Ruolin Li
31764160

CPSC 320: Divide and Conquer *

Invented by Tony Hoare in the late 1950's, the Quicksort algorithm was a breakthrough in sorting methods. Variants of Hoare's original algorithm are still a sorting method of choice today. Here you'll gain experience with the divide and conquer algorithmic design approach, as well as recurrence analysis, that led Hoare to this breakthrough, and see an application also to finding the median.

Here is a basic version of Quicksort, when the array $A[1..n]$ to be sorted has n distinct elements:

```
function QUICKSORT(A[1..n])      ▷ returns the sorted array A of n distinct numbers
    if n > 1 then
        Choose pivot element p = A[1]
        Let Lesser be an array of all elements from A less than p
        Let Greater be an array of all elements from A greater than p
        LesserSorted ← QuickSort(Lesser)
        GreaterSorted ← QuickSort(Greater)
        return the concatenation of LesserSorted, [p], and GreaterSorted
    else
        return A
```

1 Quicksort Runtime Analysis

1. Suppose that QuickSort happens to always select the $\lceil \frac{n}{4} \rceil$-th largest element as its pivot. Give a recurrence relation for the runtime of QuickSort.

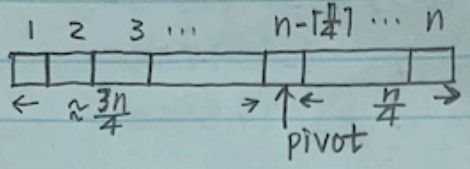
2. Using the recurrence, draw a recursion tree for QuickSort. Label each node by the number of elements in the array at that node's call (the root is labeled n) and the amount of time taken by that node but not its children. Also, label the total work (time) for each "level" of calls. Show the root at level 0 and the next two levels below the root, and also the node at the leftmost and rightmost branches of level i.

3. Find the following two quantities.

   (a) The number of levels in the tree down to the shallowest leaf. *Hint:* Is the shallowest leaf on the leftmost side of the tree, the rightmost side, or somewhere else? If you've already described the problem size of the leftmost and rightmost nodes at level i as a function of i, then set that equal to the problem size you expect at the leaves and solve for i.

   (b) The number of levels in the tree down to the deepest leaf.

Divide and Conquer

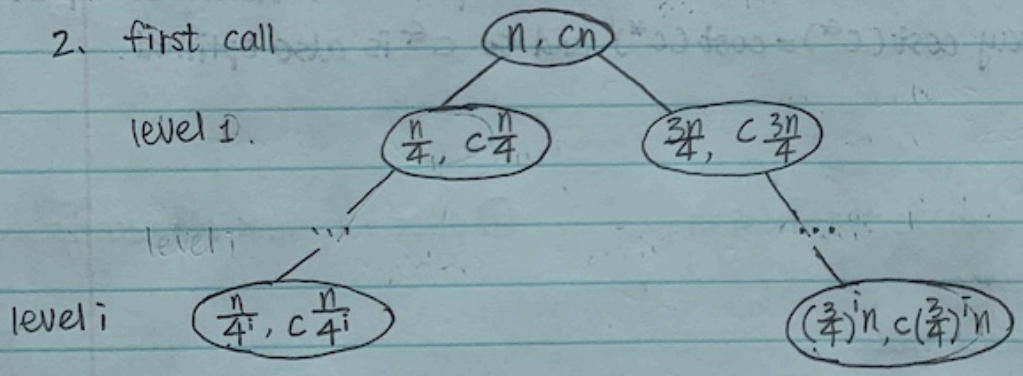
1. 

if $n>1$ then
- $\Theta(1)$ Choose pivot element $p=A[1]$
- $\Theta(n)$ { Let lesser; Let bigger
- Non-recursive part takes $\Theta(n)$ time
- $T(\dots)$ → Lessersorted; → Biggersorted
- $O(n)$ return the concatenation
- $\Theta(1)$ else return A — Base Case

Let $T(n)$ be the runtime on an array of size n.

$$T(n)=\begin{cases} c, & (n=1) \\ T(\lceil \frac{n}{4} \rceil -1)+T(\lfloor \frac{3n}{4} \rfloor)+cn & (n>1) \\ T(\frac{n}{4})+T(\frac{3n}{4})+cn & \end{cases}$$

2. first call



$$T(n) \le 2T(n/2)+cn \le 2k(n/2)\log_b(n/2)+cn$$
$$= 2k(n/2)[\log_2 n-1]+cn$$
$$= kn(\log_2 n-1)+cn$$
$$= kn\log_2 n-kn+cn$$

$\therefore T(n) \le kn\log_2 n$. → Is there a choice of k that will cause the expression to be bounded by $kn\log_2 n$?

→ Yes, choose k which is at least as large as c.

4. Use the work from the previous parts to find asymptotic upper and lower bounds for the solution of your recurrence.

5. Now, we'll relax our assumption that always selects the $\lceil \frac{n}{4} \rceil$-th largest element as its pivot. Instead, consider a weaker assumption that the rank of the pivot is always in the range between $\lceil \frac{n}{4} \rceil$ and $\lfloor \frac{3n}{4} \rfloor$. The *rank* of an element is k if the element is the kth largest in the array. What can you say about the runtime of Quicksort in this case?

6. Draw the recursion tree generated by `QuickSort([10, 3, 5, 18, 1000, 2, 100, 11, 14])`. Assume that QuickSort: (1) selects the first element as pivot and (2) maintains elements' relative order when producing `Lesser` and `Greater`.

Ruolin Li
31764160
2020/07/24

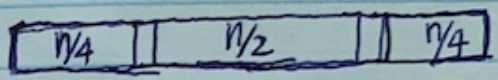
3. (a) The number of levels in the tree down to the shallowest leaf.
The $\frac{n}{4^i}$ branch will reach the base case fastest. So $\frac{n}{4^i} = 1$ $\quad i = \log_4 n$.

(b) The number of ~~leaf~~ levels in the tree down to the deepest leaf.
The $(\frac{3}{4})^i n$ branch reaches the base case slowest.
$i = \log_{\frac{3}{4}} n$.

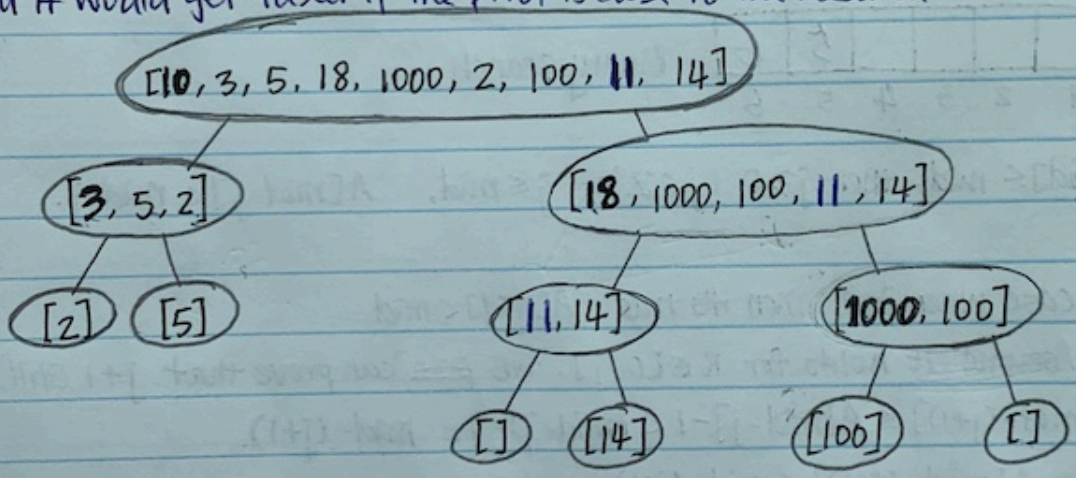
$\Omega(n)$ (lower bound)

4. Perform $O(n)$ work at each level. and in either cases there are $\Theta(\lg n)$ levels.
So total runtime: $O(n \lg n)$. $\quad \Omega(n \lg n)$ (lower bound) $\quad \rightarrow T(n) = \Theta(n \log n)$

5. The runtime remains the same.
But it would get faster if the pivot is close to the median.



6.



- An algorithm for Finding the Median

1.

2. (a) 18
(b) 3
(c) 5
(d) still keeps the same. 11 ended up on the right side. so everything that was larger than it before is still in the array. So still the 5th largest item.

2 An Algorithm for Finding the Median

Suppose that you want to find the median number in an array of length n. The algorithm below can be generalized to find the element of rank k, i.e., the kth largest element in an array, for any $1 \leq k \leq n$. (Note: the lower the rank, the larger the element.) The median is the element of rank $\lceil n/2 \rceil$. For example, the median in the array [10, 3, 5, 18, 1000, 2, 100, 11, 14] is the element of rank 5 (or $5th$ largest element), namely 11.

1. In your specific recursion tree above, mark the nodes in which the median (11) appears. (The first of these is the root.)

2. Look at the second recursive call you marked—the one below the root. 11 is **not** the median of the array in that recursive call!

   (a) In this array, what is the median?

   (b) In this array, what is the rank of the median?

   (c) In this array, what is the rank of 11?

   (d) How does the rank of 11 in this array relate to the rank of 11 in the original array (at the root)? Why does this relationship hold?

3. Look at the **third** recursive call you marked. What is the rank of 11 in this array? How does this relate to 11's rank in the second recursive call, and why?

4. If you're looking for the element of rank 24 (i.e., the $42nd$ largest element) in an array of 100 elements, and `Greater` has 41 elements, where is the element you're looking for?

5. How could you determine **before** making `QuickSort`'s recursive calls whether the element of rank k is the pivot or appears in `Lesser` or `Greater`?

6. Modify the `QuickSort` algorithm so that it finds the element of rank k. Just cross out or change parts of the code below as you see fit. Change the function's name! Add a parameter! Feel the power!

```
function QuickSort(A[1..n]) // returns the sorted array A of n distinct numbers
    if n > 1 then
        Choose pivot element p = A[1]
        Let Lesser be an array of all elements from A less than p
        Let Greater be an array of all elements from A greater than p
        Let LesserSorted = QuickSort(Lesser)
        Let GreaterSorted = QuickSort(Greater)
        return the concatenation of LesserSorted, [p], and GreaterSorted
    else
        return A
```

7. Once again, suppose that the rank of the pivot in your median-finding algorithm on a problem of size n is always in the range between $\lceil \frac{n}{4} \rceil$ and $\lfloor \frac{3n}{4} \rfloor$. Draw the recursion tree that corresponds to the worst-case running time of the algorithm, and give a tight big-O bound in the algorithm's running time. Also, provide an asymptotic lower bound on the algorithm's running time.