Divide & Conquer: mergesort, quicksort

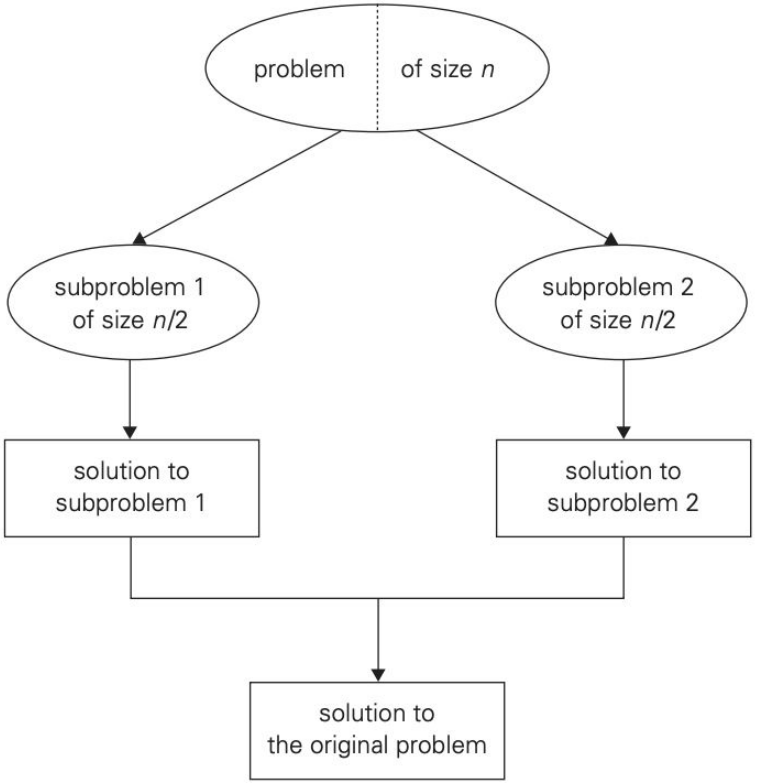
5.1- 5.2

Divide and conquer algorithms

This technique can be divided into the following three parts:

1. **Divide:** This involves dividing the problem into smaller sub-problems.
2. **Conquer:** Solve sub-problems by calling recursively until solved.
3. **Combine:** Combine the sub-problems to get the final solution of the whole problem.

Divide and conquer algorithm flowchart



Some examples of divide and conquer algorithms:

1. Mergesort
2. Quicksort
3. Closest Pair of Points
4. Strassen’s Algorithm

Mergesort algorithm

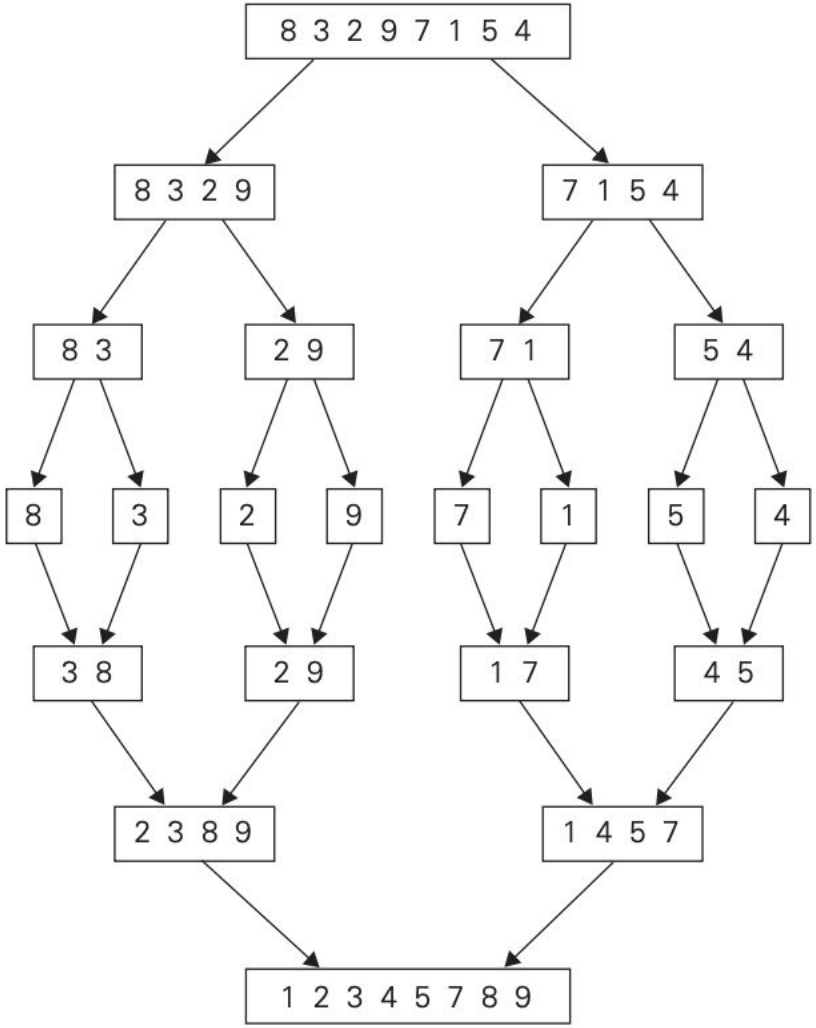
The mergesort algorithm sorts a given array by dividing the input array into two halves, sorting each of them recursively, and then merging the two smaller, sorted arrays into a single sorted one.

Here's how merge sort uses divide-and-conquer:

Divide by finding the number q of the position midway between p and r. Do this step the same way we found the midpoint in binary search: add p and r, divide by 2, and round down.

Conquer by recursively sorting the subarrays in each of the two subproblems created by the divide step. That is, recursively sort the subarray array[p..q] and recursively sort the subarray array[q+1..r].

Combine by merging the two sorted subarrays back into the single sorted subarray array[p..r].


8 3 2 9 7 1 5 4
8 3 2 9
7 1 5 4
8 3
2 9
7 1
5 4
8
3
2
9
7
1
5
4
3 8
2 9
1 7
4 5
2 3 8 9
1 4 5 7
1 2 3 4 5 7 8 9

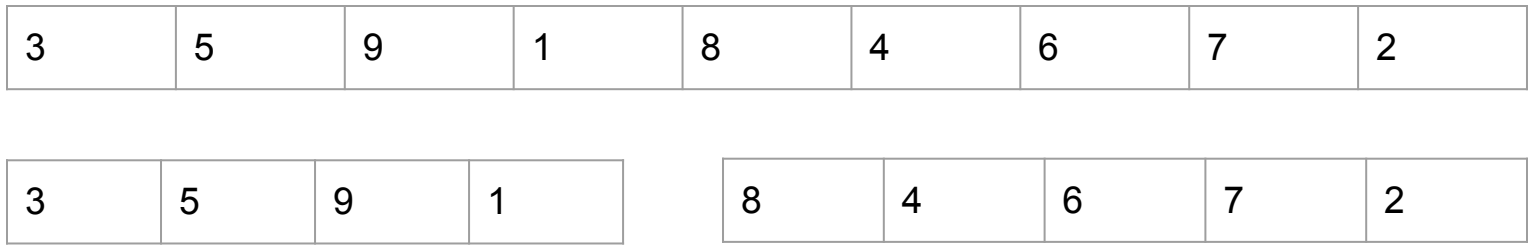
Let’s watch a brief video to diagram merge sort

https://www.youtube.com/watch?v=4VqmGXwpLqc

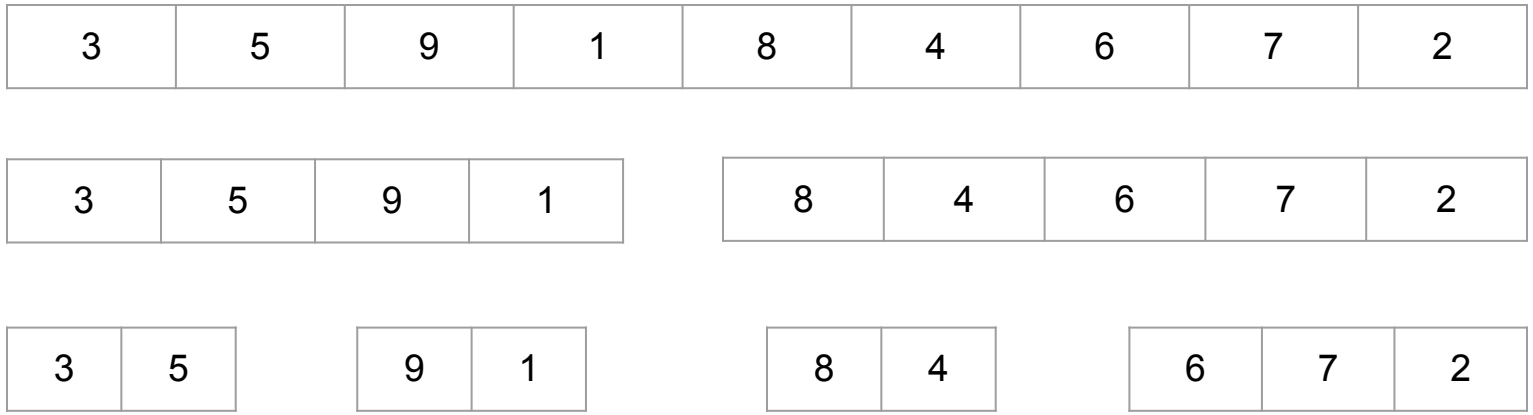
Let’s do a quick example together

3	5	9	1	8	4	6	7	2

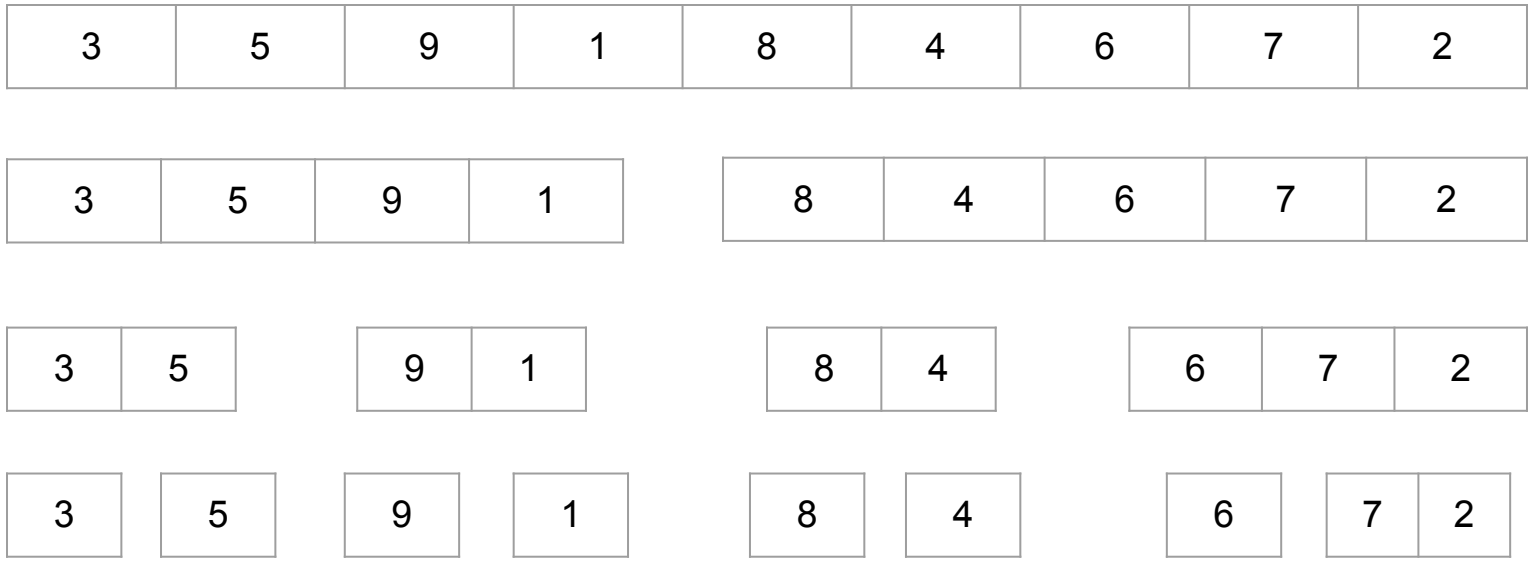
Let’s do a quick example together- divide



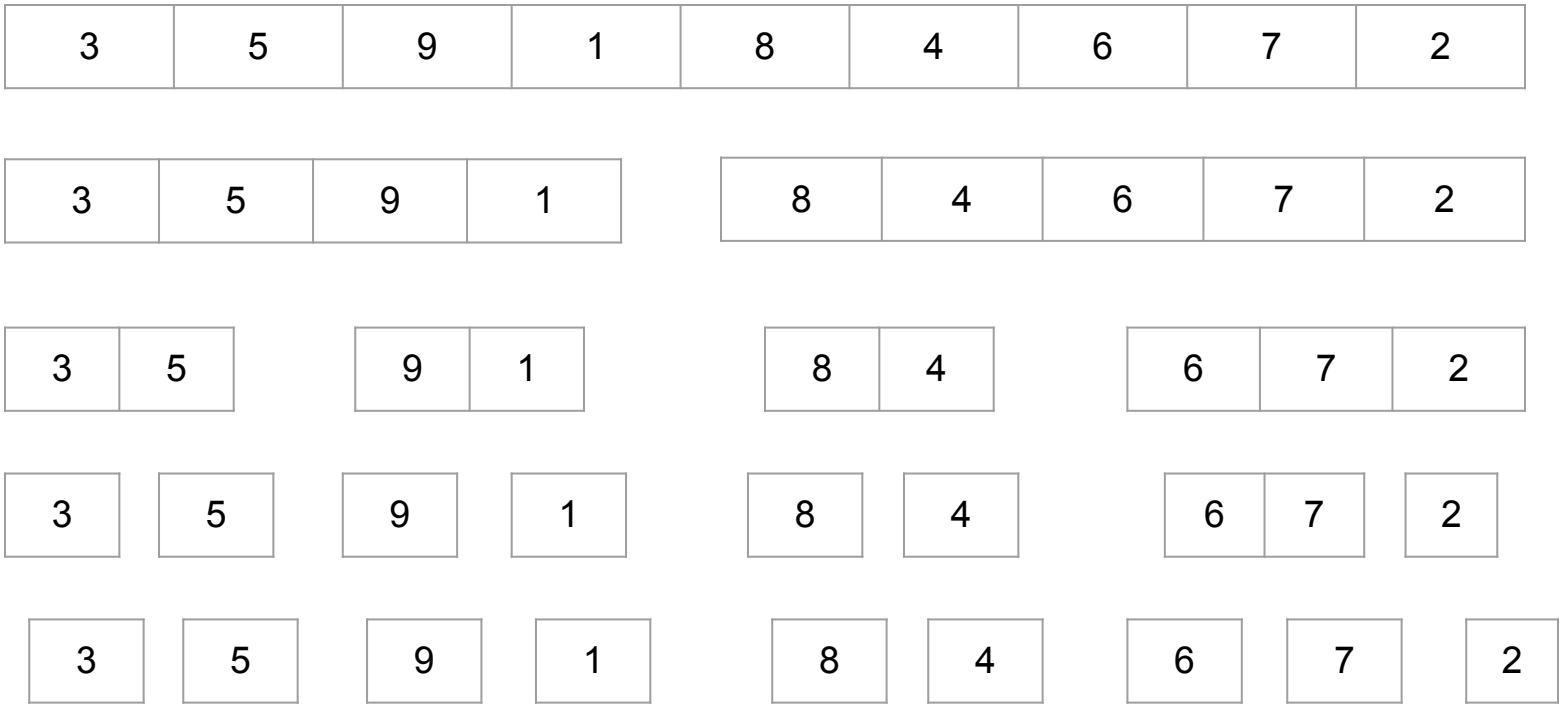
Let’s do a quick example together- divide



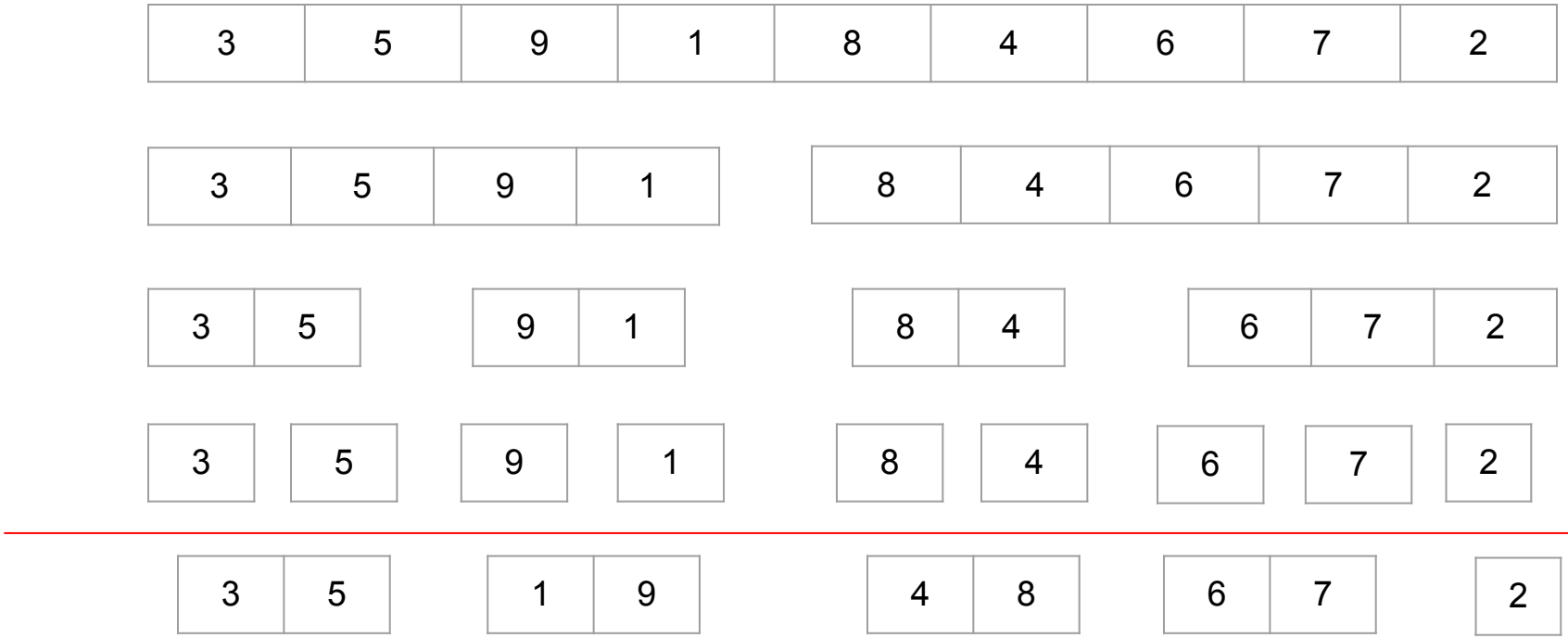
Let’s do a quick example together- divide



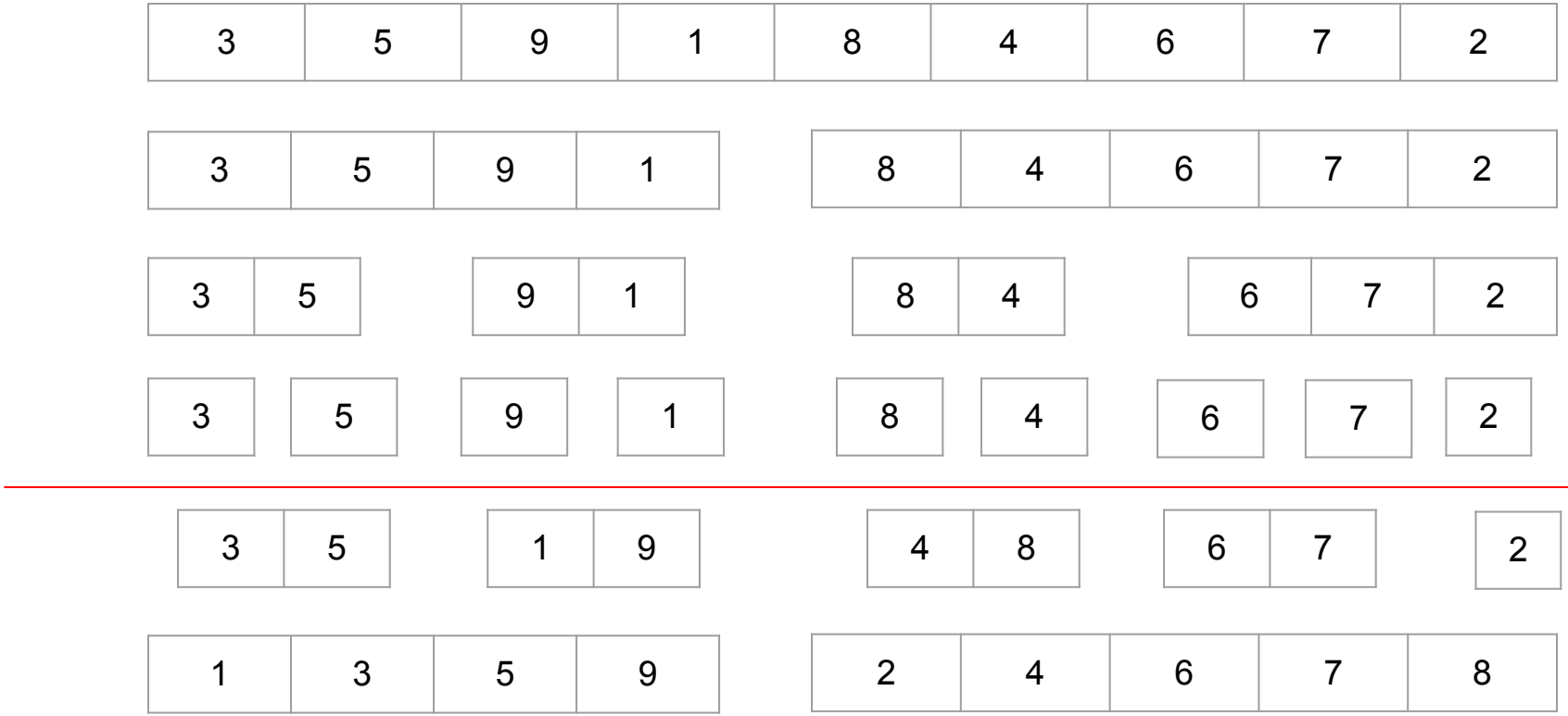
Let’s do a quick example together - we made it to 1s



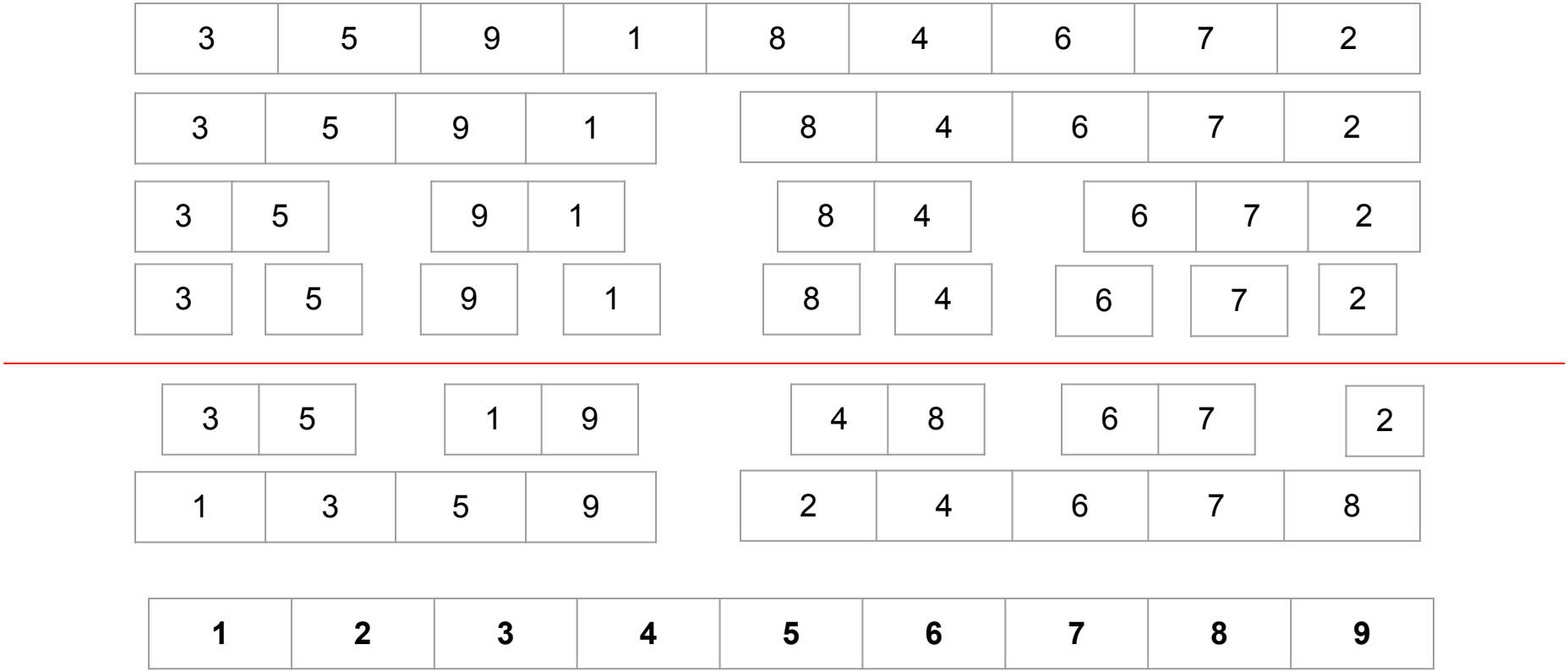
Let’s do a quick example together- combine



Let’s do a quick example together- combine



Let’s do a quick example together- combine



Time complexity of Mergesort

Your book does a little math to derive the time complexity, and we were told in the video

The time complexity for mergesort is:

O(n log n)

Time complexity of Mergesort

Your book does a little math to derive the time complexity, and we were told in the video

The time complexity for mergesort is:

O(n lg n)

This is the same for all 3 notations of time complexity (worst, average, and best case) because mergesort always divides the array into two halves and takes linear time to merge the two halves

So we could also write: Θ(n lg n)

Drawbacks of Mergesort

- Slower comparative to the other sort algorithms for smaller tasks.
- Merge sort algorithm requires an additional memory space of 0(n) for the temporary array.
- It goes through the whole process even if the array is sorted.

Pseudocode for mergesort

- Declare left and right var which will mark the extreme indices of the array
- Left will be assigned to 0 and right will be assigned to n-1
- Find mid = (left+right)/2
- Call mergeSort on (left,mid) and (mid+1,rear)
- Above will continue till left<right
- Then we will call merge on the 2 subproblems

Homework problems!

Please complete textbook problems 5.1: 1abcd, 2abc

Quicksort algorithm

The way that quicksort uses divide-and-conquer is a little different from how merge sort does. In merge sort, the divide step does hardly anything, and all the real work happens in the combine step. Quicksort is the opposite: all the real work happens in the divide step. In fact, the combine step in quicksort does absolutely nothing.

Here's how quicksort uses divide-and-conquer:

Divide by choosing any element in the subarray array[p..r]. Call this element the **pivot**.

Rearrange the elements in array[p..r] so that all elements in array[p..r] that are less than or equal to the pivot are to its left and all elements that are greater than the pivot are to its right. We call this procedure **partitioning**. At this point, it doesn't matter what order the elements to the left of the pivot are in relation to each other, and the same holds for the elements to the right of the pivot. We just care that each element is somewhere on the correct side of the pivot.

Here's how quicksort uses divide-and-conquer:

Conquer by recursively sorting the subarrays array[p..q-1] (all elements to the left of the pivot, which must be less than or equal to the pivot) and array[q+1..r] (all elements to the right of the pivot, which must be greater than the pivot).

Combine by doing nothing. Once the conquer step recursively sorts, we are done. Why? All elements to the left of the pivot, in array[p..q-1], are less than or equal to the pivot and are sorted, and all elements to the right of the pivot, in array[q+1..r], are greater than the pivot and are sorted. The elements in array[p..r] can't help but be sorted!

Let’s watch a brief video to diagram quick sort:

https://www.youtube.com/watch?v=Hoixgm4-P4M

Picking a pivot

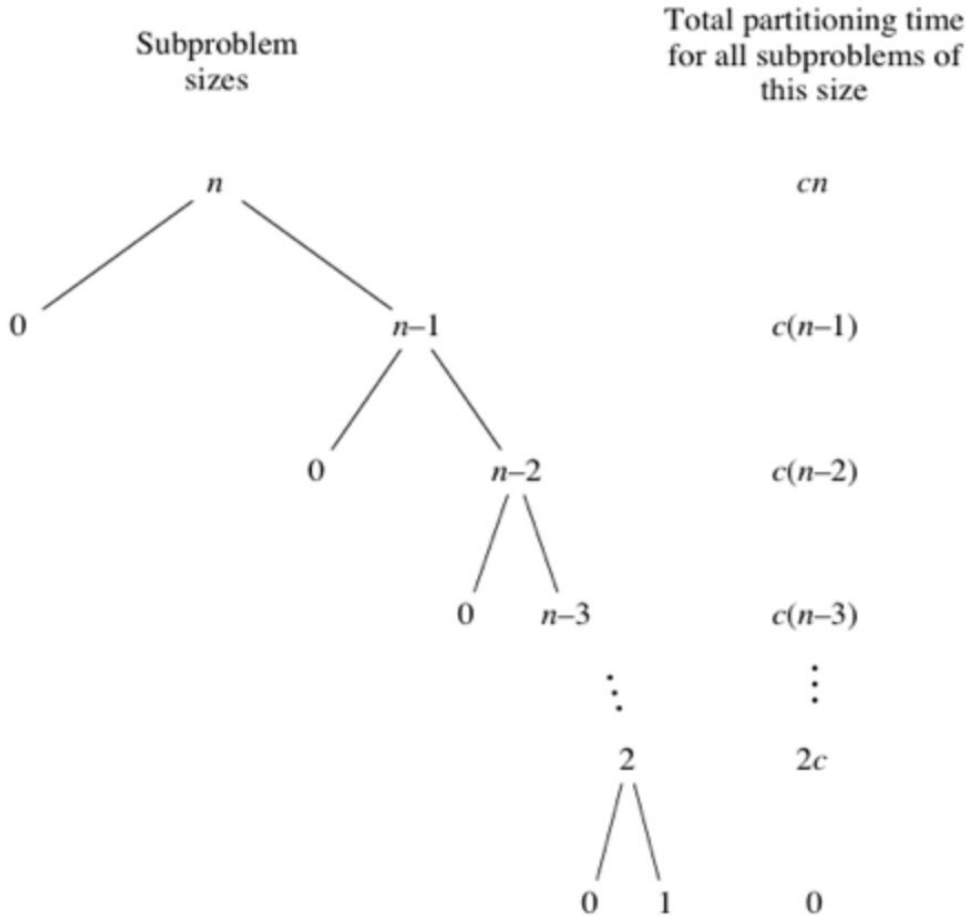
1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

Time complexity of Quicksort

Worst case run time is when we always have the most unbalanced partitions possible

quicksort's worst-case running time is $\Theta(n^2)$

Unbalanced partitions

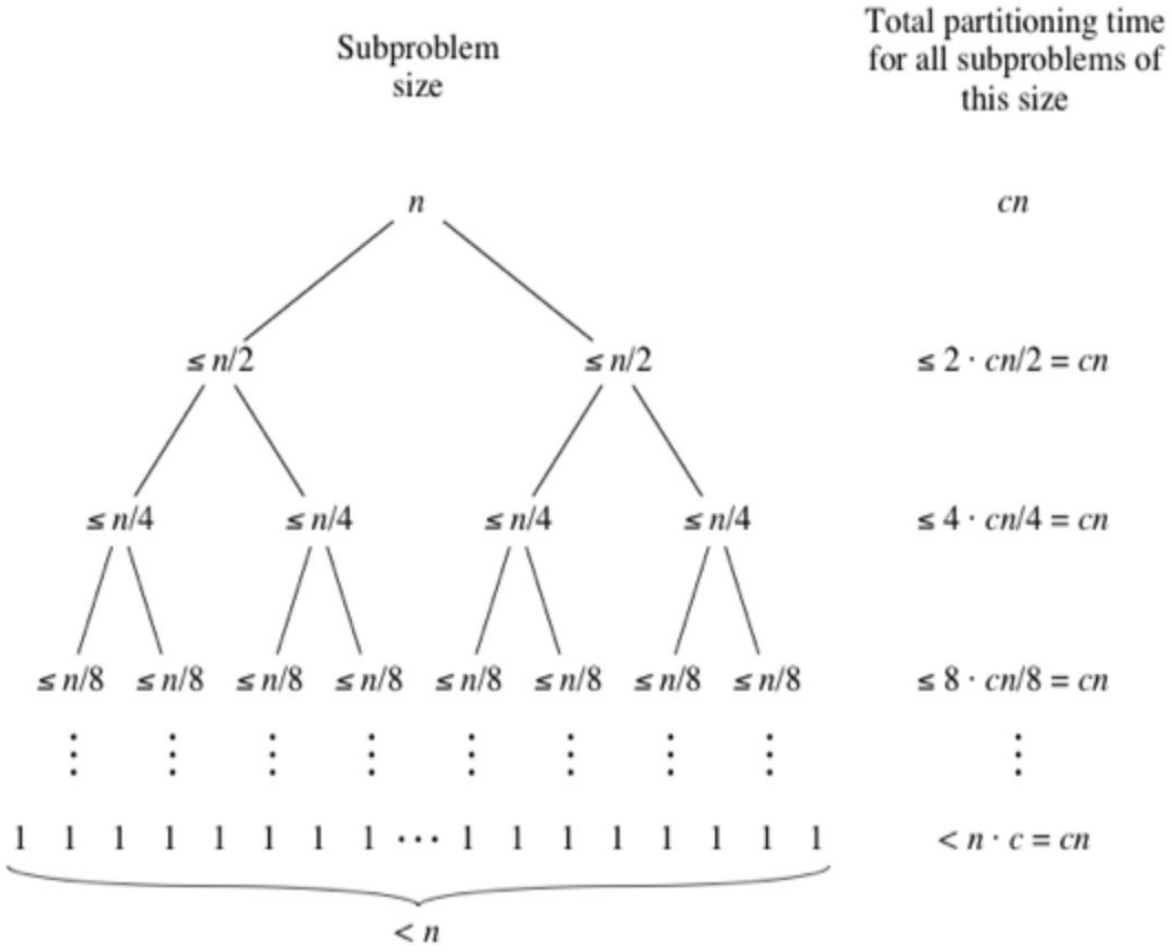


Time complexity of Quicksort

Best case run time occurs when the partitions are as evenly balanced as possible: their sizes either are equal or are within 1 of each other.

quicksort's best-case and average-case running time is the same as mergesort $\Theta(n \lg n)$

Balanced partitions



Pros of Quicksort

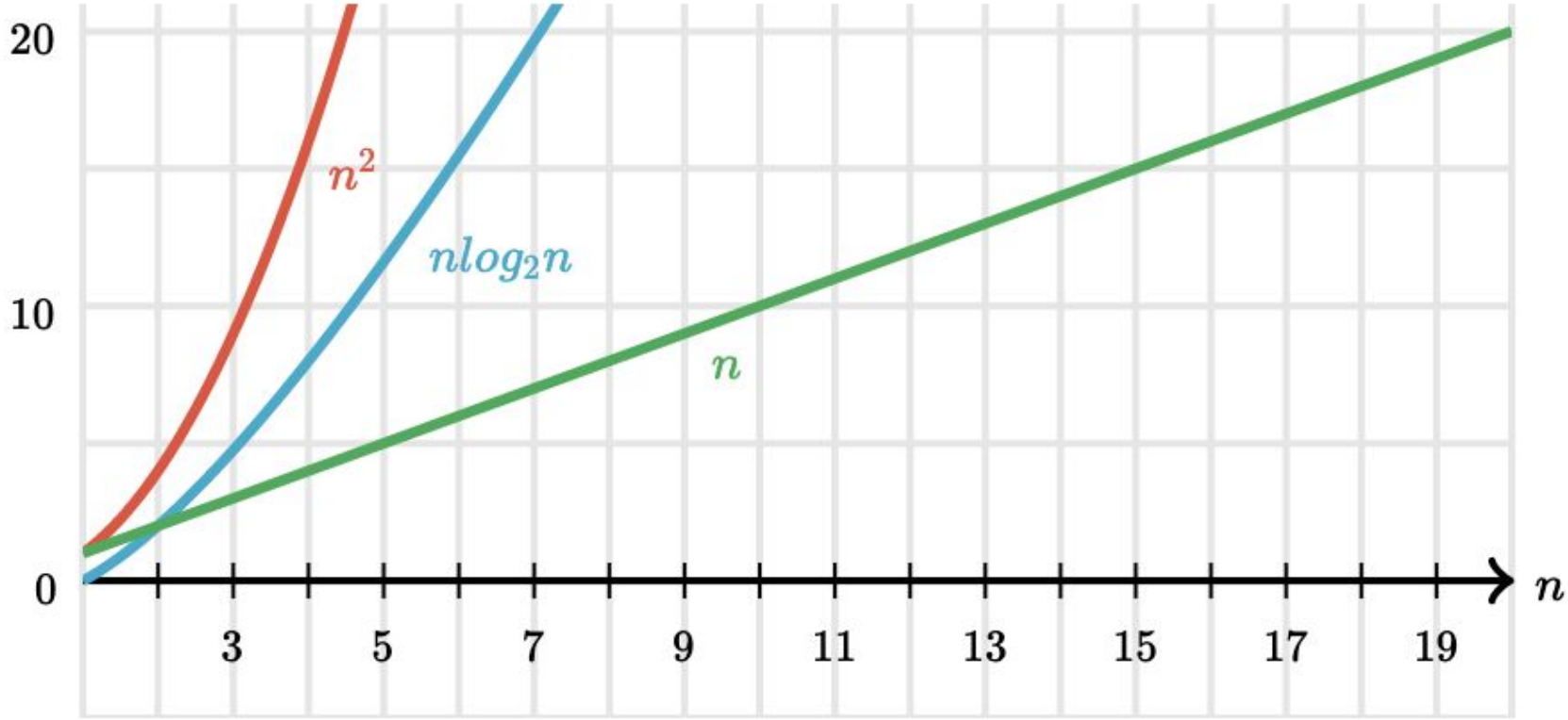
Quicksort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, mergesort is generally considered better when data is huge and stored in external storage.

Homework problems!

Please complete textbook problem 5.2: 5ab

Let's compare

Algorithm	Worst-case running time	Best-case running time	Average-case running time
Selection sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Insertion sort	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$


20
10
0
n^2
$nlog_2n$
n
3
5
7
9
11
13
15
17
19
n

Let’s break into groups and work on writing code

Choose either mergesort or quicksort (or both if you have time)