CPSC 320: Prune and Search*

In this worksheet, we investigate an algorithm design technique called *Prune and Search*. It is used to search for an element (or a group of elements) with a specific property, and it is closely related to Divide and Conquer. The main difference between the two is that we only recurse on one subproblem, instead of all of them.

1 An Algorithm for Finding the Median

Suppose that you want to find the *median* value in an array (not necessarily sorted) of length n. The algorithms we will discuss can, in fact, be used to find the element of rank k, i.e., the kth smallest element in an array, for any $1 \leq k \leq n$. (Note: the larger 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 $5th$ smallest element), namely 11.

1. We will start by thinking about how we would use Quicksort to find the median of an array. Draw the recursion tree generated by the call `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`.

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

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. Now, 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?

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

6. 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`?

7. 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
```

8. 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.