Recitation- Week 4

Michael Levet

December 4, 2020

1 Quickselect

The Quickselect algorithm is an adaptation of Quicksort, which we use to find the kth smallest element in an array A of length n. The initial steps of Quickselect are identical to Quicksort in that Quickselect partitions the input array around a pivot element. So we have left and right subarrays after partitioning the original array A. Suppose the sub-arrays have sizes:

- $\text{len}(\textsf{left}) = m$.
- $\text{len}(\textsf{right}) = n - m - 1$. [Note that the -1 term comes from accounting for the pivot element. So $n = \text{len}(\textsf{left}) + \text{len}(\textsf{right}) + 1 = m + (n - m - 1) + 1$.]

The trick now comes down to determining which piece to look at. We note that A is partitioned as follows:

$$A = [\textsf{left}|\textsf{pivot}|\textsf{right}].$$

Observe the following.

- If $\text{len}(\textsf{left}) \geq k$, then we know the kth smallest element of A must be in left. In particular, the kth smallest element of A is also the kth smallest element of left. So we recurse, using Quickselect to search for the kth smallest element in left.
- If $k = \text{len}(\textsf{left}) + 1$, then the kth smallest element in A is pivot, so we return pivot. The algorithm terminates.
- If $k > \text{len}(\textsf{left}) + 1$, then the kth smallest element in A must be in right. In particular, the kth smallest element of A is the $k - 1 - \text{len}(\textsf{left})$ smallest element of right [This is because the $\text{len}(\textsf{left}) + 1$ smallest elements of A are the pivot and elements of left). So we recurse, using Quickselect to search for the $k - 1 - \textsf{len}(A)$ smallest element in right.

We collect these observations into pseudocode.

```
Quickselect(A[1, ..., n], k):
     pivotIndex := partition(A)
     left := A[1, ..., pivotIndex - 1]
     right := A[pivotIndex + 1, ..., n]

     if len(left) + 1 ==  k:
        return A[pivotIndex]

     else if k <= len(left):
        return Quickselect(left, k)

     return Quickselect(right, k-1-len(left))
```

Recall the Partition algorithm from the Lecture Notes.

```
Partition(A[1, ..., n], start, end):
   p := A[end]
   i := start
   for j = start; j < end; j = j + 1:
        if A[j] <= p:
           swap(A, i, j)
           i = i + 1
    swap(A, i, end)
```

We now consider an example.

Example 1. Suppose we want to find the 3rd smallest element of:

$$A = [3, 7, 2, 9, 1, 6, 8, 4].$$

We proceed as follows.

- Invoke Quickselect$(A, 3)$. Using the Partition algorithm from the Lecture Notes, we select the last element as our pivot. So A is partitioned as follows:

  $$A = [3, 2, 1|4|7, 6, 8, 9],$$

  where left $= [3, 2, 1]$ and right $= [7, 6, 8, 9]$. Now as $k = 3 = \text{len}(\text{left})$, we recurse: Quickselect$(\text{left}, 3)$.

- We note that $A = [3, 2, 1]$. Using the Partition algorihtm from the Lecture Notes, we select the last element as our pivot. So A is partitioned as follows:

  $$A = [|1|2, 3]$$

  So pivot $= 1$, left $= []$, and right $= [2, 3]$. Now as $k = 3 > 1 + \text{len}(\text{left}) = 1 + 0 = 1$, we have that the 3rd smallest element of A must be the 2nd smallest element of right. So we recurse, invoking Quickselect$(\text{right}, 2)$.

- We note that $A = [2, 3]$. Using the Partition algorithm from the Lecture Notes, we select the last element as our pivot. So A is partitioned as follows:

  $$A = [2|3||],$$

  where left $= [2]$, pivot $= 3$, and pivot $= 1$. We note that $k = 2 = \text{len}(\text{left}) + 1$. So we return pivot $= 3$.

So for the original array $A = [3, 7, 2, 9, 1, 6, 8, 4]$, the algorithm found 3, which was the third smallest element in the array.

2 Questions

Problem 1. Considerthe initial input array $A = [3, 7, 2, 9, 1, 6, 8, 4]$ from Example 1. In the first call to Quickselect, the algorithm partitioned A as follows:

$$A = [3, 2, 1|4|7, 6, 8, 9],$$

where left $= [3, 2, 1]$ and right $= [7, 6, 8, 9]$.

- Suppose we wanted to find the 4th smallest element instead of the 3rd smallest element. What would the algorithm's next step be after this partitioning?

- Suppose we wanted to find the 6th smallest element instead of the 3rd smallest element. What would the algorithm's next step be after this partitioning?

Problem 2. The Quickselect algorithm has a worst case runtime complexity of $O(n^2)$. What is a worst-case input for Quickselect? [**Hint:** Think about the worst-case runtime for Quicksort.]

Problem 3. The *best-case* runtime complexity of Quickselect is given by the recurrence:

$$T(n) = \begin{cases} T(n/2) + \Theta(n) & : n > 1, \\ \Theta(1) & : \text{otherwise.} \end{cases}$$

Do the following.

- Solve the recurrence. Your final answer should be a Big-Theta bound. That is, find a suitable function $f(n)$ such that $T(n) = \Theta(f(n))$.
- Clearly explain when we acheive the best-case runtime for Quickselect.