Nikhil Kumar Arora

Date
Page No.

Quick Sort

* Approach → & Example

	0	1	2	3	4	5
a	5	8	1	2	6	7

↑
a[0]

1) Find a suitable place for a[0]. (say that position x)

2)

		5		

↑
x

All the element present before x position should be less than a[x] & the elements present after the x position should be greater than a[x].

0	1	2	3	4	5
1	2	5	8	7	6

3)
→ Partition the bigger array into 2 parts i.e. one ~~belo~~ befor x & one after x.
→ Now sort the two arrays.

⟶ | 1 | 2 | 5 | 6 | 7 | 8 | ⟶ Desired O/p.

→ Quick Sort is inplace sorting Algo as it does not requires extra space.

NIKHIL KUMAR ARORA

Date
Page No.

Bubble Sort (Example & Approach)

→ Bubble Sort is inplace Algorithm. (as it does not require any extra space).

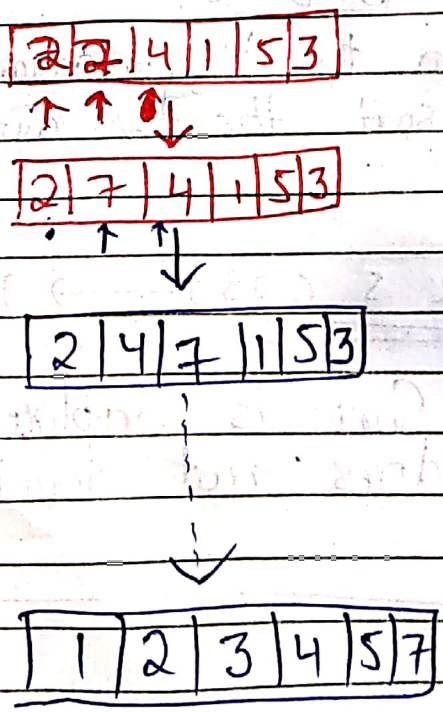
Approach

→ | 2 | 7 | 4 | 1 | 5 | 3 |

Example

(1) We will compare the adjacent elements many times.
(2) if the previous element is greater than adjacent element than we will just swap these elements.
(3) We will do this many times & in the last we will get our sorted array.

{We will exactly compare the elements 'n-1' times}

→ 

NIKHIL KUMAR ARORA

Date
Page No.

* TIME COMPLEXITY OF QUICK SORT →

```
Void quicksort (int input[], int size){
if (size == 0 || size == 1)
        return;
int counter = 0;
for (int i = 1; i < size; i++) → O(n)
  if (input [i] < input [0])
        counter ++;
int temp = input [counter];
input [counter] = input [0];
input [0] = temp;

int i = 0, j = size - 1;
while (i < counter && j > counter) → O(n/2)
  {
   if (input [i] < input [counter])
        i++;
   else {
        int temp 1 = input [i];
        input i = input [j];
        input j = temp 1;
        i++;
        j--;
        }
  }
quicksort (input, counter);
quicksort (input + counter + 1, size -
                counter - 1);
}
```

NIKHIL KUMAR ARORA Date Page No.

QUICK SORT TIME COMPLEXITY

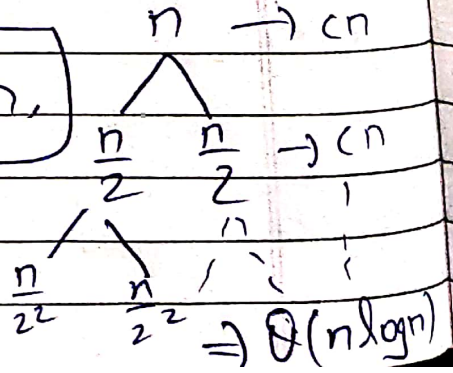
```
QS (array, p, q)          [ALGO.]
  if (p<q)
    r = partition (arr, p, q).
    QS (array, p, r-1)
    QS (arr, r+1, q).

PARTITION (arr, p, q):
    x = arr[q]
    i = p-1
    for (j ← p to q-1)
        if arr[j] < x:
            swap (arr[j], array[++i])
    array[++i] = x
    return i.
```

* Since the partition func^n is traversing through the array once, hence the time complexity of partition = $\theta(n)$.
  $P(n) = \theta(n)$

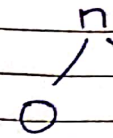
* Best Case Time Comp. of Quick Sort -
  → When we input already sorted array it is the best case.

$$T(n) = n \log n$$



NIKHIL KUM

* WORST CASE

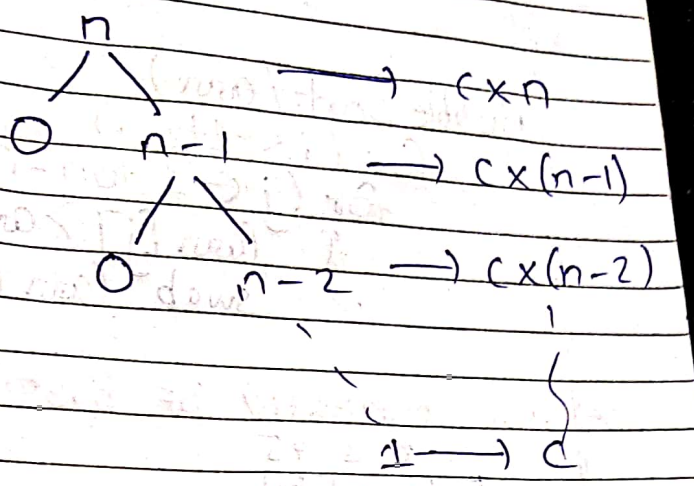


* AVERAGE

NIKHIL KUMAR ARORA

Date
Page No.

* WORST CASE O.S →



$$c\left[\frac{n(n-1)}{2}\right]$$

$$T(n) \approx O(n^2)$$

* AVERAGE CASE ⇒ $T(n) = O(n \log n)$.

NIKHIL KUMAR ARORA

Date
Page No.

BUBBLE SORT

* ALGO →

```
Bubble Sort (arr)
    for (i ← 1 to n)
        for (j ← 1 to n-i)
            if (arr[j] > arr[j+1])
                swap (arr[j], arr[j+1])
```

* TIME COMPLEXITY OF BUBBLE SORT →

1	2	3	4	5
6	5	2	1	3

i	j	Comparision	Swaps
1	1,2,3,4,5	1,2,3,4,5 (5)	(4)
2	1,2,3,4	(4)	(3)
3	1,2,3	(3)	(2)
4	1,2	(2)	(1)
5	1	(1)	(0)

$\Rightarrow \boxed{T(n) = O(n^2)}$

Best Case: In Best Case, (comparision no. will be same so $T(n) = O(n^2)$.

NIKHIL KUMAR ARORA

Date
Page No.

* MERGE SORT Vs QUICK SORT Vs INSERTION SORT Vs BUBBLE SORT →

	MERGE SORT	QUICK SORT	INSERTION SORT	BUBBLE SORT
TIME COMP.	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
CATEGORY	OUT OF PLACE	INPLACE	INPLACE	INPLACE
(EXTRA) (SPACE)	$O(n)$			
BEST CASE			$O(n)$	