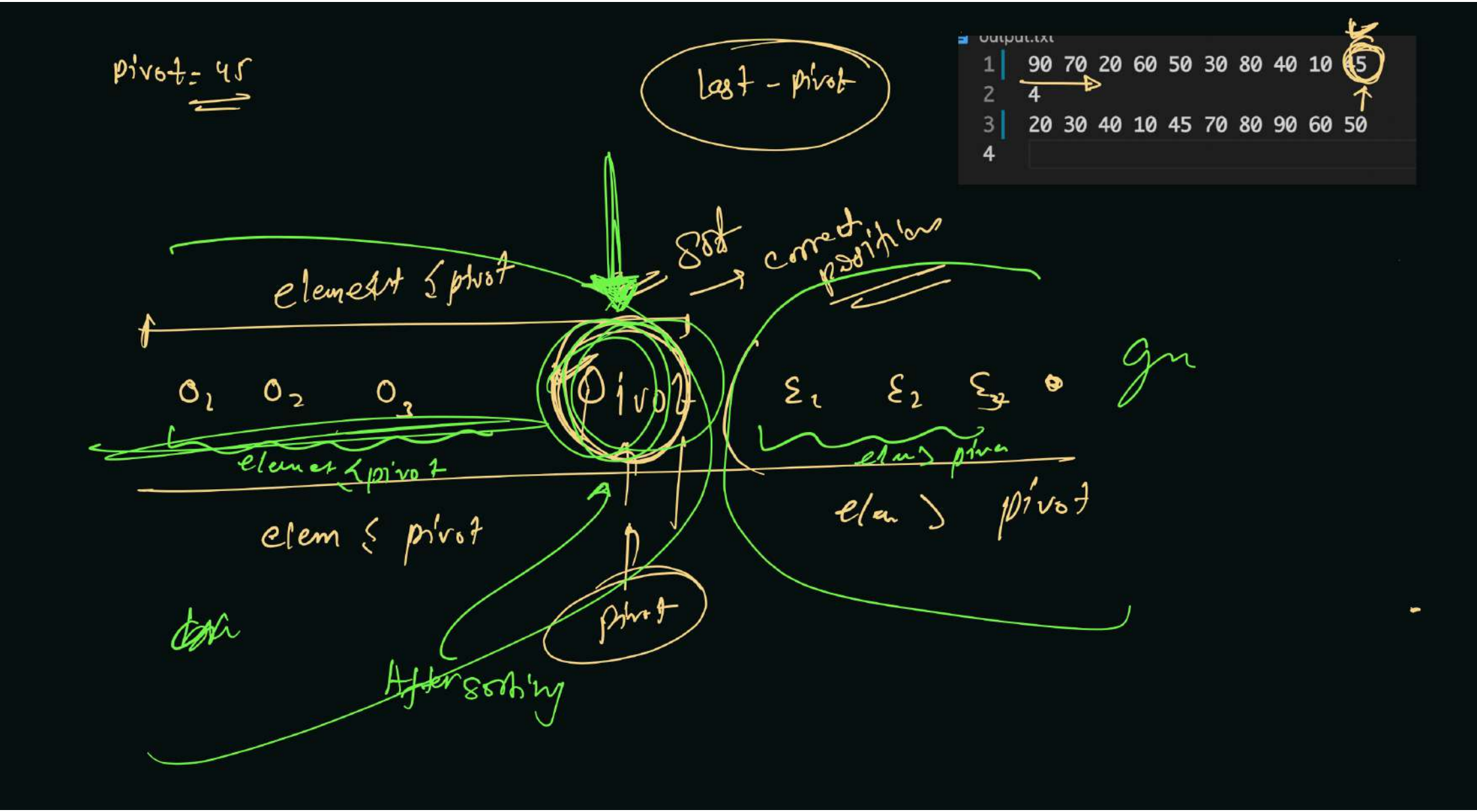
generation + queries Resolve

$O(m)$

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
Pivot
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

Expectation →

= quickSort(arr, lo, hi);

faith →

quickSort(arr, lo, ?); → pi-1

quicksort(arr, ?, hi) → pi+1

pivot = 40

Segregate. solve for right part as well

pi → partition index

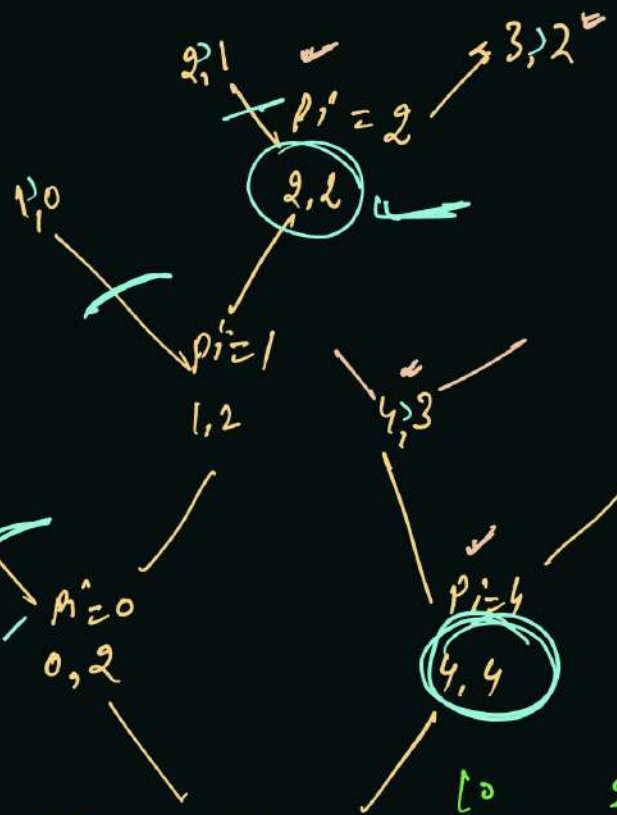
```
pivot = arr[hi];
pi = partition index(arr, lo, hi, pivot);
quick(arr, lo, pi-1);
quick(arr, pi+1, hi);
```

Merging of faith & Expectation →

```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 -



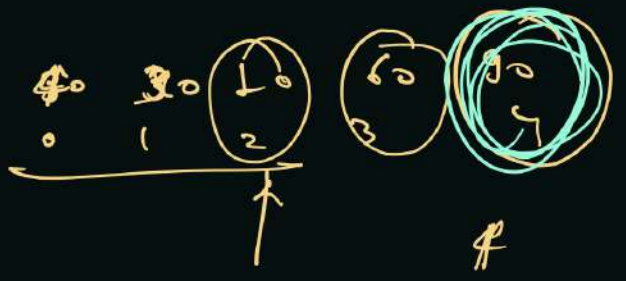
Base case → lo ≥ hi

Return

arr → 90, 70, 30, 10, 60

0 1 2 3 4

Batch -



Random data → nlogn

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