CS303 - Algorithms and Data Structures

Lecture 6

Quick Sort

Professor : Mahmut Unan – UAB CS

Agenda

- Quick Sort
- Median of Three-Five-Seven….
- Time Complexity of Quick Sort
- Comparison Sorts
- Sorting in Linear Time
- Counting Sort
- Radix Sort
- Bucket Sort

Divide-and-conquer approach
Quick Sort

- Quick Sort is a fast sorting algorithm and it is widely applied in practice.

- Like merge sort, quicksort uses divide-and-conquer, and so it's a recursive algorithm.

- It consists 3 main steps ;
    - Choose a pivot value
    - Partition
    - Sort both parts

Divide-and-conquer paradigm

• **Divide:** Partition (rearrange) the array A[p..r] into

two subarrays A[p..q-1] and A[q+1..r] such

– each element of A[p..q-1] is less than or equal to A[q]

– A[q] is less than or equal to each element of A[q+1..r]

– A[q] is called **pivot** - the element around which to partition

(Note that element A[q] is in it's final position in the array)

• **Conquer:** Sort the two subarrays A[p..q-1] and

A[q+1..r] by recursive calls to quicksort

• **Combine:** Since the subarrays are already sorted,

there is no work needed to combine the subarrays,

the entire array A[p..r] is now sorted

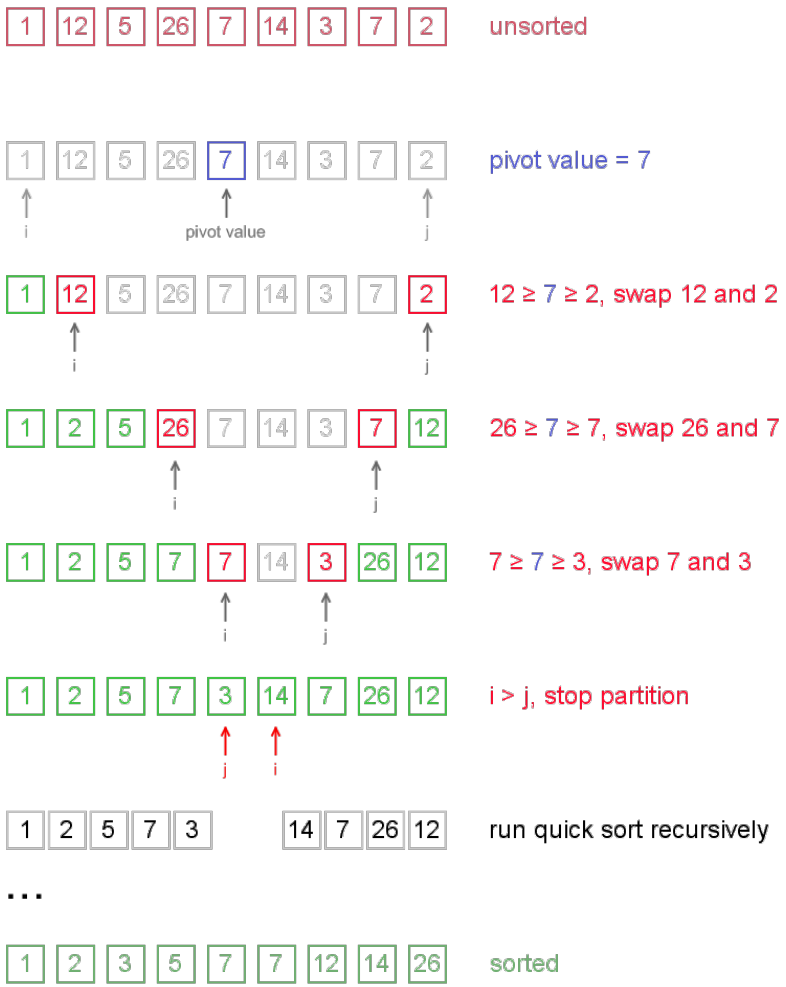
```
QUICKSORT(A, p, r)
  if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

There are two indices **i** and **j** and at the very beginning of the partition algorithm **i** points to the first element in the array and **j** points to the last one.
Then algorithm moves **i** forward, until an element with value greater or equal to the pivot is found.
Index **j** is moved backward, until an element with value lesser or equal to the pivot is found.
If **i ≤ j** then they are swapped and i steps to the next position (**i + 1**), j steps to the previous one **(j - 1)**. Algorithm stops, when **i** becomes greater than **j**.
After partition, all values before **i-th** element are less or equal than the pivot and all values after **j-th** element are greater or equal to the pivot.

Example 1

Arr = [1, 12, 5, 26, 7, 14, 3, 7, 2]



http://www.algolist.net/Algorithms/Sorting/Quicksort

Example 2

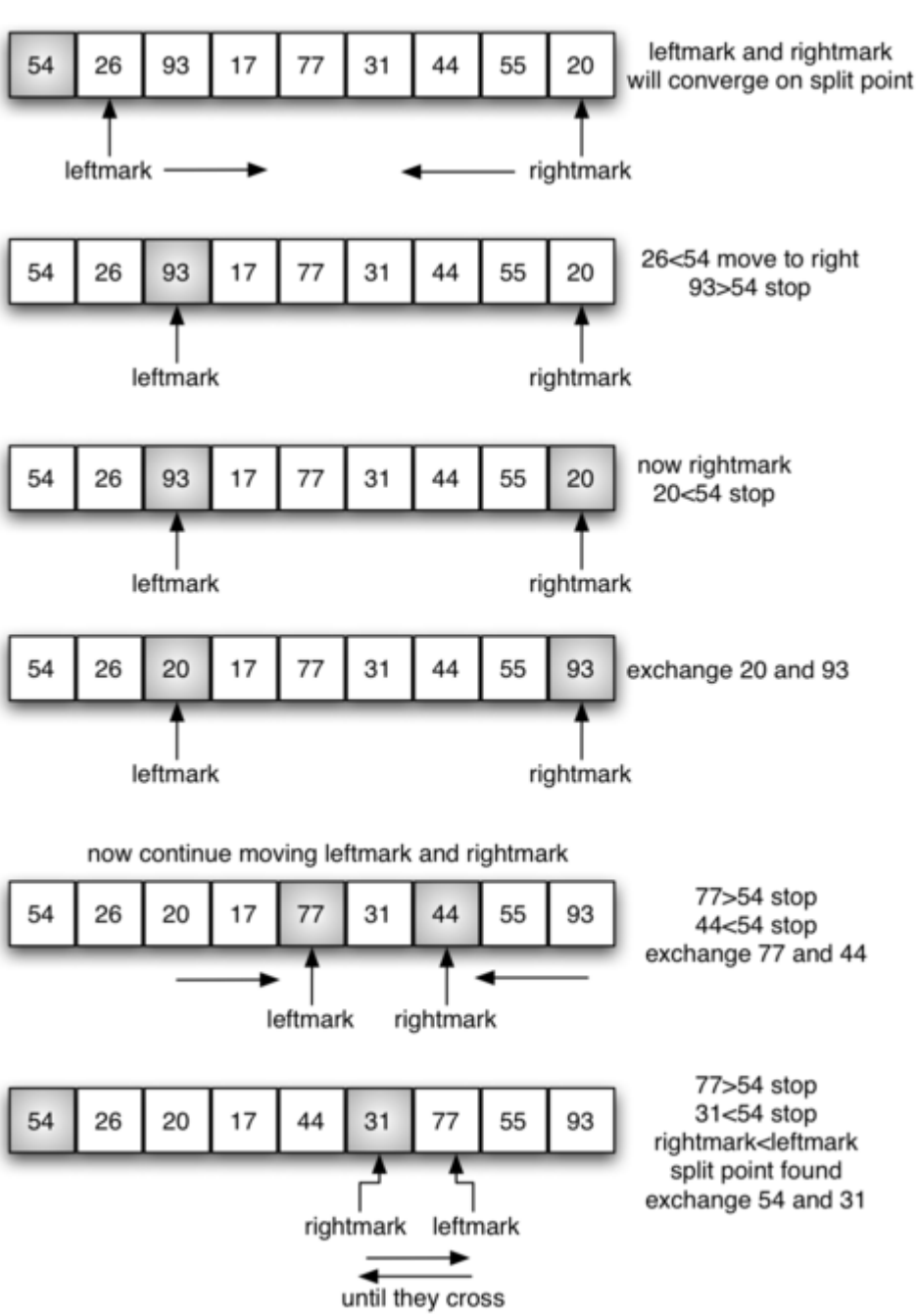
Arr = [57, 22 ,3, 1, 64, 25, 6, 19, 49, 77, 13]

Example 3

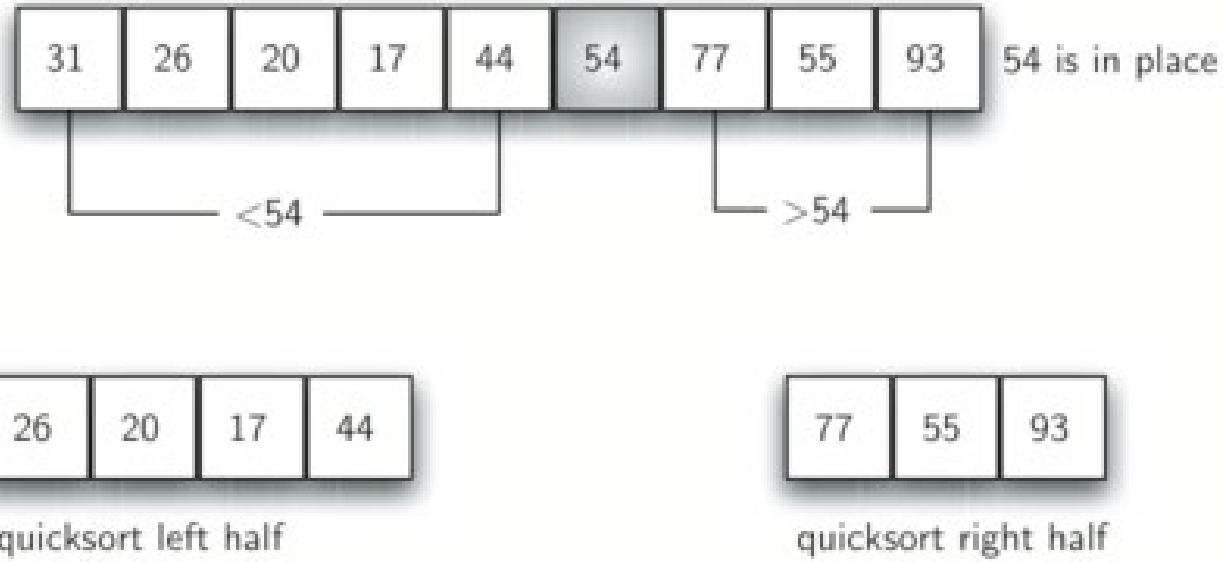
Arr = [11, 150 ,4, 100, 110, 520, 177, 129, 24]

Example 4

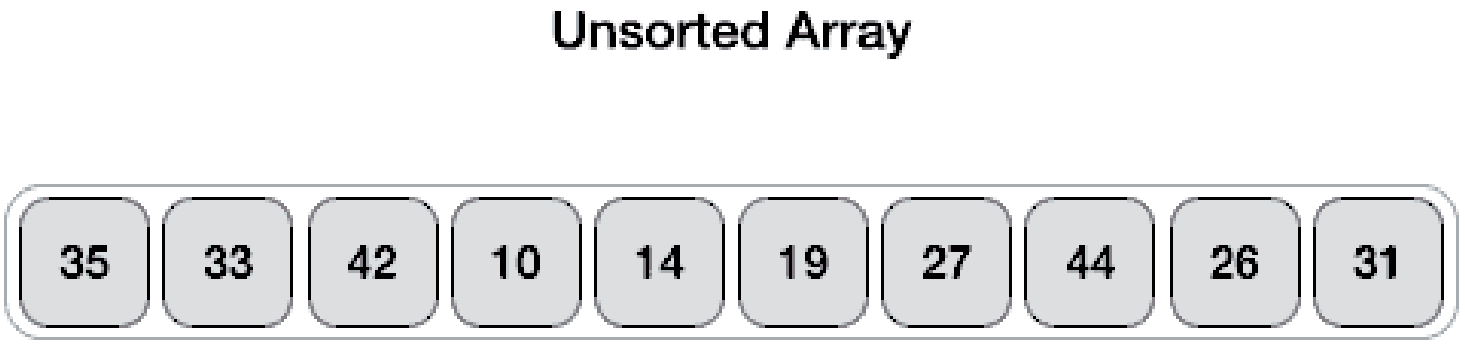
Arr = [U, A, B, C, S, A, L]



https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheQuickSort.html


31
26
20
17
44
54
77
55
93
54 is in place
<54
>54
31
26
20
17
44
quicksort left half
77
55
93
quicksort right half

Assign the last element as a pivot



https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm

Median-of-three Partitioning

- Pivot = median of the **left-most, right-most** and **center** element of the array.

- A = [**3** 6 8 2 **9** 10 1 21 **12**]
- median = [3, **9**, 12] → pivot = 9

- A2 = [**12** 10 5 **7** 19 21 **14**]
  - = [12 7 14] Sorted = [7 **12** 14] → pivot = 12

Median-of-three Partitioning

- Randomly choose three elements from the subarray, and take median of the three as the pivot….
- Median of five?
- Median of nine?
- ….
- Different versions of the quicksort is available

Time Complexity

- ***Worst Case:*** occurs when the partition process always picks greatest or smallest element as pivot. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot.

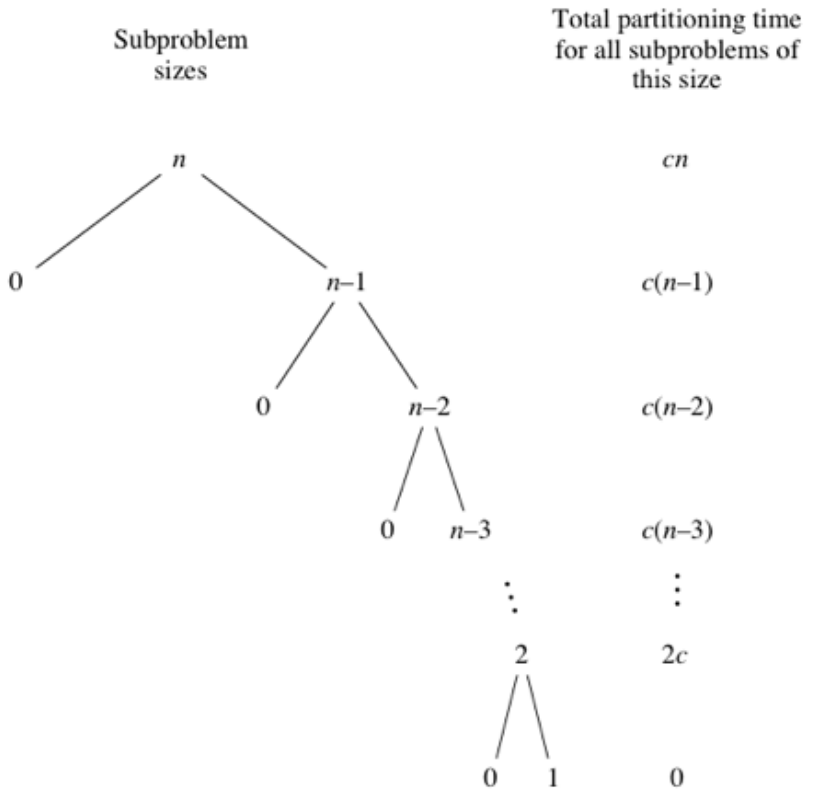
- ***Best Case:*** occurs when the partition process always picks the middle element as pivot.

Worst Case Running Time

- When quicksort always has the most unbalanced partitions possible,
  - then the original call takes cn time for some constant c,
  - the recursive call on $n-1$ elements takes $c(n-1)$ time,
  - the recursive call on $n-2$ elements takes $c(n-2)$ time,
  - and so on.

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Worst Case Running Time / 2



$cn + c(n-1) + c(n-2) + \cdots + 2c$
$= c\,(n + (n-1) + (n-2) + \cdots + 2)$
$= c\,(\,(n+1)\,(n/2) - 1\,)\,.$

$\Theta(n\text{^}2)$

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

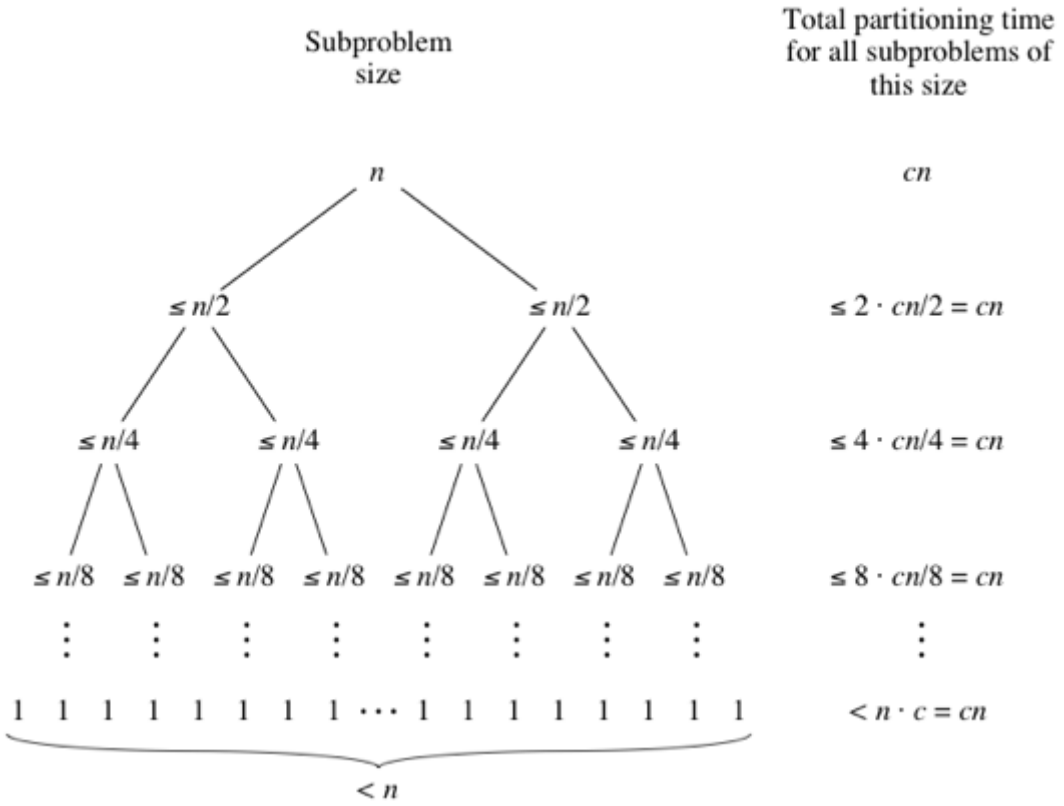
Best Case Running Time

Quicksort's best case occurs when the partitions are as evenly balanced as possible: their sizes either are equal or are within 1 of each other.

- The former case occurs if the subarray has an odd number of elements and the pivot is right in the middle after partitioning, and each partition has *(n-1) / 2* elements.
- The latter case occurs if the subarray has an even number *n* of elements and one partition has *n / 2* elements with the other having *n/2 - 1*

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Best Case Running Time / 2



https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

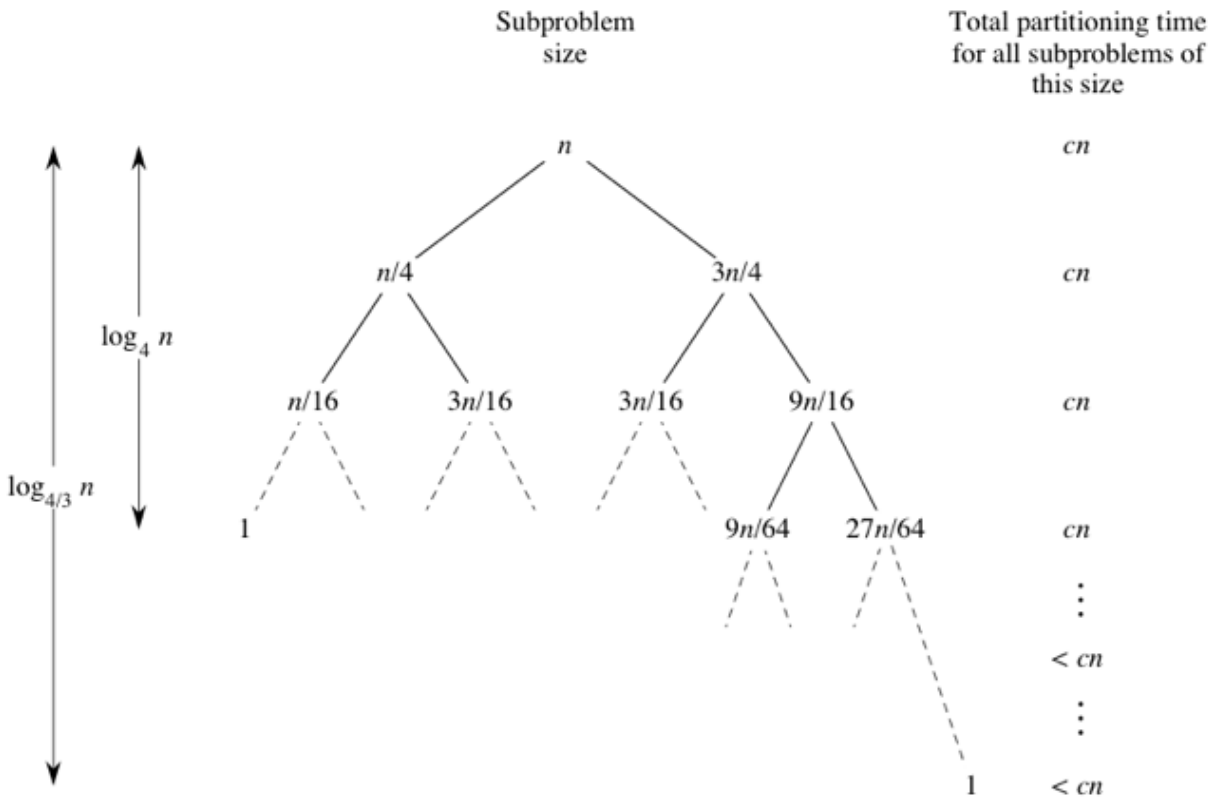
Average Case Running Time

The average-case running time is also $\Theta(n \log_2 n)$

Let's imagine that we don't always get evenly balanced partitions, but that we always get at worst a 3-to-1 split.
That is, imagine that each time we partition, one side gets $3n/4$ elements and the other side gets $n/4$. (To keep the math clean, let's not worry about the pivot.)

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Average Case Running Time / 2



https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Average Case Running Time / 3

$$\log_a n = \frac{\log_b n}{\log_b a}$$

for all positive numbers a, b, and n. Letting $a = 4/3$ and $b = 2$, we get that

$$\log_{4/3} n = \frac{\log_2 n}{\log_2(4/3)} \ ,$$

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Time Complexity of Quick Sort

- Best Case - Θ(n log n)
- Average Case - Θ(n log n)
- Worst Case - Θ(n^2)

Median of Three

- Aims to improve worst case scenario
  - ~(n log n)

- If you are interested in, read the following research paper;
- https://github.com/brucelilly/quickselect/blob/master/lib/libmedian/doc/pub/generic/paper.pdf

Quiz

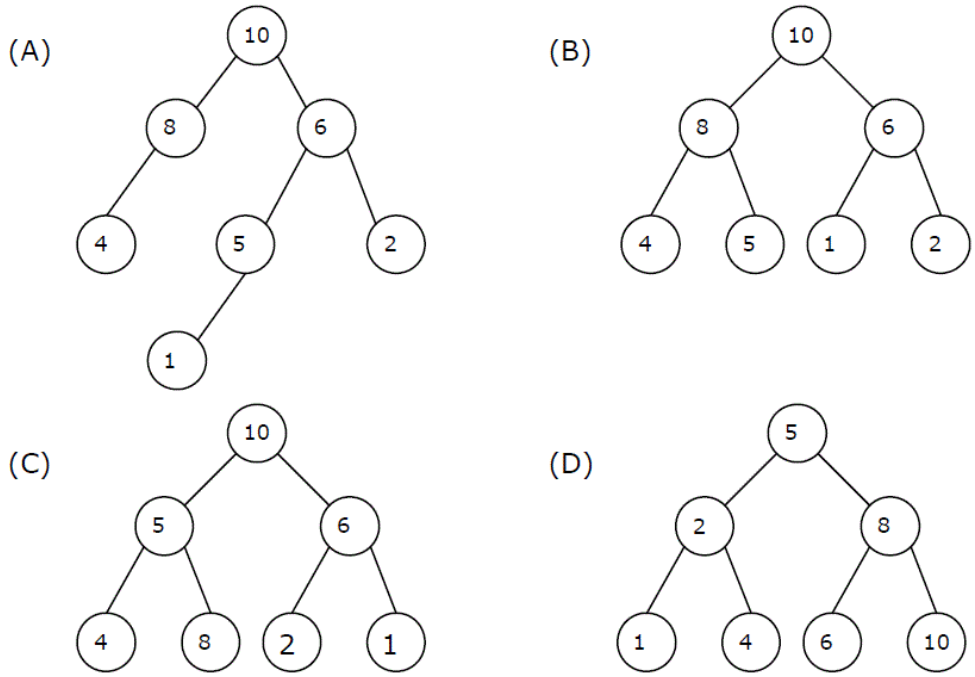
- What is the time complexity of **Build Heap** operation. Build Heap is used to build a max(or min) binary heap from a given array.
- A) O(n log(n))
- B) O(n)
- C) O(n^2)
- D) O(log (n))

Quiz

- What is the time complexity of **Build Heap** operation. Build Heap is used to build a max(or min) binary heap from a given array.
- A) O(n log(n))
- **B) O(n)**
- C) O(n^2)
- D) O(log (n))

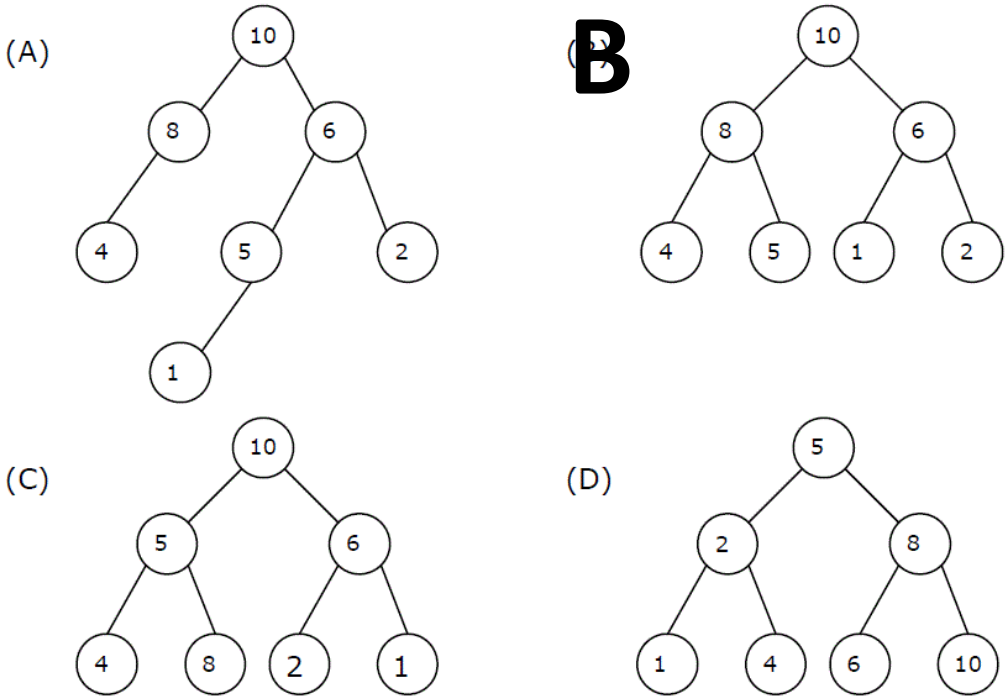
Quiz

- A max-heap is a heap where the value of each parent is greater than or equal to the values of its children. Which of the following is a max-heap?



Quiz

- A max-heap is a heap where the value of each parent is greater than or equal to the values of its children. Which of the following is a max-heap?



Quiz

- In a binary max heap containing n numbers, the smallest element can be found in time
- A) O(n)
- B) O(log n)
- C) O(log log n)
- D) O (n log n)

Quiz

- In a binary max heap containing n numbers, the smallest element can be found in time
- **A) O(n)**
- B) O(log n)
- C) O(log log n)
- D) O (n log n)

Quiz

- What is the best case complexity for heap sort algorithm

A) O(n)

B) O(n*n)

C) O(nlogn)

D) O(logn)

Quiz

- What is the best case complexity for heap sort algorithm

A) O(n)

B) O(n*n)

C) O(nlogn)

D) O(logn)

Quiz

- If a max heap is implemented using a partially filled array called data, and the array contains n elements (n > 0), where is the entry with the greatest value? (most appropriate answer)

a. data[0]
b. data[n-1]
c. data[n]
d. data[2*n + 1]
e. data[2*n + 2]

Quiz

- If a max heap is implemented using a partially filled array called data, and the array contains n elements (n > 0), where is the entry with the greatest value? (most appropriate answer)

a. data[0]
b. data[n-1]
c. data[n]
d. data[2*n + 1]
e. data[2*n + 2]

Quiz

- Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).
- A) Quick Sort
- B) Heap Sort
- C) Merge Sort
- D) Insertion Sort

Quiz

- Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).
- A) Quick Sort
- B) Heap Sort
- C) Merge Sort
- **D) Insertion Sort**

Quiz

- Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this: 2 5 1 7 9 12 11 10 Which statement is correct?
- A) The pivot could be either the 7 or the 9
- B) The pivot could be the 7, but it is not the 9
- C) The pivot is not the 7, but it could be the 9
- D) Neither the 7 nor the 9 is the pivot.

Quiz

- Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this: 2 5 1 7 9 12 11 10 Which statement is correct?
- **A) The pivot could be either the 7 or the 9**
- B) The pivot could be the 7, but it is not the 9
- C) The pivot is not the 7, but it could be the 9
- D) Neither the 7 nor the 9 is the pivot.

Quiz

- In quick sort, for sorting n elements, the (n/4)th smallest element is selected as pivot using an O(n) time algorithm. What is the worst case time complexity of the quick sort?
- A) Θ(n)
- B) Θ(nLogn)
- C) Θ(n^2)
- D) Θ(n^2 log n)

Quiz

- In quick sort, for sorting n elements, the (n/4)th smallest element is selected as pivot using an O(n) time algorithm. What is the worst case time complexity of the quick sort?
- A) Θ(n)
- **B) Θ(nLogn)**
- C) Θ(n^2)
- D) Θ(n^2 log n)

Quiz

- Let P be a QuickSort Program to sort numbers in ascending order using the **first element** as pivot. Let t1 and t2 be the number of comparisons made by P for the inputs {1, 2, 3, 4, 5} and {4, 1, 5, 3, 2} respectively. Which one of the following holds?
- A) t1 = 5
- B) t1<t2
- C) t1>t2
- D) t1=t2

Quiz

- Let P be a QuickSort Program to sort numbers in ascending order using the **first element** as pivot. Let t1 and t2 be the number of comparisons made by P for the inputs {1, 2, 3, 4, 5} and {4, 1, 5, 3, 2} respectively. Which one of the following holds?
- A) t1 = 5
- B) t1<t2
- **C) t1>t2**
- D) t1=t2

Sample Interview Questions

- Given an array which contains N-2 numbers in unsorted order, find two missing numbers?
- How to merge two sorted linked list, write a program in your favorite programming language e.g. Java, C++, or Python
- How to check if a binary tree is balanced or not?
- When does the worst case of QuickSort occur?
- Given a big array, how to efficiently find k’th largest element in it?

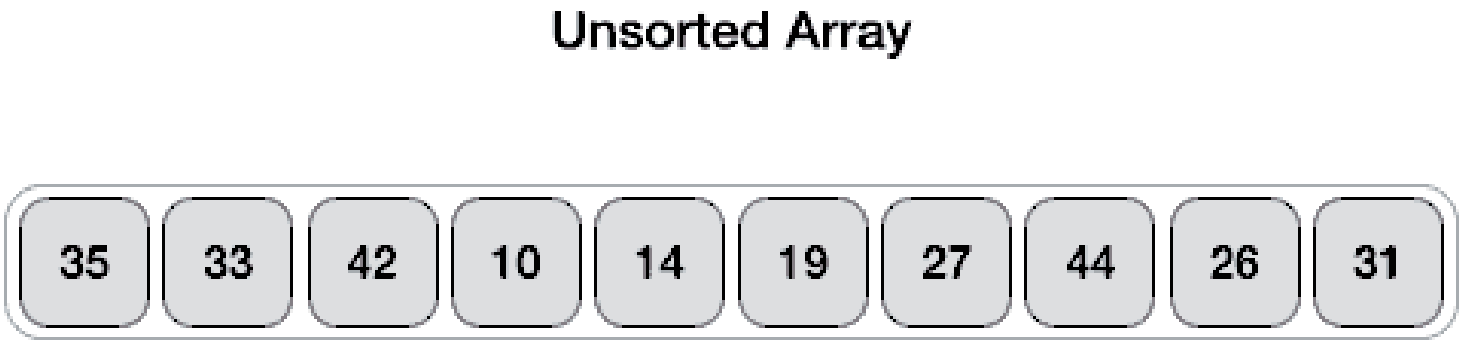
Sample Interview Questions / 2

- You are given an array with duplicates. You have to sort the array with decreasing frequency of elements. If two elements have the same frequency, sort them by their actual value in increasing order.
- Ex: [2 3 5 3 7 9 5 3 7]
- Output: [3 3 3 5 5 7 7 2 9]

References

- https://ctsasikumar.wordpress.com/2007/10/27/heap-sort-interview-question-part-1/
- https://www.geeksforgeeks.org/data-structure-gq/heap-gq/

Assign the last element as a pivot



https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm

Median-of-three Partitioning

- Pivot = median of the **left-most, right-most** and **center** element of the array.

- A = [**3** 6 8 2 **9** 10 1 21 **12**]
- median = [3, **9**, 12] → pivot = 9

- A2 = [**12** 10 5 **7** 19 21 **14**]
  - = [12 7 14] Sorted = [7 **12** 14] → pivot = 12

Median-of-three Partitioning

- Randomly choose three elements from the subarray, and take median of the three as the pivot….
- Median of five?
- Median of nine?
- ….
- Different versions of the quicksort is available

Time Complexity

- ***Worst Case:*** occurs when the partition process always picks greatest or smallest element as pivot. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot.

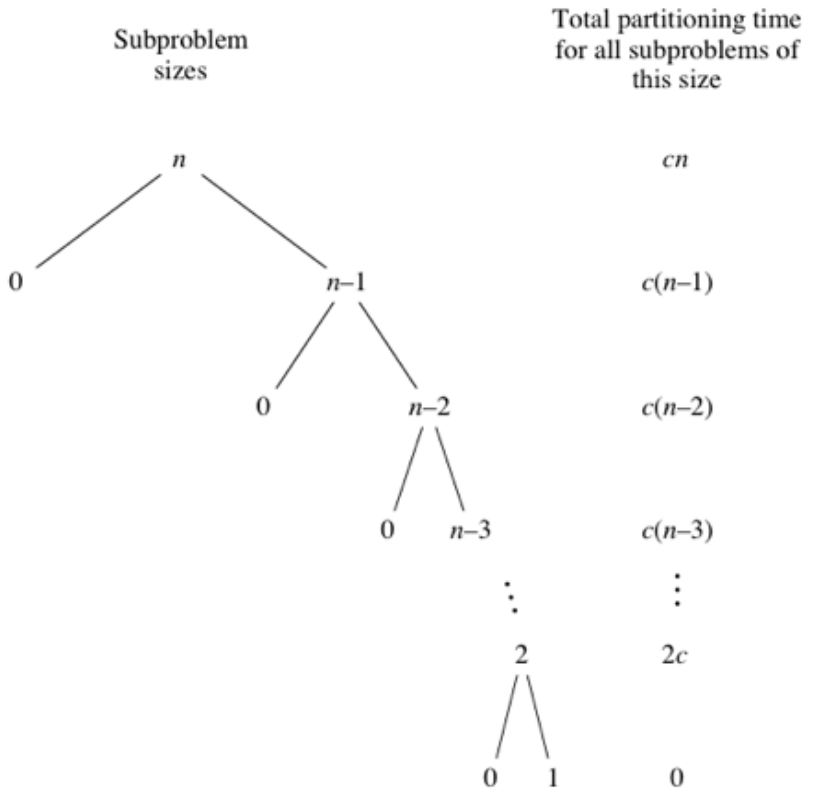
- ***Best Case:*** occurs when the partition process always picks the middle element as pivot.

Worst Case Running Time

- When quicksort always has the most unbalanced partitions possible,
  - then the original call takes cn time for some constant c,
  - the recursive call on $n-1$ elements takes $c(n-1)$ time,
  - the recursive call on $n-2$ elements takes $c(n-2)$ time,
  - and so on.

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Worst Case Running Time / 2



$cn + c(n-1) + c(n-2) + \cdots + 2c$
$= c\,(n + (n-1) + (n-2) + \cdots + 2)$
$= c\,(\,(n+1)\,(n/2) - 1\,)\,.$

$\Theta(n\text{^}2)$

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

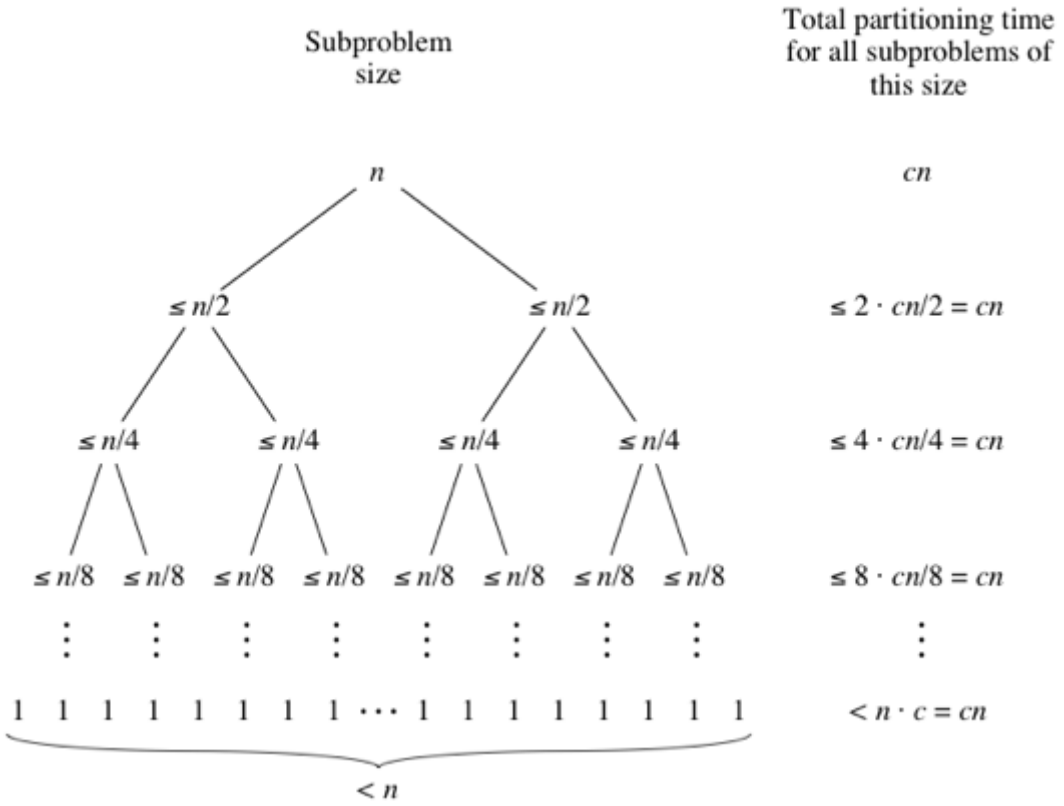
Best Case Running Time

Quicksort's best case occurs when the partitions are as evenly balanced as possible: their sizes either are equal or are within 1 of each other.

- The former case occurs if the subarray has an odd number of elements and the pivot is right in the middle after partitioning, and each partition has *(n-1) / 2* elements.
- The latter case occurs if the subarray has an even number *n* of elements and one partition has *n / 2* elements with the other having *n/2 - 1*

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Best Case Running Time / 2



https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

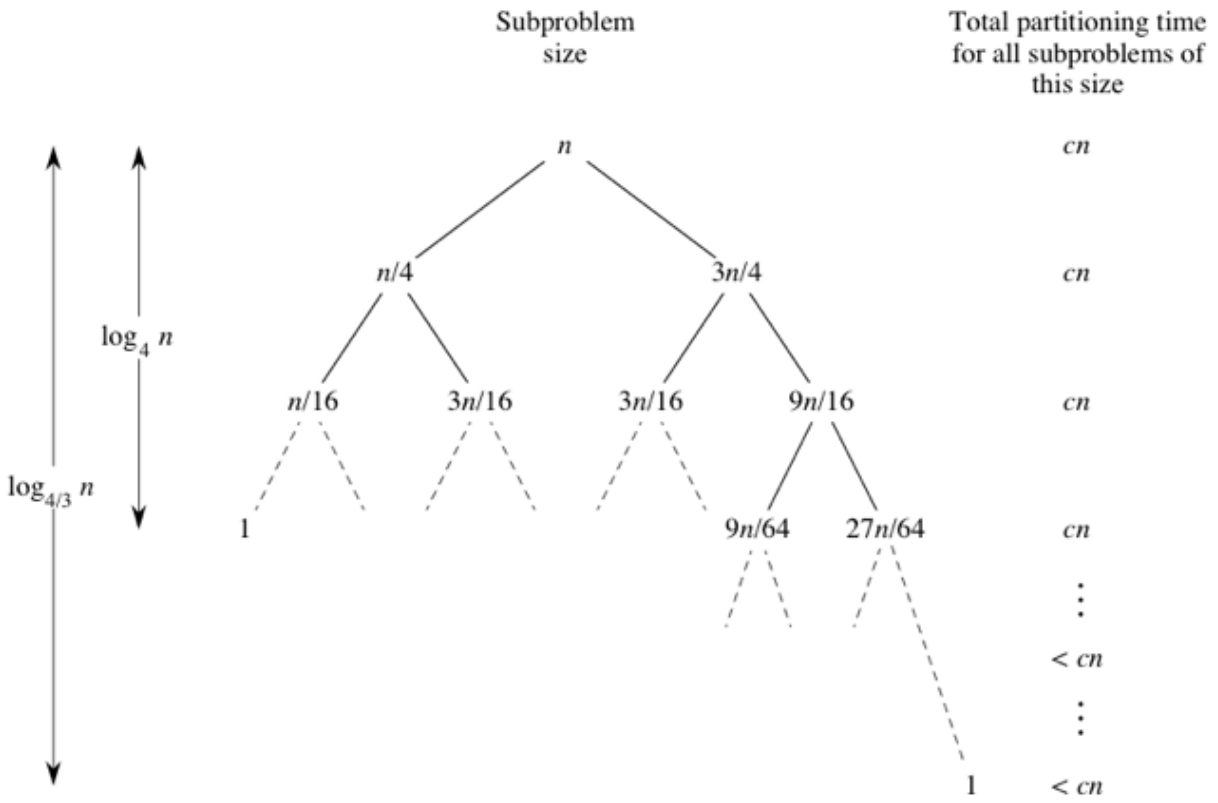
Average Case Running Time

The average-case running time is also $\Theta(n \log_2 n)$

Let's imagine that we don't always get evenly balanced partitions, but that we always get at worst a 3-to-1 split.
That is, imagine that each time we partition, one side gets $3n/4$ elements and the other side gets $n/4$. (To keep the math clean, let's not worry about the pivot.)

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Average Case Running Time / 2



https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Average Case Running Time / 3

$$\log_a n = \frac{\log_b n}{\log_b a}$$

for all positive numbers a, b, and n. Letting $a = 4/3$ and $b = 2$, we get that

$$\log_{4/3} n = \frac{\log_2 n}{\log_2(4/3)} \ ,$$

https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Time Complexity of Quick Sort

- Best Case - Θ(n log n)
- Average Case - Θ(n log n)
- Worst Case - Θ(n^2)