Sorting & Divide and Conquer

ANCC, IITD

June 2021

Sorting

Formal Definition

Sorting is a technique which is used to permute an array A such that for a given (total order) relation $\mathbb{R}$ and any two indices i, j where $i < j$, $A[i] \leq_{\mathbb{R}} A[j]$.

Simpler Explanation

Sorting is used to rearrange the array such that all elements are arranged in an ordered fashion. The ordering can be a simple increasing/decreasing one or it can also be something relatively more complicated.

Sorting Algorithms

Overview

There are a lot of sorting algorithms, with varying time complexities ranging from $O(n \cdot log(n))$ to $O(n!)$. We will discuss in brief about two of the common and fast sorting algorithms.

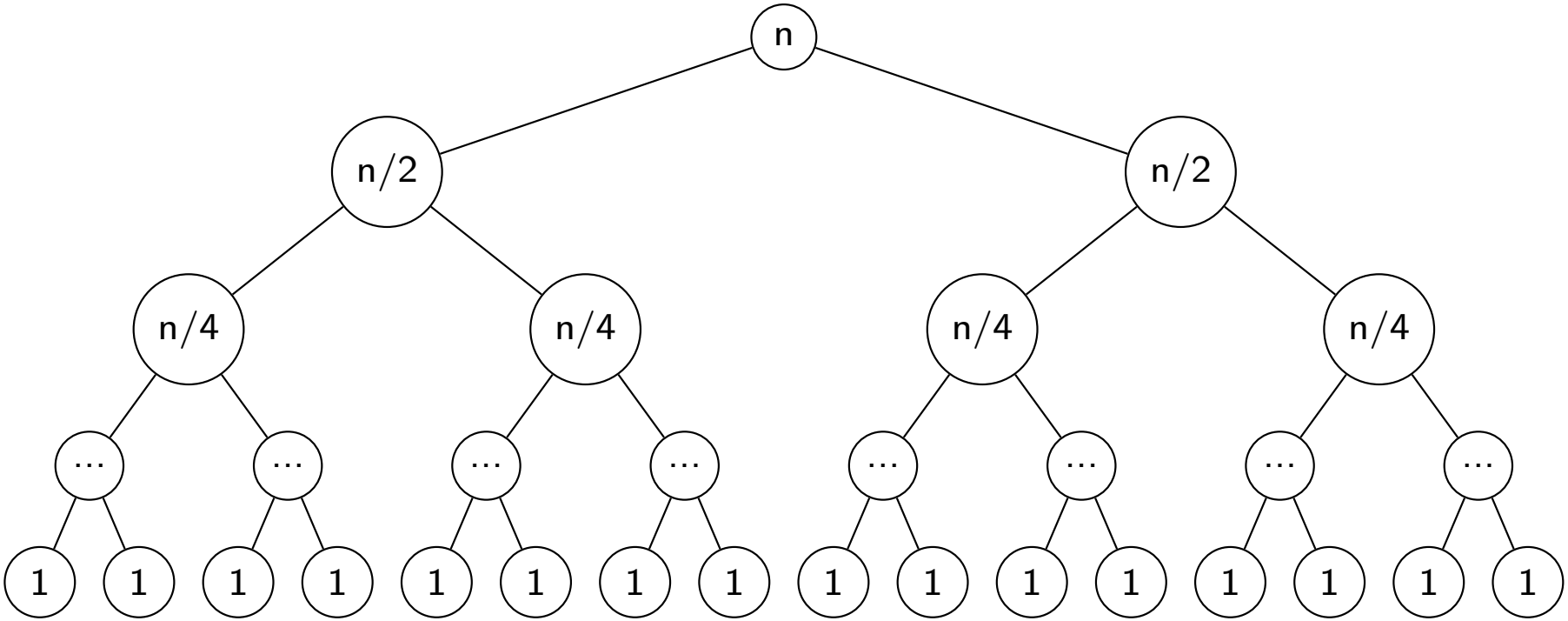
Merge Sort

This algorithm uses a divide and conquer strategy:

1. Split the array A into two (almost) equal halves - $B = A[l \ldots mid]$ and $C = A[mid + 1 \ldots r]$, where $mid = (l + r)/2$ (initially $l = 0$ and $r = n$).
2. Merge Sort these two arrays B and C.
3. Now merge these two arrays to get the final sorted array A.

Time complexity: Since the array is split into two halves at every steps, we will have $log(n)$ levels of the recursion and on each level we perform $O(n)$ operations. Hence to total complexity is $O(n \cdot log(n))$.

Merge Sort - Recursion Tree



Quick Sort

This algorithm also uses a different kind of divide and conquer strategy:

1. Choose any element, e of the array A as a pivot.
2. Partition the array into two arrays B and C such that all elements in B are $\leq e$ and all elements in C are $> e$.
3. Quick sort the two arrays B and C.

Time complexity: In the worst case, the partitions can divided unequally as $n-1$ and 1. Making the partition takes $O(n)$ and this would lead to a total time complexity of $O(n^2)$. However, in general, the quick sort algorithm has an average running time of $O(n \cdot log(n))$ since *on an average* the array is partitioned into two equal halves.

Divide and Conquer

Overview

Divide and conquer is a technique which is used to **divide** the given problem into subparts, solve the problem independently for those two halves and then combine (**conquer**) the obtained solutions for each subpart to find the final result.

Strategy

Recursive implementations are usually intuitive when thinking of a divide and conquer strategy since solving the subproblem is the same as solving another problem on a different input. The basic recursive algorithm would look like -

```
function divideAndConquer(problem):
  solutions = []
  for subProblem in problem:
    solution = divideAndConquer(subProblem)
    solutions.add(solution)
  conquer(solutions) // find the actual solution for the current problem
```