QuickSort

QuickSort

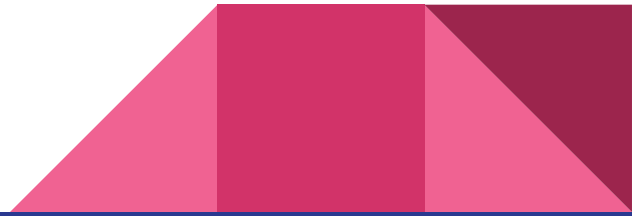
QuickSort works similarly to QuickSelect:

- It picks an element as a pivot
- It creates partitions based on the picked pivot
- QuickSelect returns the value of the Kth element while the goal of the QuickSort is to use the partitions logic to sort the array.

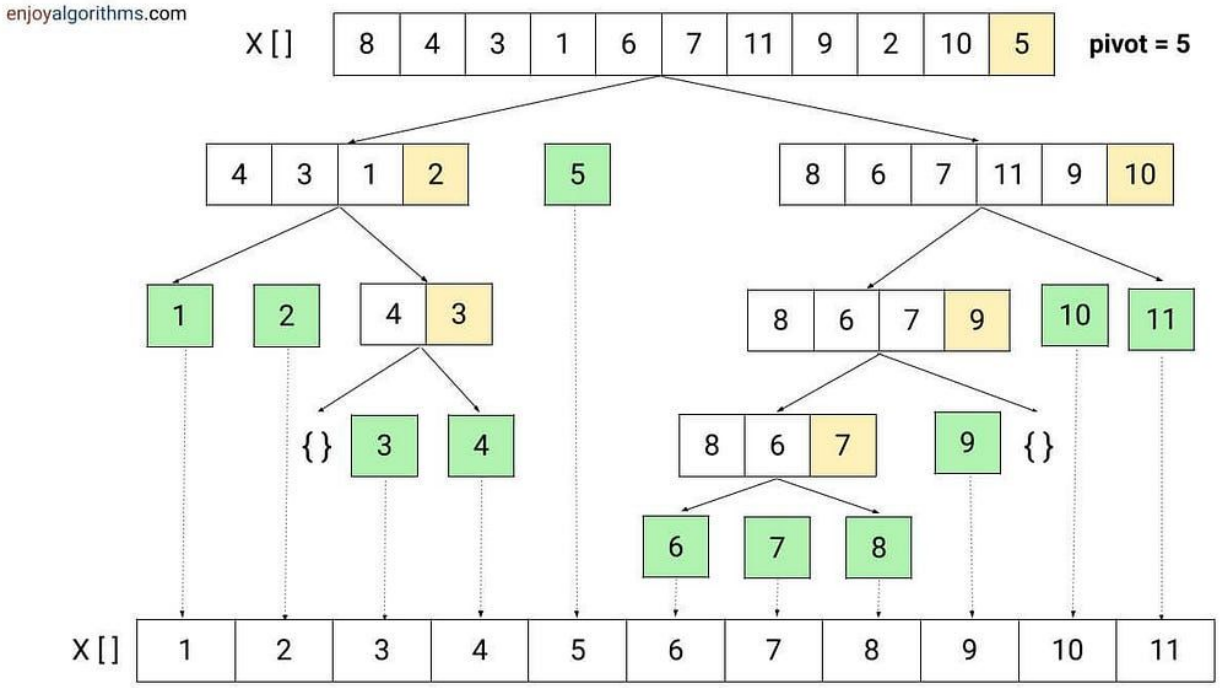
QuickSort versions

A pivot element for the QuickSort can be picked in different ways:

- Pick median as the pivot.
- **Pick a random element as a pivot.**
- Always pick the first element as a pivot.
- Always pick the last element as a pivot.



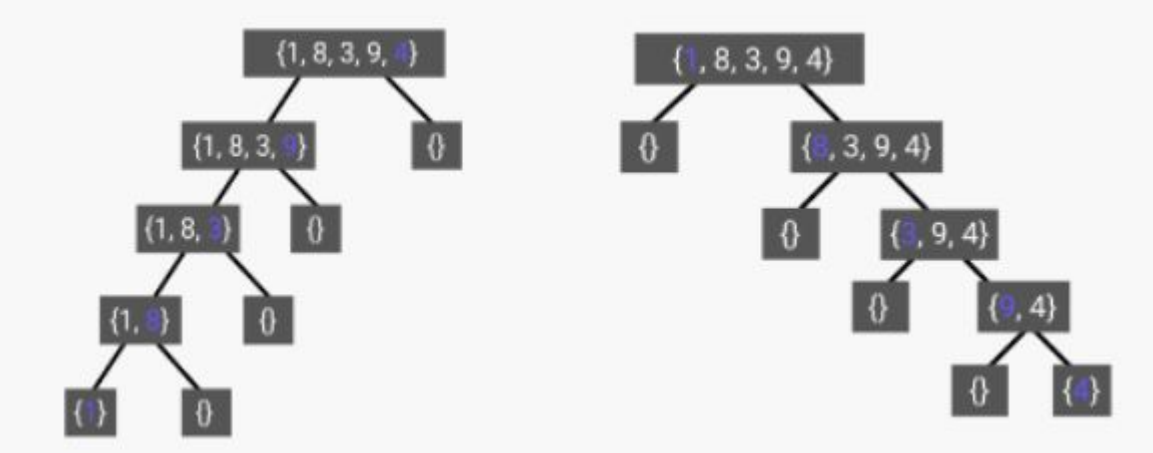
QuickSort



What does it happen after each call to partition(....)?

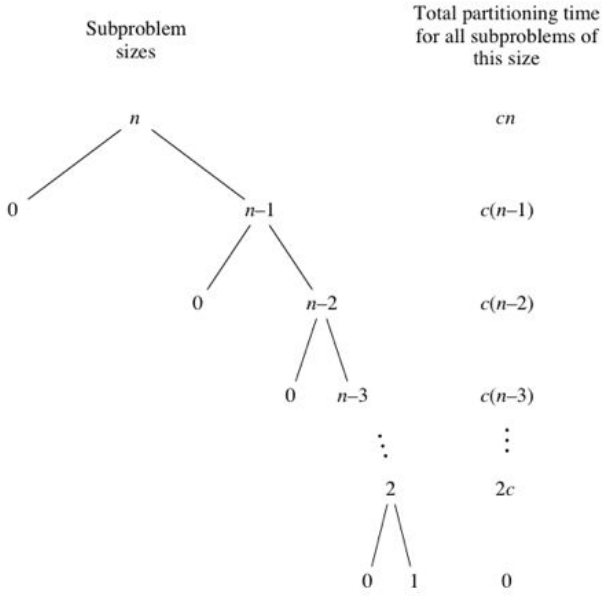
Time Complexity - Worst Case

The worst case occurs when the pivot element is either the greatest or the smallest element.



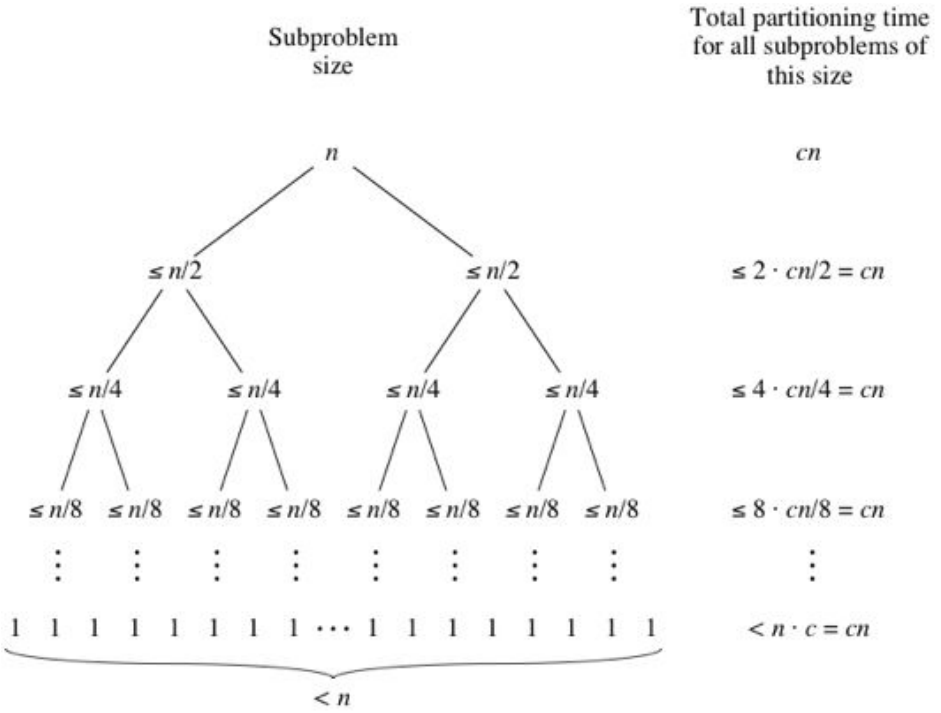
Time Complexity - Worst Case

The **worst-case** time complexity of quicksort is **$O(n^2)$**.



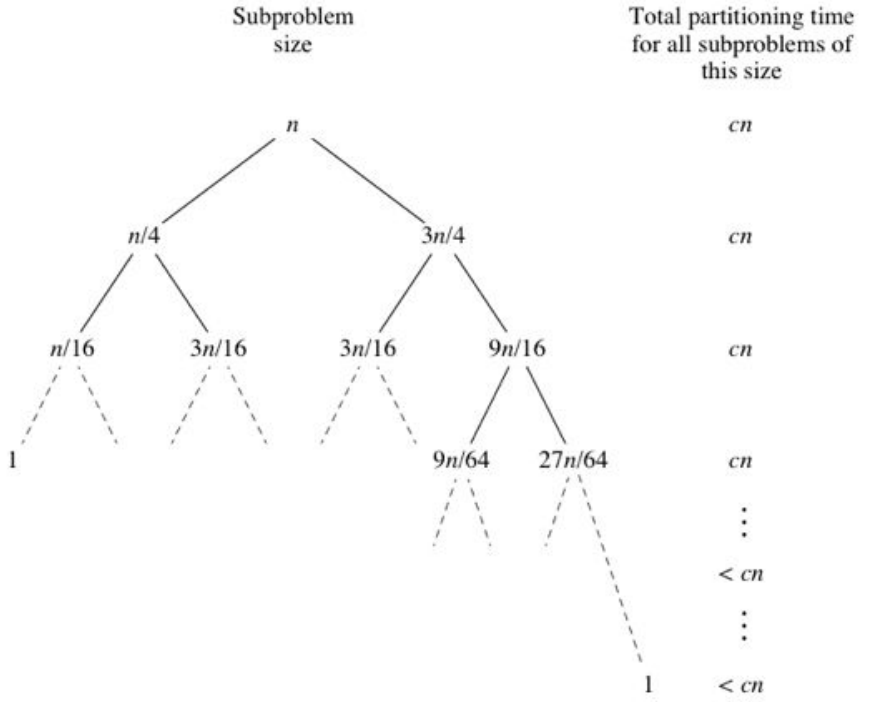
$$T(n) = cn + c(n-1) + c(n-2) + \ldots + 2c$$
$$= c(n + (n-1) + (n-2) + \ldots + 2)$$
$$= c((n+1)(n/2) - 1)$$

Time Complexity - Best Case



The tree levels represent the number of times we can repeatedly halve, starting at n, until we get the value 1, plus one.

Time Complexity - Average Case

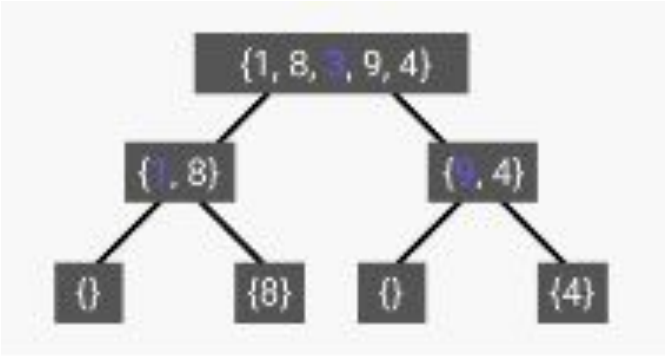


Time Complexity - Best/Avg Cases

Best case time complexity **O(n log(n))**.

The best-case occurs when the pivot element is the middle element or near to the middle element

In this case the recursion will look as shown in the diagram, as we can see the height of tree is logN and in each level we will be traversing to all the elements with total operations will be logN * N.



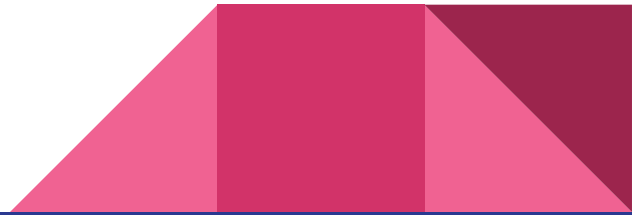
Average case time complexity **O(n log(n))**.

QuickSort: Performance issue

QuickSort exhibits poor performance for inputs that contain many repeated elements. The problem is visible when all the input elements are equal.

Then at each point in recursion, the left partition is empty (no input values are less than the pivot), and the right partition has only decreased by one element (the pivot is removed).

The algorithm takes quadratic time to sort an array of equal values.



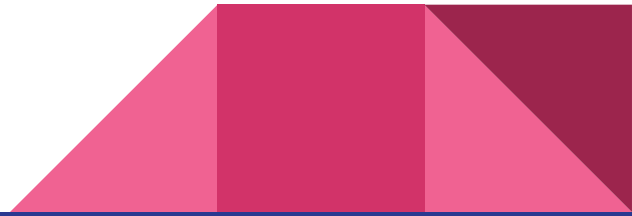
Quicksort using Dutch National Flag Algorithm

It Implements QuickSort efficiently for inputs containing many repeated elements.

We can use an alternative linear-time partition routine to solve this problem that separates the values into three groups:

- The values less than the pivot,
- The values equal to the pivot, and
- The values greater than the pivot.

The values equal to the pivot are already sorted, so only the less-than and greater-than partitions need to be recursively sorted.

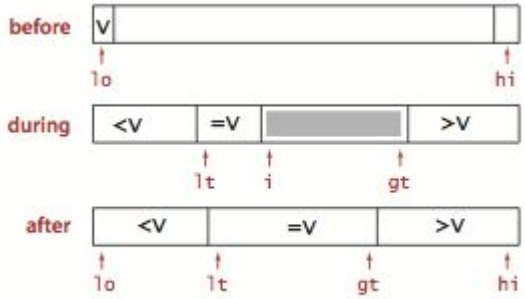


Quicksort using Dutch National Flag Algorithm

This method will not call QuickSort on the middle section which has the elements equals to pivot.

It must return the left and right index of the middle section.

The method must return an array with the the left and right indices. QuickSort calls will use those values.



3-way partitioning overview

Coding Time = Fun Time

QuickSort

```
public static void quickSort(int[] data, int start, int end) {

}
```

Dutch Partition is Optional

```
public static int[] partitionDutch(int[] data, int start, int end) {

}
```