CSCI 570 Summer 2016 Homework 3 Solutions

On the first page of your homework, please write your name and USC ID # clearly. Please do something to indicate which is your last (family) name, such as underlining it. If your submission is multiple pages, you must *staple* them together. These are requirements of every homework assignment this semester.

1. Suppose you are given an array A of distinct numbers that is circularly sorted. That is, it begins at some value, and increases until it reaches the maximum value in the array. The next number is the minimum, and values increase from there until the last value, which will be less than the first value.

   Given such an array A, give an efficient algorithm to find the minimum value in the array. State the running time of your algorithm and explain its correctness.

   For example, if your input array A is:

10	13	17	20	1	2	3	4	5	6

   Then your return value should be 1.

   Note: A brute-force algorithm can solve this in time $O(n)$. For credit, your solution must have a running time strictly better than this.

   If the size of the array is sufficiently small (any constant base-case will do for asymptotic purposes; it probably should be $3-5$), we can check by examining each element with a normal find-min, or we could look for the first element that is smaller than its predecessor (comparing the first element to the last for this purpose). Either way, this takes $O(1)$ for the base case.
   For the general case, we have a middle element m and have narrowed the array down to $A[\text{lo}\ldots\text{hi}]$. If $A[m-1] > A[m] < A[m+1]$, we have found it and can return this value. If it isn't, and if $A[\text{lo}] < A[m]$, then either $A[\text{lo}]$ is the minimum element (we can check this in $O(1)$ by comparing it to $A[n]$, returning if it is the case) or the min element is in $A[m+1\ldots\text{hi}]$, which we can check recursively. Otherwise, we can inspect $A[\text{lo}\ldots m-1]$ recursively for the minimum element.
   This takes a total of $\Theta(\log n)$ time and has the recurrence $T(n) = T(\frac{n}{2}) + \Theta(1)$

2. Suppose you are given a collection S of n distinct objects. Each object has an associated value, which is an integer. You are looking for a particular object in the set, called the "golden" object, of which there is exactly one.

   You cannot test an object directly to determine whether it is the golden object, and you cannot directly manipulate the contents of S. Instead, you may call the following functions on S:

Function	Description	Running Time
`Sizeof(`S`)`	Returns the number of elements in S.	$O(1)$
`PruneL(`S`, `v`)`	Removes from S all elements with value less than v.	$O(\|S\|)$
`PruneG(`S`, `v`)`	Removes from S all elements with value greater than v.	$O(\|S\|)$
`PruneLE(`S`, `v`)`	Removes from S all elements with value less than or equal to v.	$O(\|S\|)$
`PruneGE(`S`, `v`)`	Removes from S all elements with value greater than or equal to v.	$O(\|S\|)$
`ContainsGold(`S`, `v`)`	Returns whether or not the golden object is contained within S *and* has value greater than v.	$O(\|S\|)$
`Select(`S`, `k`)`	Returns the value of the kth smallest (by value) object in S.	$O(\|S\|)$

   Give a **divide and conquer** algorithm that finds the golden object in $O(n)$ time. Show that your algorithm has the required runtime.

while Sizeof(S) > 1 **do**
  Let x = Select(S, Sizeof(S)/2).
  if ContainsGold(S, x) **then**
    PruneLE(S, x)
  else
    PruneG(S, x)
  The sole remaining element of S is the golden object

Note that, at every iteration, S has half as many items, but still has the golden object each time. We can describe this with the recurrence relation $T(n) = T(n/2) + O(n) = O(n)$.