CPSC 320: Prune and Search Solutions[*]

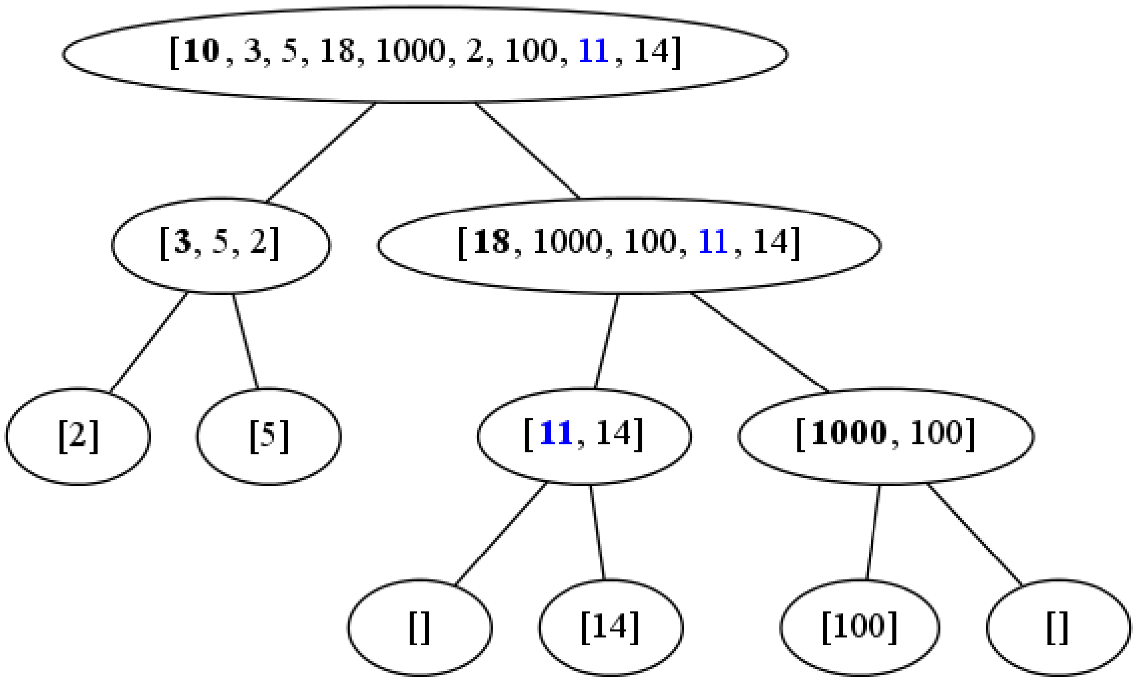
In this worksheet, we investigate an algorithm design technique called *Prune and Search*. It is used to search for an element (or a group of elements) with a specific property, and it is closely related to Divide and Conquer. The main difference between the two is that we only recurse on one subproblem,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

   SOLUTION: Here it is with each node containing the array passed into that node's call, the pivot in **bold**, and the number 11 in blue.



[*]Copyright Notice: UBC retains the rights to this document. You may not distribute this document without permission.

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

   SOLUTION: These are the ones with the blue 11 in them above.

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?

   SOLUTION: Note that we're looking at the node containing $[\mathbf{11}, 14]$. This is the Lesser array of its parent, and the rank of 11 is 1. The rank did not change because to obtain the Lesser array from its parent array, only elements larger than 11 were removed, namely the pivot and the two elements in Greater.

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!

   SOLUTION:

   (a) In this array, what is the median? **18**

   (b) In this array, what is the rank of the median? The element 18 has rank 3, i.e., 18 is the 3rd smallest element.

   (c) In this array, what is the rank of 11? The element 11 has rank 1; it is the smallest element.

   (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?
   Note that we're looking at the node containing $[\mathbf{18}, 1000, 100, 11, 14]$. The rank of 11 has changed: it is 1 here, but it was 5 at the root. Why? 11 ended up on the right side (in `Greater`). So, everything that was smaller than 11 in the original array (at the root) is no longer in the array $[\mathbf{18}, 1000, 100, 11, 14]$. We discarded four elements (3, 5, 2 and the median 10) from the array at the root of the tree, so 11's rank has gone down by four.

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?

   SOLUTION: If `Lesser` has 41 elements, then there are 41 elements smaller than the pivot. That makes the pivot the $42nd$ smallest element. In other words, if the size of `Lesser` is $k-1$, then the pivot has rank k. (This is what happened—with much smaller numbers—in the node where 11 is also the pivot.)

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`?

   SOLUTION: Putting together what we saw above:

   - If $|\text{Lesser}|$ is equal to $k-1$, then the k-th smallest element is the pivot.
   - If $|\text{Lesser}|$ is larger than $k-1$, then the pivot is larger than the k-th smallest element, which puts it in the `Lesser` group (and the left recursive call).
   - If `|Lesser|` is smaller than $k-1$, then the k-th smallest element is in `Greater` (and the right recursive call).

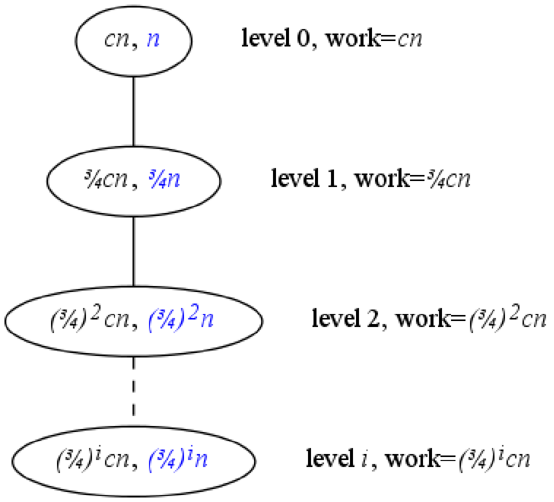
7. Modify the `QuickSort` algorithm so that it finds the element of rank k. Just cross out or change parts of the code below as you see fit. Change the function's name! Add a parameter! Feel the power!

SOLUTION: The key difference is that we no longer need the recursive calls on both sides, only on the side with the element we seek.

```
function QuickSelect($A[1..n]$, $k$) // returns the element of rank $k$ in an array $A$ of $n$ distinct numbers, where $1 \leq k \leq n$
    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 $p$
        if |Lesser| = k - 1 then
            return $p$
        else
            if |Lesser| > k - 1 then // all smaller elts are in Lesser; $k$ is unchanged
                return QuickSelect(Lesser, $k$)
            else// |Lesser| $< k - 1$
                // subtract from k the number of smaller elts removed
                return QuickSelect(Greater, $k-$ |Lesser| $-1$)
    else
        return $A[1]$
```

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.

SOLUTION: Here's the recursion tree (well, stick):



cn, n
level 0, work=cn
$\frac{3}{4}cn$, $\frac{3}{4}n$
level 1, work=$\frac{3}{4}cn$
$(\frac{3}{4})^2 cn$, $(\frac{3}{4})^2 n$
level 2, work=$(\frac{3}{4})^2 cn$
$(\frac{3}{4})^i cn$, $(\frac{3}{4})^i n$
level i, work=$(\frac{3}{4})^i cn$

Summing the work at each level, we get: $\sum_{i=0}^{?}(\frac{3}{4})^i cn = cn\sum_{i=0}^{?}(\frac{3}{4})^i$. We've left the top of that sum as ? because it turns out not to be critical. This is a *converging* sum. If we let the sum go to infinity (which is fine for an upper-bound, since we're not making the sum any smaller by adding positive terms!), it still approaches a constant: $\frac{1}{1-\frac{3}{4}} = \frac{1}{\frac{1}{4}} = 4$. So the whole quantity approaches $4cn \in O(n)$.

Unlike the Quicksort algorithm, there's no need to analyze recursion tree structure to conclude that the algorithm takes $\Omega(n)$ time. Since the algorithm partitions the n elements of A before any recursive call, it must take $\Omega(n)$ time, and thus $\Theta(n)$ time.

In case you're curious, here's one way to analyse the above sum. Let $S = \sum_{i=0}^{\infty}(\frac{3}{4})^i$. Then:

$$\begin{aligned}
S &= \sum_{i=0}^{\infty}(\frac{3}{4})^i \\
&= 1 + \sum_{i=1}^{\infty}(\frac{3}{4})^i \\
&= 1 + \sum_{i=0}^{\infty}(\frac{3}{4})^{i+1} \\
&= 1 + \frac{3}{4}\sum_{i=0}^{\infty}(\frac{3}{4})^i \\
&= 1 + \frac{3}{4}S
\end{aligned}$$

Solving $S = 1 + \frac{3}{4}S$ for S, we get $\frac{1}{4}S = 1$, and $S = 4$.

While recursion trees provide a reliable way to solve pretty much all recurrences that we'll see in this class, it's good to know about alternative methods too. One popular method is the Master theorem, provided in a separate handout. Again, if we assume that we "get lucky" with the choice of pivot at every recursive call, so the size of the subproblem is at most $3n/4$, we can express the QuickSelect runtime using the following recurrence (ignoring floors and ceilings):

$$T_{QS}(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } n = 1 \\ T_{QS}(3n/4) + cn & \text{otherwise} \end{cases}$$

This has the form of the Master theorem, where $a = 1$, $b = 4/3$, and $f(n) = cn$. Since $\log_b a = \log_{4/3} 1 = 0$ (the log of 1 is 0, no matter what the base is), we see that $f(n) = cn = \Omega(n^{(\log_b a + \epsilon)})$, where we can choose ϵ to be any positive number that is at most 1. So case 3 of the Master theorem applies, and we can conclude that $T_{QS}(n) = \Theta(n)$.

2 Deterministic Select

In the previous section, we developed a $\Theta(n)$ time algorithm to find the kth smallest element of an unsorted array, assuming that the rank of the pivot in our algorithm was always in the range between $\lceil \frac{n}{4} \rceil$ and $\lfloor \frac{3n}{4} \rfloor$, on a problem of size n. This unreasonable assumption will be relaxed in the bonus question at the end of this worksheet, and we will ask you to prove that if we choose the pivot at random then the *expected* (think, average) running time of the algorithm is in $\Theta(n)$. Here, we will develop an algorithm for this problem whose *worst-case* running time is in $\Theta(n)$.

1. Assuming you had a somewhat powerful magic wand you could use when comes the time to choose a pivot, what would you use that wand for? Note that you can't force the pivot to always be the element you're searching for (your magic wand is not *that* powerful).

   SOLUTION: We would use the wand to ensure that the pivot we get is not too close to either the smallest or the largest element of the array (maybe not exactly between those whose rank are $\lceil \frac{n}{4} \rceil$ and $\lfloor \frac{3n}{4} \rfloor$, but not too far from that).

2. Since magic wands don't exist, we will need to choose our pivot the old fashioned way (through computations). Suppose you are given an array containing the values

   19 35 44 13 42 23 21 62 27 24 40 53 31 16 48 25 17 13 9 26 18 50 21 30 67 17 71 87

   Group them into groups of 5 (from left to right), and circle the median of each group.

   SOLUTION:

13	19	$\boxed{35}$	42	44
21	23	$\boxed{24}$	27	62
16	31	$\boxed{40}$	48	53
9	13	$\boxed{17}$	25	26
18	21	$\boxed{30}$	50	67
17	71	87		

3. What is the median of the circled elements?

   SOLUTION: 30

4. How many of the elements are smaller than that median? How many are larger?

   SOLUTION: 15 elements are smaller, 12 are larger.

5. Now let us generalize the example from steps 2 through 4. Suppose we divide the elements into groups of 5 (unlike our previous example, up to 4 ungrouped elements may remain), find the median of each group by running insertion sort and picking the 3rd smallest (with only 5 elements, there is need to be fancy), and then compute the median m of the group medians (recursively this time, since we have many group medians). How many groups of 5 elements will we get?

   SOLUTION: We will get $\lfloor n/5 \rfloor$ groups.

6. How many elements of the array are *guaranteed* to be smaller than m?

   SOLUTION: The value m is the median of $\lfloor n/5 \rfloor$ group medians, and so it is the $\lceil \lfloor n/5 \rfloor /2 \rceil$ smallest of them. There are $\lceil \lfloor n/5 \rfloor /2 \rceil - 1$ groups with medians smaller than m, each of which contains 3 elements smaller than m (the smallest two, and their median). The group that contains m also has two elements smaller than it. So there are are least $3(\lceil \lfloor n/5 \rfloor /2 \rceil - 1) + 2$ elements smaller than m.

7. How many elements of the array are *guaranteed* to be larger than m?

   SOLUTION: The value m is the median of $\lfloor n/5 \rfloor$ group medians, and so it is the $\lceil \lfloor n/5 \rfloor /2 \rceil$ smallest of them. There are $\lfloor n/5 \rfloor - (\lceil \lfloor n/5 \rfloor /2 \rceil) = \lfloor \lfloor n/5 \rfloor /2 \rfloor$ groups with medians larger than m, each of which contains 3 elements larger than m (the largest two, and their median). The group that contains m also has two elements larger than it. So there are are least $3(\lfloor \lfloor n/5 \rfloor /2 \rfloor) + 2$ elements larger than m.

8. Consider once again the function `QuickSelect` from the last worksheet, with a name change and a different way of choosing p:

```
function DeterministicSelect($A[1..n]$, $k$) // returns the element of rank $k$ in an array $A$ of $n$ distinct numbers, where $1 \le k \le n$
    if $n \ge 5$ then
        Choose pivot element $p$ using the procedure described above
        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 $p$
        if |Lesser| = k - 1 then
            return $p$
        else
            if |Lesser| > k - 1 then // all smaller elts are in Lesser; $k$ is unchanged
                return DeterministicSelect(Lesser, $k$)
            else// |Lesser| $< k - 1$
                // subtract from k the number of smaller elts removed
                return DeterministicSelect(Greater, $k-$ |Lesser| $-1$)
    else
        sort $A$ and return $A[k-1]$
```

   What is the *largest* possible size of one of the `Lesser` or `Greater` lists?

   SOLUTION: The `Lesser` list is guaranteed not to contain m or the $3(\lfloor \lfloor n/5 \rfloor /2 \rfloor) + 2$ elements that are larger than it. So there are $3(\lfloor \lfloor n/5 \rfloor /2 \rfloor) + 2 + 1 = 3(\lfloor \lfloor n/5 \rfloor /2 \rfloor + 1)$ elements that it does not contain.

   The `Greater` list is guaranteed not to contain m or the $3(\lceil \lfloor n/5 \rfloor /2 \rceil - 1) + 2$ elements smaller than it. So there are $3(\lceil \lfloor n/5 \rfloor /2 \rceil - 1) + 2 + 1 = 3(\lceil \lfloor n/5 \rfloor /2 \rceil - 1) + 3 = 3\lceil \lfloor n/5 \rfloor /2 \rceil$ elements that it does not contain.

   The smallest of these two values is $3\lceil \lfloor n/5 \rfloor /2 \rceil$, and so in the worst case the largest one of `Lesser` and `Greater` contains at most

$$n - 3\left\lceil \frac{\lfloor n/5 \rfloor}{2} \right\rceil \le n - 3\frac{\lfloor n/5 \rfloor}{2} \le n - 3\frac{(n-4)}{10} = n - \frac{3n}{10} + \frac{12}{10} = \frac{7n}{10} + 1.2 \le \frac{7n}{10} + 2$$

   elements.

9. Derive a recurrence relation for the worst-case running time $T(n)$ of algorithm `DeterministicSelect` running on an array with n elements.

   SOLUTION: Splitting the array into groups of 5 and finding the median of each group takes $\Theta(n)$ time (we perform $\lfloor n/5 \rfloor$ constant time work), as does splitting the array into `Lesser` and `Greater` once we have found the median of the group medians. All other steps take $\Theta(1)$ time, except for the recursive call to find the median of the group medians ($T(\lfloor n/5 \rfloor)$) and the final recursive call on either `Lesser` or `Greater` (at most $T(7n/10 + 2)$). Therefore the worst-case running time of `DeterministicSelect` is described by the recurrence relation

$$T(n) \le \begin{cases} T(\lfloor n/5 \rfloor) + T(7n/10+2) + cn & \text{if } n \ge 5. \\ \Theta(1) & \text{if } n \le 4. \end{cases}$$

10. Prove that the function $T(n)$ described by your recurrence relation is in $\Theta(n)$.

SOLUTION: By drawing the recursion tree, we see that the amount of work done by row i is approximately at most $(9/10)^i cn$ (it's only approximately that because the +2 terms). This is almost a decreasing geometric series, and hence we can guess that $T(n) \in O(n)$. The work done at the root is cn, and so $T(n) \in \Omega(n)$ as well. Therefore $T(n) \in \Theta(n)$.

It is possible to prove the upper bound formally using mathematical induction. We first need a lemma:

Lemma 1 *If $n \ge 25$, then for every $c \in \mathbf{R}^+$, there exists $d \in \mathbf{R}^+$ for which $cn + 2d \le dn/10$.*

Proof: Consider an unspecified positive integer c, and choose $d \ge 50c$. Then $c \le d/50$ and so

$$\begin{aligned} cn + 2d &\le (d/50)n + 2d \\ &\le (dn/10)(1/5 + 20/n) \end{aligned}$$

Since $n \ge 25$, we know that $20/n \le 4/5$, and so $cn + 2d \le dn/10$. QED

Now we prove that the function $T(n)$ described by the recurrence relation

$$T(n) \le \begin{cases} T(\lfloor n/5 \rfloor) + T(7n/10+2) + cn & \text{if } n \ge 5 \\ \Theta(1) & \text{if } n \le 4 \end{cases}$$

is in $\Theta(n)$.

Clearly $T(n) \ge cn$, and so $T(n) \in \Omega(n)$. Now let us prove that $T(n) \le dn$ as long as n is large enough. We use the strong form of induction, starting with the induction step. Consider an unspecified value of $n \ge 25$. Assuming that the theorem is true for $\lfloor n/5 \rfloor$ and $7n/10 + 2$, we have

$$\begin{aligned} T(n) &\le T(\lfloor n/5 \rfloor) + T(7n/10+2) + cn \\ &\le d\lfloor n/5 \rfloor + d(7n/10+2) + cn \\ &\le dn/5 + 7dn/10 + 2d + cn \\ &\le 9dn/10 + (cn + 2d) \end{aligned}$$

Now, we know that as long as we pick $d \ge 50c$, $cn + 2d \le dn/10$. Hence, as long as $d \ge 50c$, $T(n) \le 9dn/10 + dn/10 = n$, as required.

It remains to consider the base case. The induction step works for $n \ge 25$, and assumes the theorem holds for $\lfloor n/5 \rfloor$, which means that we need to have $n_0 = 5$, and to treat $n = 5, 6, 7, \ldots, 24$ as base cases. Let us suppose that the values of d that work for them are $d_5, d_6, \ldots, d_{24}$. Then we can choose $d^* = \max\{d_5, d_6, \ldots, d_{24}, 50c\}$, and then for every $n \ge 5$, $T(n) \le d^* n$, as required. Therefore $T(n) \in O(n)$.