Luis Gascon, Ethan Webb, Femi Dosumu
COSC 336
April 22, 2023

Assignment 6

Exercise 1. Recall the Partition subroutine employed by QuickSort. You are told that the following array has been partitioned around some pivot element:

3	1	2	4	5	8	7	6	9

Which of the elements could have been the pivot element? (List all that apply; there could be more than one possibility.)

Answer: 4, 5, and 9 are all possible pivots.

Exercise 2. Let α be some constant, independent of the input array length n, strictly between 0 and 1/2. What is the probability that, with a randomly chosen pivot element, the Partition function produces a split in which the size of both the resulting subproblems is at least $\alpha \cdot n$. Choose the answer from the following list and justify your answer.

(a) α

(b) $1 - \alpha$

(c) $1 - 2\alpha$

(d) $2 - 2\alpha$

Answer: (c) 1-2a
The pivot must be greater or equal to a·n (left side of the pivot), meaning the right side of the pivot must be less than or equal to n - a·n. We then just compute this probability and do some simple cancellations.

$$a \cdot n \leq pivot \leq n - a \cdot n$$

$$P(a \cdot n \leq pivot \leq n - a \cdot n)$$

$$P(A) = \frac{n - 2a \cdot n}{n}$$

$$P(A) = 1 - 2a$$

We augment the node class by adding an integer property of size, which counts the number of descendant a node has, plus itself. To increment size, we modify the recursive insert helper function to increment `root.size` by 1. We utilize the size property by using it as a number to be compared with k. We first initialize a variable `leftSize`, which stores the size of the current node's left child.
We have 3 cases that determines whether our algorithm recurses at a certain direction or return and we have found the solution.

- Case 1: $k = leftSize + 1$
  - This is the case when we've found the k-th smallest element and we just return the value of the current node
- Case 2: $k \leq leftSize$
  - This is the case when the solution is on the left subtree, so we recurse down the left child node of the root until the first case is met.
- Case 3: $k > leftSize$
  - This is the case when the kth smallest element can be found in the right subtree and the algorithm recurses on the right. The recursive function call has the k being subtracted by `leftSize` -1 since we've already accounted for the nodes on the left subtree, if there is one.

The next page shows the algorithm and the solutions obtained

```java
  int select(Node root, int k) {
    // Find the median kth order statistic
    k = (k / 2) + 1;

    // Input validation
    if (k > root.size)
        return -1;

    // Prevents NullPointerExceptions
    int leftSize = root.left != null ? root.left.size : 0;

    if (k == leftSize + 1)
        return root.item;
     else if (k <= leftSize)
        return select(root.left, k);
    return select(root.right, k - leftSize - 1);
}
```

Solutions obtained

Input	k-th smallest element
{7, 10, 3, 13, 13}	10
input-6.1.txt	501
input-6.2.txt	5019