CSC-421 Applied Algorithms and Structures
Winter 2019

Instructor: Iyad Kanj
Office: CDM 832
Phone: (312) 362-5558
Email: `ikanj@cs.depaul.edu`
Office Hours: Monday & Wednesday 4:00-5:30
Course Website: https://d2l.depaul.edu/

Solution Key to the Sample Midterm

I. Sort A in $O(n \lg n)$ time using Merge Sort. Create an empty array C and initialize it to $A[1]$. Iterate through A starting at position 2, and for each element $A[i]$, if $A[i] \neq A[i-1]$ then add $A[i]$ to C. The running time is $O(n \lg n) + O(n) = O(n \lg n)$.

II. 1. The idea is to divide the list into two sublists each with $n/2$ entries. The algorithm would then find the maximum and minimum elements of each of the two halves, by recursive applications of the algorithm. The specifications for our algorithm are:

```
Input: An array A[1..n] of unsorted, comparable entries and indices
1 <= p <= r <= n.
Output: The minimum entry m and the maximum entry M in A.


Find-Min-Max(A, p, r, m, M)

if p = r - 1 then
  if A[p] < A[r]
    m = A[p];
    M = A[r];
  else
    m = A[r];
    M = A[p];
else
  q =(p+r)/2;
  Find-Min-Max(A, p, q, m1, M1);
  Find-Min-Max(A, q+1, r, m2, M2);
  if m1 < m2
    m = m1;
  else
    m = m2;
  if M1 < M2
    m = M2;
  else
    m = M1;
```

The number of comparisons is described by the recurrence $T(n) = 2T(n/2) + 2$ if $n > 2$, and $T(2) = 1$. Using the iteration method, we can show that $T(n) = 3n/2 - 2$.

2. Here is the idea. We arrange the n entries into pairs, then compare the two elements within each pair, marking the larger of the two. This costs $n/2$ comparisons. Then go through the $n/2$ larger elements, keeping track of the current maximum, and similarly through the $n/2$ smaller elements, keeping track of the current minimum. This costs twice a total of $n/2+2(n/2-1) = 3n/2-2$ comparisons.

   Here is a formal description of a more streamlined version that does not use any marking:

```
Iter-FMM(A, n)

if A[1] < A[2] then
  m = A[1];
  M=A[2];
else
  m=A[2];
  M=A[1];

for i = 2 to n/2 do
  if A[2i-1] < A[2i] then
    if A[2i-1] < m then
       m = A[2i-1];
    if M < A[2i] then
       M = A[2i];
  else
    if A[2i] < m then
       m = A[2i];
    if M < A[2i-1] then
       M = A[2i-1];

return (m, M);
```

III. The answers are:

(i) $\Theta(n \lg n)$

(ii) $\Theta(\lg n)$

(iii) $\Theta(n)$

(iv) (A)

(v) (A)

IV. We only need to consider $A[1..k]$ and $B[1..k]$, i.e., a problem with $2k$ entries overall. Let $q = (k + 1)/2$, and $p = k - q$.

```
1. If A[q] = B[p] then q - 1 entries in A and p - 1 entries in B are
no greater than A[q] or B[p], and hence, either A[q] or B[p] is the
k-th smallest element.

2. If A[q] < B[p] then the k-th smallest entry must be either in A[q+1..k]
or in B[1..p]. In fact, the k-th smallest entry of the original problem
will be the (k-q)-th smallest entry in the arrays A[q+1..k] and B[1..p]
each of size k - q.

3. If A[q] > B[p] then the k-th smallest entry must be either in A[1..q]
or in B[p + 1..k] and the k-th smallest entry of the original problem
will be the (k -p)-th smallest entry in the arrays A[1..q] and B[p+1, k]
each of size k-p.
```

The algorithm uses the above idea to either solve the problem (case 1) or recursively solve the problem by solving a subproblem of half the size. The running time is thus $O(\lg k)$.

V. The polynomial-time reduction takes an instance (G, k) of VERTEX COVER, where $G = (V, E)$, and constructs an instance of SET COVER in which $X = E$, and $\mathcal{F}$ is constructed as follows: for each vertex $v \in V$, add to $\mathcal{F}$ a set consisting of all edges covered by v (i.e., incident to v). k remains the same in the reduction. Now show that (G, k) is a yes-instance of VERTEX COVER iff the constructed instance is a yes-instance of SET COVER.