Quicksort And Liner Time Sorts - Recitation 6

Quicksort, Countingsort, Radixsort And Bucketsort.

November 19, 2022

Till now we have quantiffied the algorithm permformance against the worst case scenerio. And we saw that accorrding to that masure, In the comparisons model, one can not sorting in less then $\Theta\left(n \log n\right)$ time. In this recation we present a two new main concepts that, in certain cases, achive batter anlysies. The first one is the Excpection Complexitiy, By Letting the algorithm to behave undetrmistcly, we might obtain an algortihm that most of the time run compute the task fast. Yet we will not secussed get down the $\Theta\left(n \log n\right)$ lower bound, but we will back to use that concept in the pending of the course. The seconed concept is to ristrict ourselves to deal only in particular type of inputs. For example We will see that if we suppose that the a given array contains only integer in bounded domain then we can sort it in linear time.

0.1 Quicksort.

The quicksort algorithm is a good example for a **non-determistic** algorithm that has a worst-case running time of $\Theta\left(n^2\right)$. Yet its expected running time is $\Theta\left(n \log n\right)$. Namley fix an a array of n numbers, the ruunings of Quicksort over that array might be diffrent, each of them is an diffrent event in probability space, and the rununing time of the algorithm is a random variabile defiend over that space. Saying that the algorithm has worst space complexitiy of $\Theta(n^2)$ means that there exist event in which it runs $\Theta\left(n^2\right)$ time with non-zero probability. But parcticaly the interesting question is not the existance of such event but how likily that it happend. It turns out that expection of the runnning time is acctuly $\Theta\left(n \log n\right)$.

What is the exactly reason that happens? By giving up on the algoirthm behaveoir centertiy we are going to turn the task of eneginering bad input impossible.

Our study of quicksort is broken into four sections. Section 7.1 describes the algorithm and an important subroutine used by quicksort for partitioning. Because the behavior of quicksort is complex, we'll start with an intuitive discussion of its performance in Section 7.2 and analyze it precisely at the end of the chapter. Section 7.3 presents a randomized version of quicksort. When all elements are distinct,1 this randomized algorithm has a good expected running time and no particular input elicits its worst-case behavior. (See Problem 7-2 for the case in which elements may be equal.) Section 7.4 analyzes the randomized algorithm, showing that it runs in $\Theta(n^2)$ time in the worst case and, assuming distinct elements, in expected $O\left(n \log n\right)$ time.

randomized-partition(A, p, r)

$i \leftarrow$ random (p, r)
$A_r \leftrightarrow A_i$
return Partition (A, p, r)

randomized-quicksort (A, p, r)

if $p < r$ **then**
    $q \leftarrow$ randomized-partition (A, p, r)
    randomized-quicksort $(A, p, q-1)$
    randomized-quicksort $(A, q+1, r)$