CS 390 HW 0

Amy Kang

January 23, 2025

Problem 1

1.3

1. **rotation-find** (A[1...n]): Given a rotated sorted array A[1...n] , returns the rotation index $i \in [n]$.

   `rotation-find(A[1...n]: rotated sorted array)`

   1. If $n = 1$, return 1.
   2. m $\leftarrow \lceil \frac{n}{2} \rceil$.
   3. If $A[m] > A[n]$ then
      A. Return m + `rotation-find`$(A[m+1...n])$.
   4. Else
      A. If $m = 1$ or $A[m] < A[m-1]$ then
         1. Return m.
      B. Else
         1. Return `rotation-find`$(A[1...m-1])$.

   Explicitly stated, the runtime of this algorithm can be written

   $$T(n) = T(\frac{n}{2}) + O(1)$$

   where $T(\frac{n}{2})$ comes from recursive calls to `rotation-find` and other operations are constant-time.

   Evaluating, we get $T(n) \in O(\log n)$; each recursive call to `rotation-find` begets at most another recursive call to the same function with about half the input size, and all other operations are done in constant time.

2. **find-peak**$(A[1...n])$: Given a mountain $A[1..n]$, finds and returns the index of the peak.

   `find-peak(`$A[1...n]$`)`

   1. If $n \leq 1$, return 1
   2. $m \leftarrow \lceil \frac{n}{2} \rceil$
   3. If $A[m] < A[m+1]$ then
      A. Return $m +$ `find-peak`$(A[m+1...n])$
   4. Else

A. If $m > 1$ and $A[m-1] < A[m]$ then
   1. Return m
B. Else
   1. Return `find-peak`$(A[1...m-1])$

Explicitly stated, the runtime of this algorithm can be written

$$T(n) = T(\frac{n}{2}) + O(1)$$

where $T(\frac{n}{2})$ comes from recursive calls to `find-peak` and other operations are constant-time.

Evaluating, we get $T(n) \in O(\log n)$; each recursive call to `find-peak` begets at most another recursive call to the same function with about half the input size, and all other operations are done in constant time.

3. **local-min**$(A[1...n])$: Given an array $A[1..n]$ of comparable elements, finds and returns the index of a local minimum.

   `local-min`$(A[1...n])$
   1. If $n \leq 1$, return 1.
   2. If $n = 2$ then
      A. If $A[1] < A[2]$
         1. Return 1.
      B. Else
         1. Return 2.
   3. $m \leftarrow \lceil \frac{n}{2} \rceil$.
   4. If $A[m] > A[m+1]$ then
      A. Return $m +$ `local-min`$(A[m+1...n])$.
   5. Else if $A[m] > A[m-1]$ then
      A. Return `local-min`$(A[1...m-1])$.
   6. Else
      A. Return m.

   Explicitly stated, the runtime of this algorithm can be written

$$T(n) = T(\frac{n}{2}) + O(1)$$

   where $T(\frac{n}{2})$ comes from recursive calls to `local-min` and other operations are constant-time.

   Evaluating, we get $T(n) \in O(\log n)$; each recursive call to `local-min` begets at most another recursive call to the same function with about half the input size, and all other operations are done in constant time.

4. Each of the above problems involves finding an some index $i \in [n]$ in an array of comparable elements.

   Now suppose we can find this index strictly by using comparison queries; that is, for some k, asking k comparison queries will give us at least n possible outcomes. Since each query has two possible outcomes, k queries yield 2^k results.

But given that the array is of size n, there are n possible outcomes; therefore, in order for the k queries to produce all possible outcomes, we must have

$$2^k \geq n$$

or equivalently,

$$k \geq \log n$$

Finally, since each comparison query runs in at least constant time, we can conclude that each problem requires a lower bound of $\Omega(\log n)$ comparison queries.