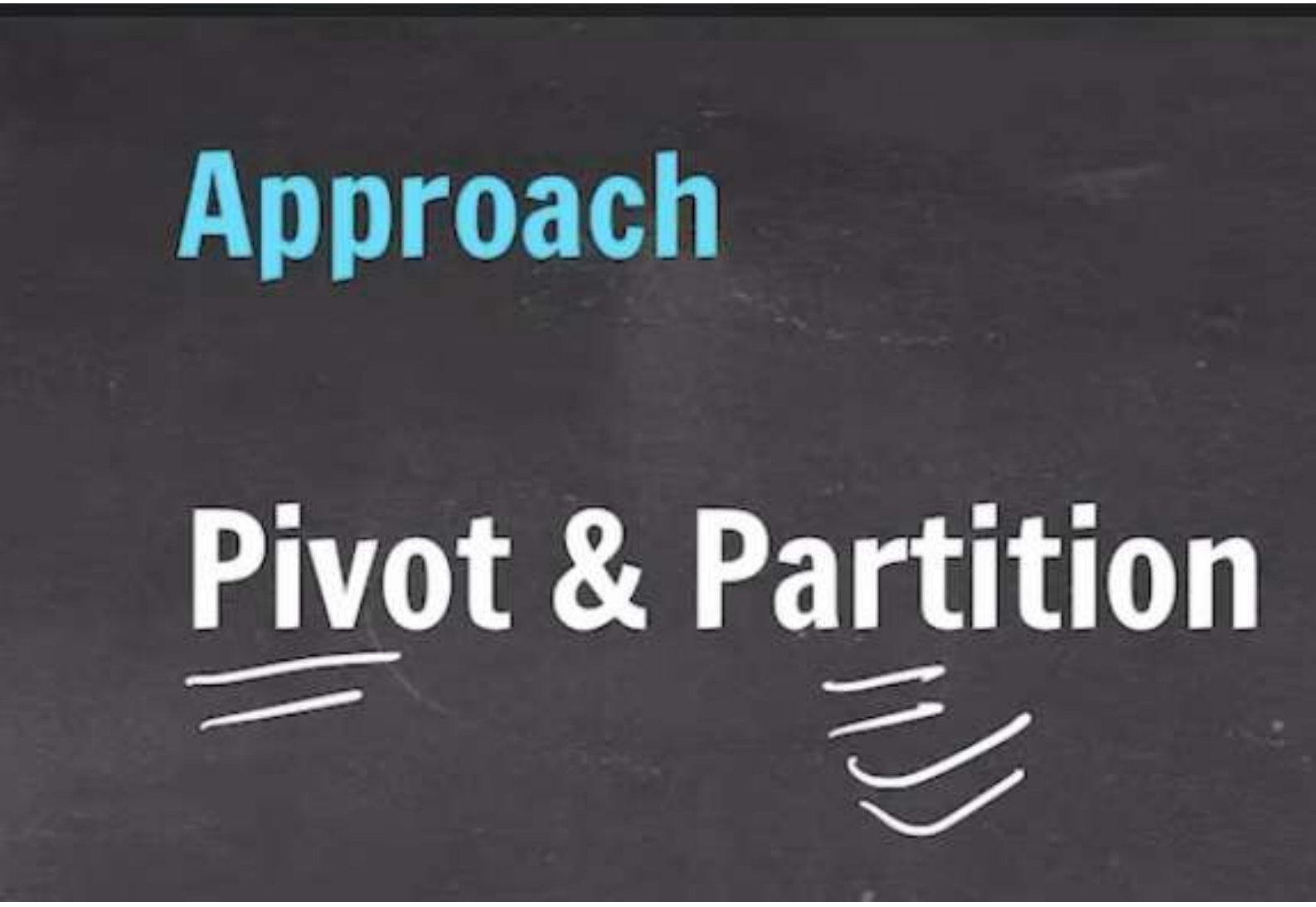
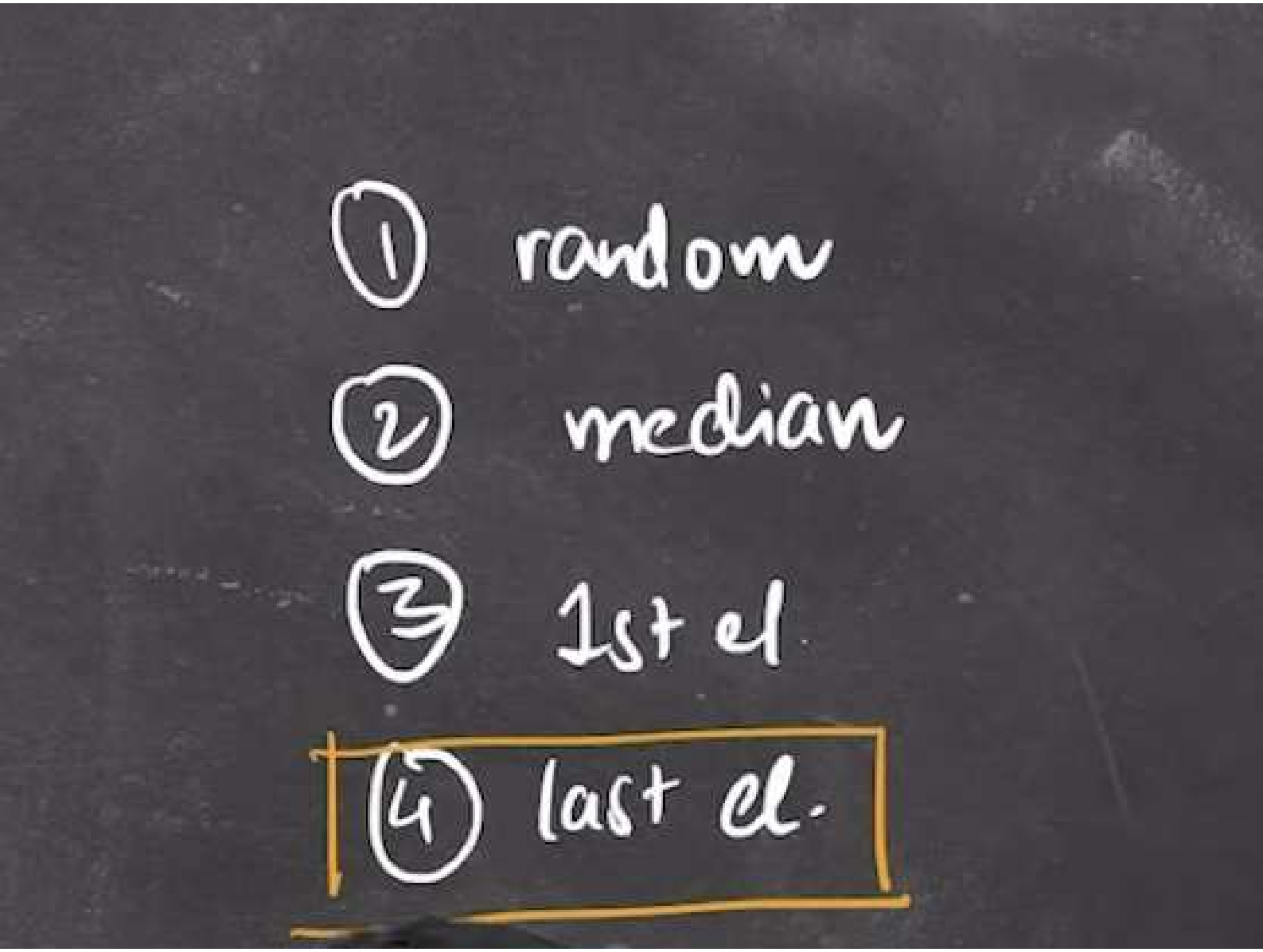
Quick Sort

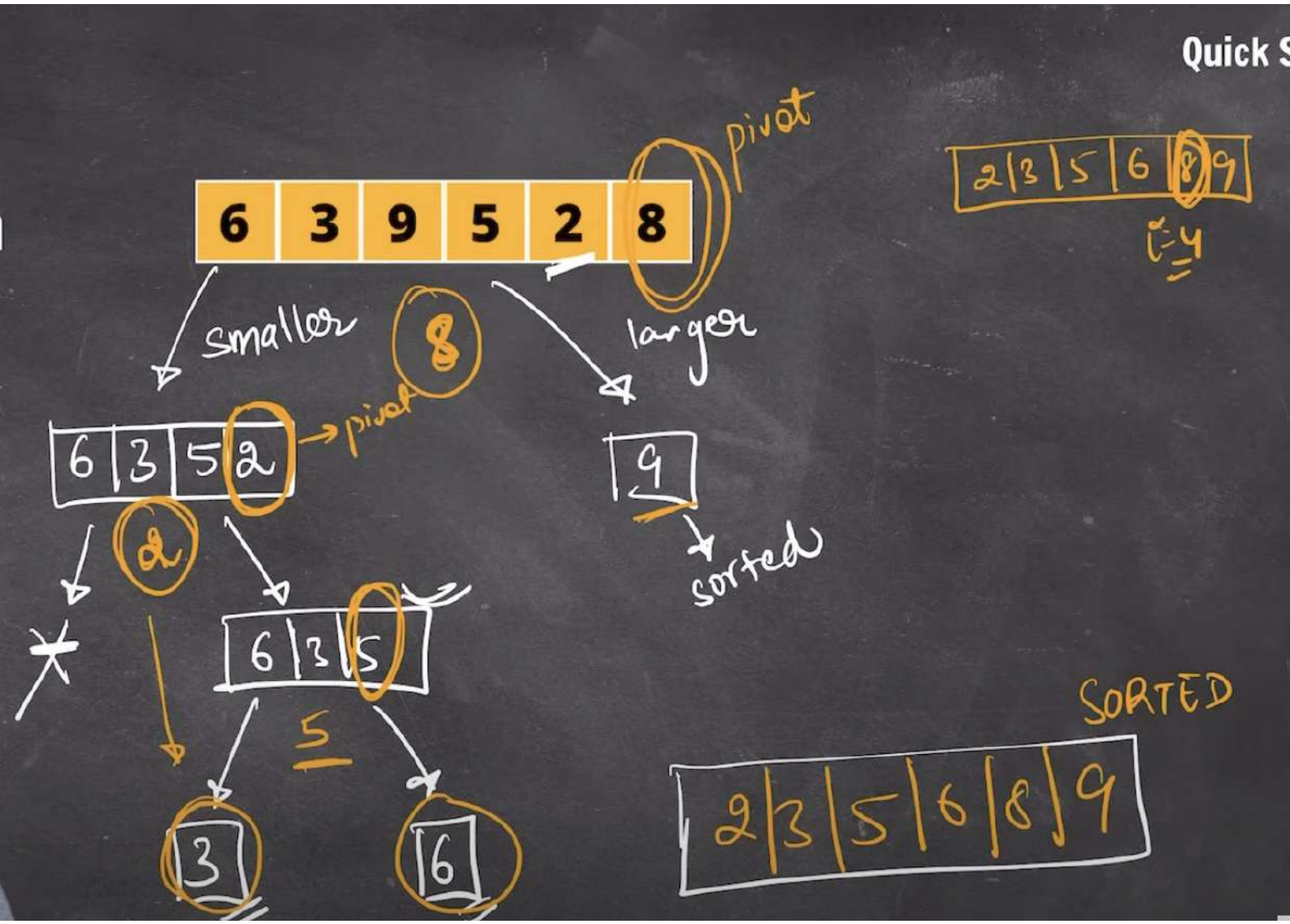
unsorted

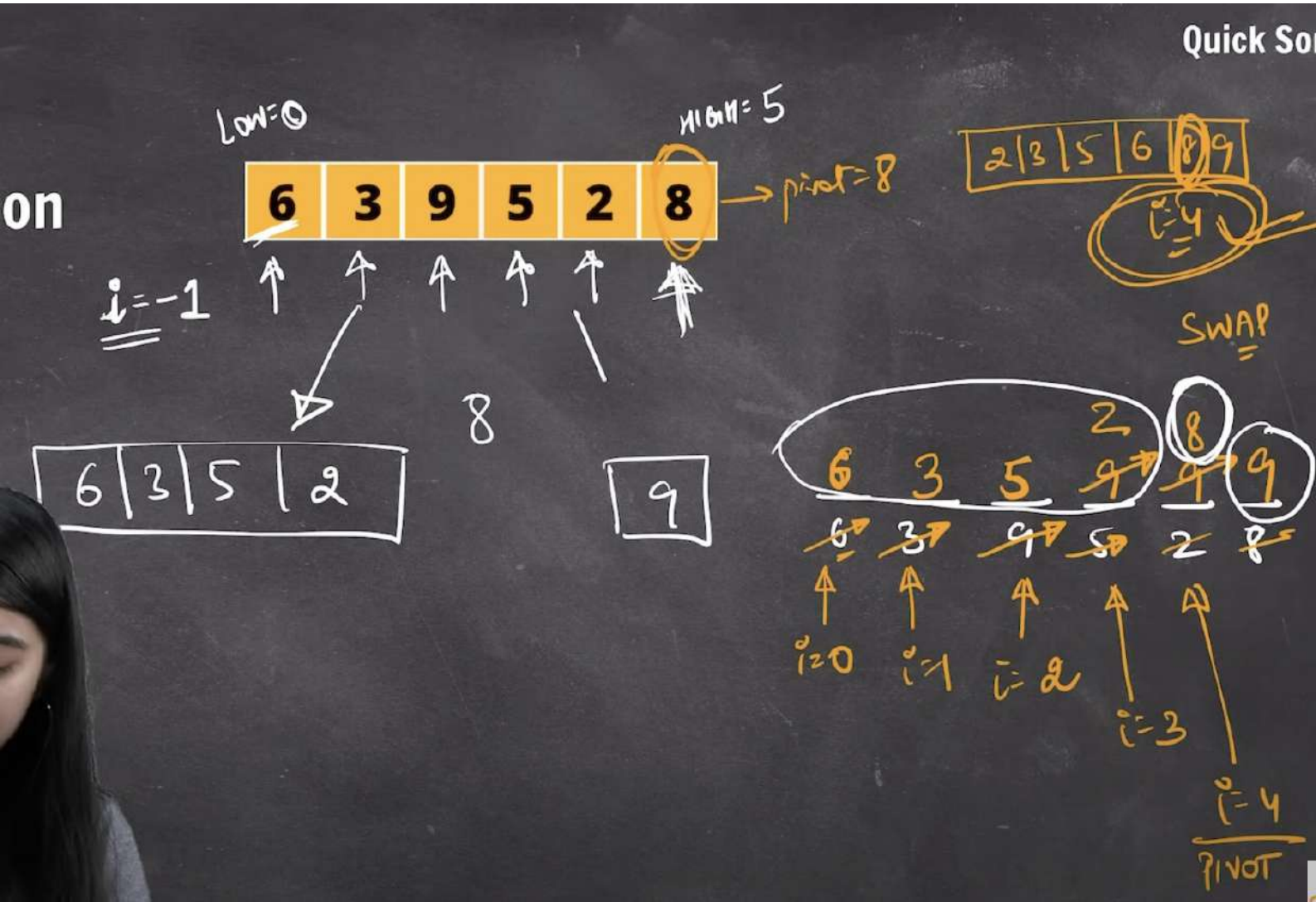
sorted


Approach
Pivot & Partition


1 random
2 median
3 1st el.
4 last el.


6 3 9 5 2 8
pivot
smaller
larger
8
6 3 5 2
pivot
9
2
6 3 5
5
3
6


Quick S
6 3 9 5 2 8
pivot
smaller
larger
8
pivot
6 3 5 2
9
sorted
2
6 3 5
5
3
6
2 3 5 6 8 9
i=4
SORTED
2 3 5 6 8 9


Quick So
on
Low=0
High=5
6 3 9 5 2 8
pivot=8
i=-1
8
6 3 5 2
9
2 3 5 6 8 9
i=4
SWAP
2
8
6 3 5 9 9 9
6 3 9 5 2 8
i=0
i=1
i=2
i=3
i=4
PIVOT

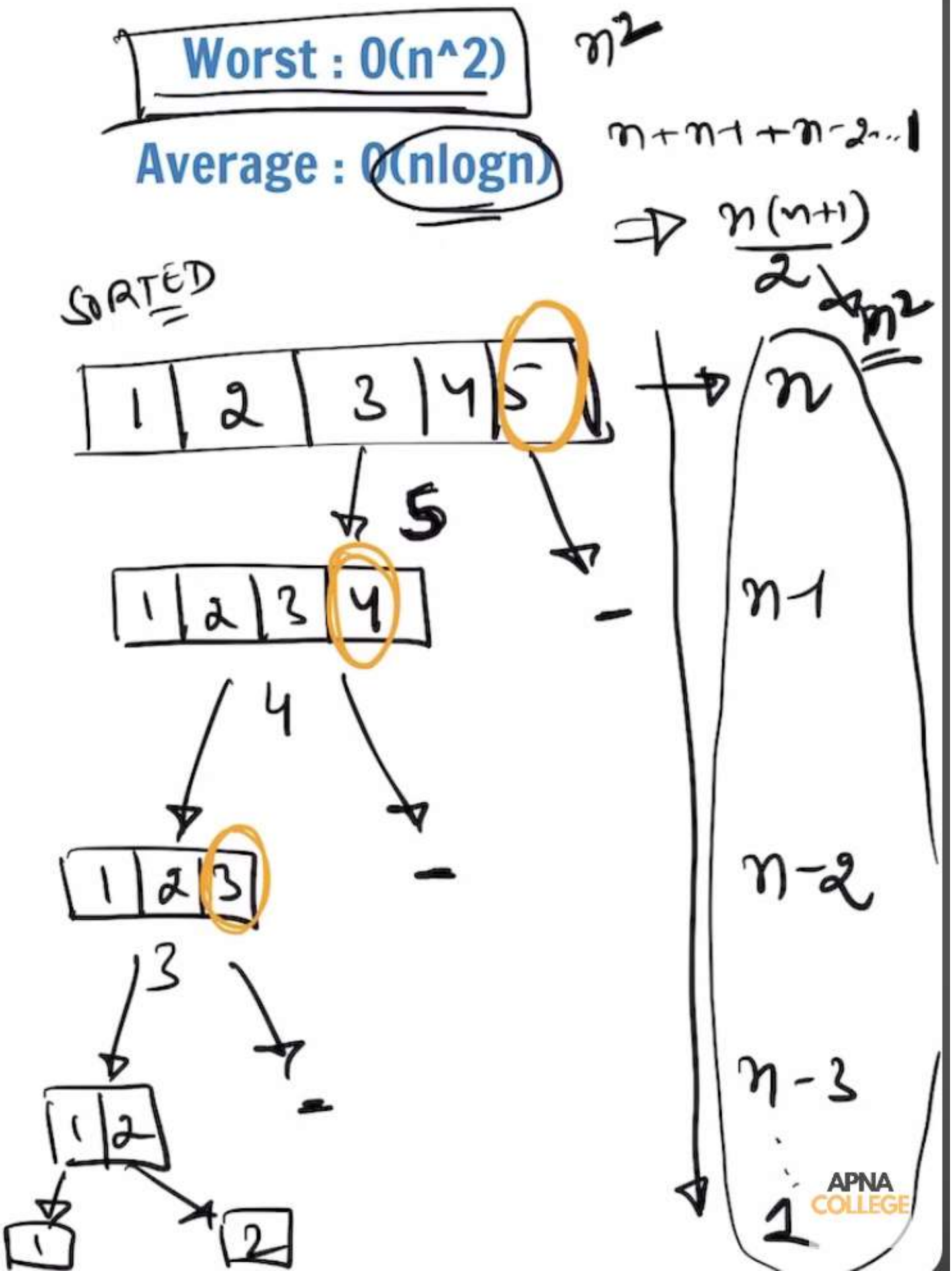
nlogn

Quick Sort

Time Complexity

Worst : O(n^2)

Average : O(nlogn)



Code

```java
public static void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```java
public static int partition(int[] arr, int low, int high) {
    int pivot = arr[high];
    int i = low-1;

    for(int j=low; j<high; j++) {
        if (arr[j] < pivot) {
            i++;
            //swap
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    //swap with pivot
    i++;
    int temp = arr[i];
    arr[i] = arr[high];
    arr[high] = temp;
    return i;
}
```

Time Complexity

Worst : O(n^2)

Average : O(nlogn)

Important

Worst case occurs when pivot is always the smallest or the largest element.