Quick Sort

Course on Sorting and Searching

Rohit Mazumder • Lesson 4 • Feb 22, 2021

Agenda:

1. Sorting Problem (Recap)
2. The partitioning problem
3. Quicksort
4. Critical Analysis of Quicksort
5. Merge Sort vs Quick Sort
6. QuickSort: Performance Summary
7. Assessment
8. HW

What is the sorting problem?

Arranging elements in a specific order.

Commonly used orders are:

- Ascending order
- Descending order

Example: Arranging the rank list in decreasing order of marks, Alphabetic ordering.

Target: Sort an array in ascending order

The Partitioning Problem

Given an array A of N elements, your task is to rearrange this array such that every element that is smaller than A[0] occurs to the left of it and every element larger than or equal to A[0] occurs to its right.

NOTE: There may be multiple possible solutions. Print any of them.

[20, 5, 27, 3, 45, 30, 19, 77, 1] => [1, 5, 3, 19, 20, 77, 45, 27, 30]

Solution 1: Naive Logic?

Partitioning: Lomuto Algorithm

Assume: Pivot element is at the end of the array

NOTE: You can always bring the pivot element at the end of the array, with an extra swap operation! :)

```
algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo
    for j := lo to hi do
        if A[j] < pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[65, 27, 3, 45, 30, 19, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[65, 27, 3, 45, 30, 19, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 27, 65, 45, 30, 19, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 27, 65, 45, 30, 19, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 27, 65, 45, 30, 19, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 19, 65, 45, 30, 27, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 19, 65, 45, 30, 27, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 19, 65, 45, 30, 27, 77, 1, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 19, 1, 45, 30, 27, 77, 65, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 19, 1, 45, 30, 27, 77, 65, 20]

i

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

j

[3, 19, 1, 45, 30, 27, 77, 65, 20]

i

Exit condition encountered:
j has reached the end of the array!
swap(A[high], A[i]) and exit

Core Idea: Search for A[j], such that A[j] < pivot and secure its position by placing it towards the beginning of the array

[3, 19, 1, 20, 30, 27, 77, 65, 45]

Exit condition encountered:
j has reached the end of the array!
swap(A[high], A[i]) and exit

Time, Space Complexity? Inplace? Stability?

Solution-3: Hoare's Algorithm

1. Find i = index of first item from left larger than pivot
2. Find j = index of first item from right smaller than pivot
3. swap(A[i], A[j])
4. Repeat.

Exit condition: j <= i

After exiting swap(A[0], A[j])

j

[20, 5, 27, 3, 45, 30, 19, 77, 1]

i

j

[20, 5, 1, 3, 45, 30, 19, 77, 27]

i

j

[20, 5, 1, 3, 45, 30, 19, 77, 27]

i

j

[20, 5, 1, 3, 19, 30, 45, 77, 27]

i

Exit Condition Encountered!
Swap(A[0], A[j]) and exit

j

[20, 5, 1, 3, 19, 30, 45, 77, 27]

i

Exit Condition Encountered!
Swap(A[0], A[j]) and exit

[19, 5, 1, 3, 20, 30, 45, 77, 27]

Time Complexity, Space Complexity, Inplace, Stability of Hoare's Scheme?

Comparison: Hoare's Scheme vs Lomuto's Scheme

Interesting fact:

Upon partitioning an array, the position that the pivot ends up in, is same the position it would have taken on sorting the array.

Quicksort: Core idea

[5, 27, 3, 45, 30, 19, 77, 1, 20]

[5, 3, 19, 1, 20, 27, 77, 45, 30]

[5, 3, 19, 1, 20, 27, 77, 45, 30]

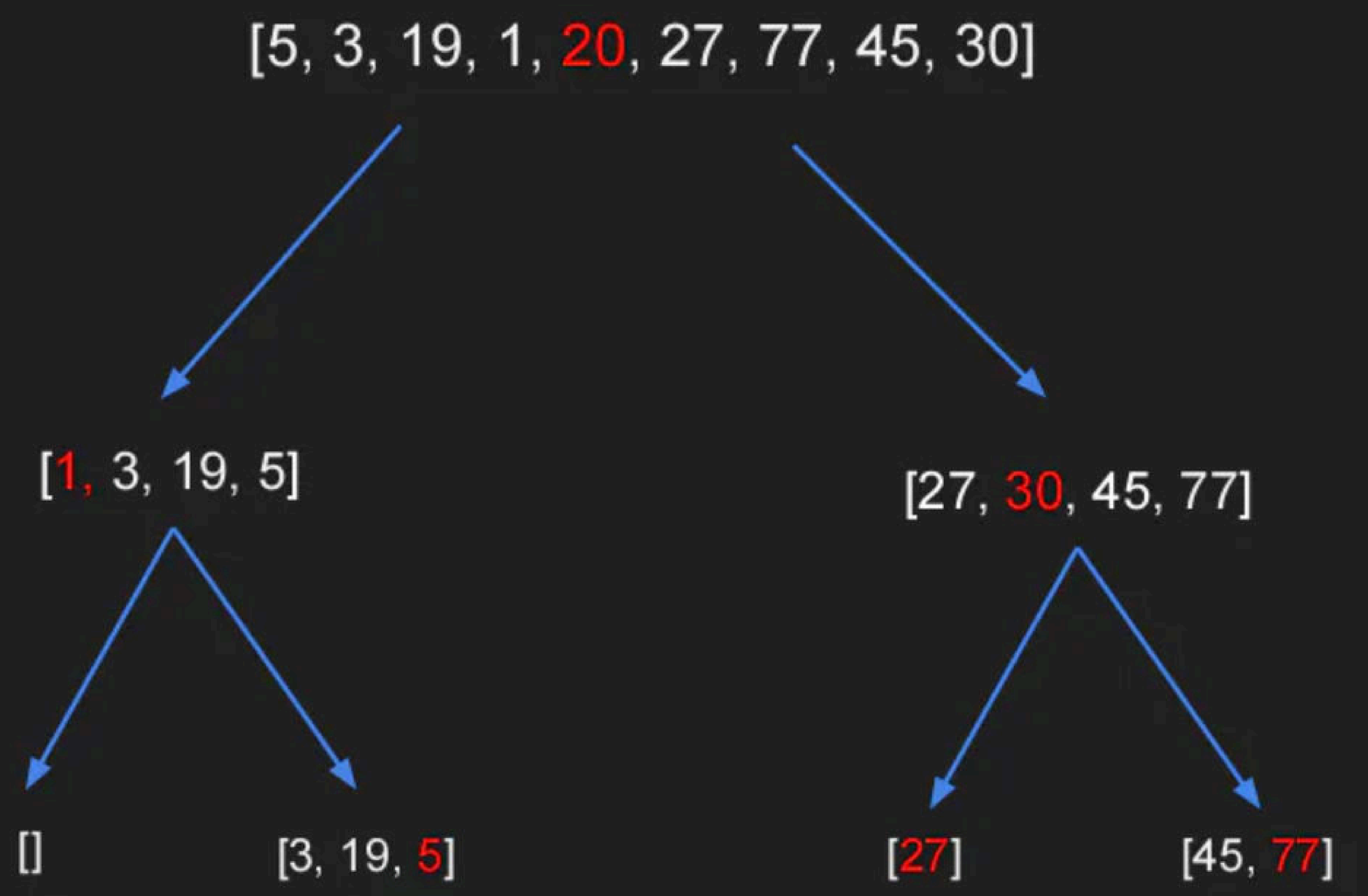
[5, 3, 19, 1]

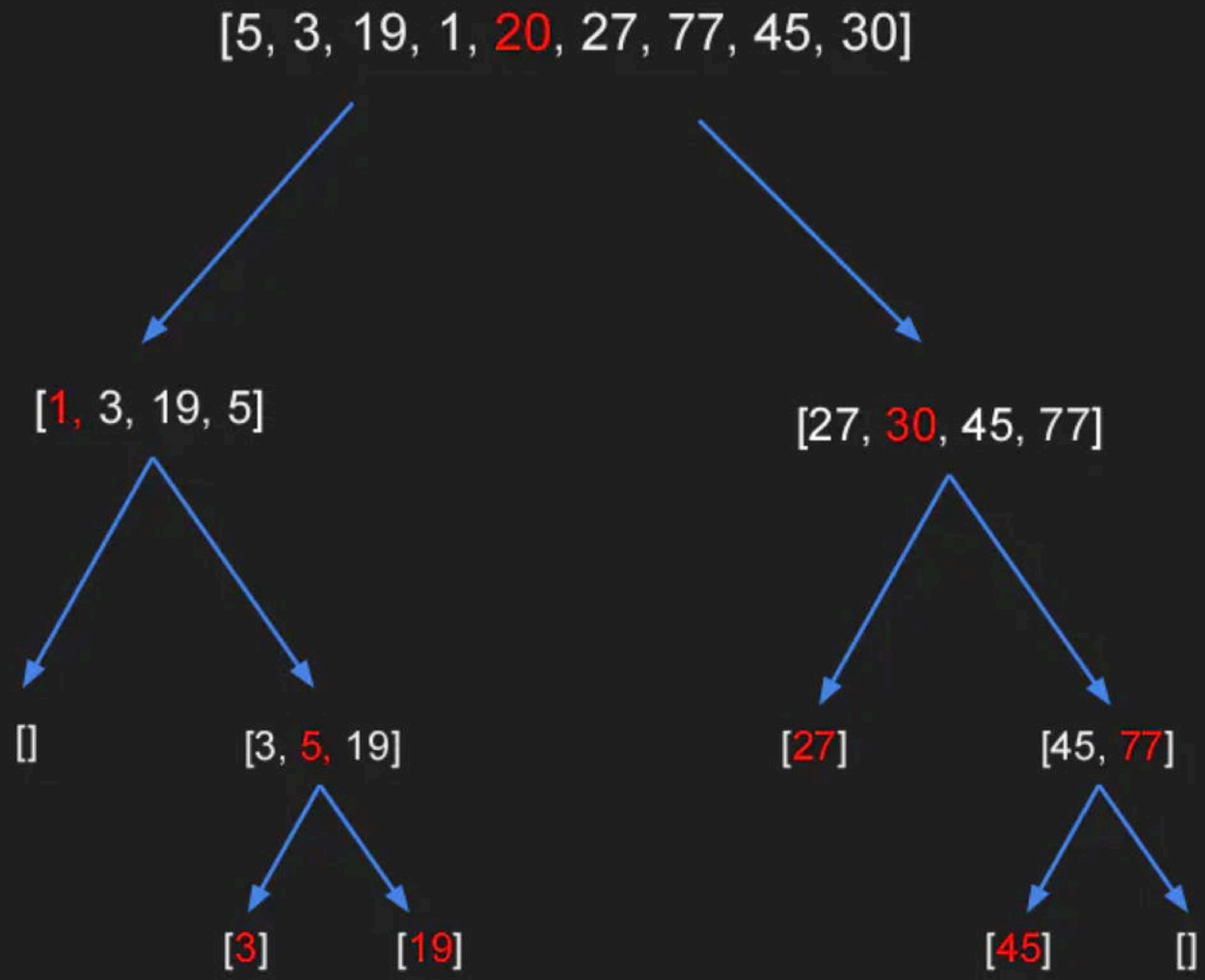
[27, 77, 45, 30]

[5, 3, 19, 1, 20, 27, 77, 45, 30]

[1, 3, 19, 5]

[27, 30, 45, 77]


[5, 3, 19, 1, 20, 27, 77, 45, 30]
[1, 3, 19, 5]
[27, 30, 45, 77]
[]
[3, 19, 5]
[27]
[45, 77]


[5, 3, 19, 1, 20, 27, 77, 45, 30]
[1, 3, 19, 5]
[27, 30, 45, 77]
[]
[3, 5, 19]
[27]
[45, 77]
[3]
[19]
[45]
[]

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p - 1)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo
    for j := lo to hi do
        if A[j] < pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```

Recurrence Relation, Time Complexity & Space Complexity, Stability?

Best Case: When the pivot always comes to the center after partitioning

Worst Case: When the pivot always comes to one of the end when partitioning

Selection of Pivot:

1. First or last element
2. Middle element
3. Random Pivot
4. Median of three

Merge Sort vs QuickSort

QuickSort: Performance Summary

Performance is determined by:

1. Partition Schemes
2. Selection of Pivot
3. Other Optimizations like: Tail Recursion Optimization, Hybriding the algorithm with Insertion Sort etc.

1. What is the recurrence relation for worst case of quick sort?

A. $T(n) = T(n-1)+O(n)$
B. $T(n) = T(n-2) + O(n^2)$
C. $T(n) = 2*T(n/2) + O(1)$
D. $T(n) = 2*T(n/2) + O(n)$

A. $T(n) = T(n-1)+O(n)$
B. $T(n) = T(n-2) + O(n^2)$
C. $T(n) = 2*T(n/2) + O(1)$
D. $T(n) = 2*T(n/2) + O(n)$

Solution : In the worst case the pivot element can be greatest or smallest element.

2. Suppose we have a O(n) time algorithm that finds median of an unsorted array. Now consider a quicksort implementation where we first find the median using the above algorithm, then use median as pivot. What will be the worst case time complexity of this quick sort.?

A. $O(n^2)$
B. O(nlogn)
C. O(nloglogn)
D. O(n)

A. $O(n^2)$
B. $O(n\log n)$
C. $O(n\log\log n)$
D. $O(n)$

Solution : After this the recurrence becomes same as merge sort : $T(n) = 2*T(n/2) + O(n)$ which is known to have $O(n\log n)$

3. Suppose we are sorting an array of eight integers using quicksort and we have just finished partitioning with the array looking like this : [1,5,1,7,9,12,11,10]

A. Pivot could be 7 or 9
B. Pivot could be 7 but not 9
C. Pivot is not 7 but could be 9
D. Neither 7 nor 9 is the pivot.

A. Pivot could be 7 or 9
B. Pivot could be 7 but not 9
C. Pivot is not 7 but could be 9
D. Neither 7 nor 9 is the pivot.

Solution : For every element check if all small numbers are on the left and all big numbers are on the right.

H.W.

1. Segregate positives and negatives: Revisited!