Midterm I solutions

1. [15 points]

Suppose that we have an array of \( n \) records to sort and that the key of each record has the value 0 or 1. Give a simple linear-time algorithm for sorting the \( n \) records in place. Use no storage of more than constant size in addition to the storage provided by the array. [Note that you can not count the number of 1’s and 0’s and simply put those many in the sorted array because the keys have distinct records attached to them]

The record will be stored as \((\text{key}_i, \text{data}_i)\) where \(\text{key}_i \in \{0, 1\}\) and \(\text{data}_i \in \mathcal{R}\). For instance, if the input given to your algorithm is:
\[
\{(0, 12), (1, 10), (1, 0.4), (0, 4), (0, 9), (1, 40)\},
\]
there are many possible outputs, one of them being:
\[
\{(0, 12), (0, 4), (0, 9), (1, 10), (1, 0.4), (1, 40)\}. \quad \text{Your algorithm should give any one of the valid outputs.}
\]

Hint: Try using two pointers, the sort need not be stable.

**sol:** The algorithm will be as follows

(a) Keep 2 pointers \( \text{first} \) and \( \text{end} \) where \( \text{first} \) will point to \( A[1] \) and \( \text{end} \) to \( A[n] \). That is, \( \text{first} = 1 \) and \( \text{end} = n \).

(b) while \( A[\text{first}] \neq 1 \) and \( \text{first} < \text{end} \)

do \( \text{first}++ \)

(c) while \( A[\text{end}] \neq 1 \) and \( \text{first} < \text{end} \)

do \( \text{end}-- \)

(d) if \( \text{first} < \text{end} \)

do swap \( A[\text{first}] \) and \( A[\text{end}] \)

go to (b)

else stop

Since each record will be examined exactly once, time complexity will be \( O(n) \).

Correctness: All keys that are before \( \text{first} \) are always 0 and those after \( \text{end} \) are always 1 at every point of execution. As we stop when \( \text{first} = \text{end} \), we know the list is sorted.

2. [20 points]

6 points Label the following binary tree with numbers from the set \( \{6, 22, 9, 14, 13, 1, 8\} \) so that it is a legal binary search tree.
6 points Label each node in the figure above with \( r \) or \( b \) denoting the colors RED and BLACK, respectively, so that the tree is a legal red-black tree.

8 points Make the left child of the root be the root by performing a single rotation. Draw the binary search tree that results, and label your tree with the keys from part (a). Is it possible to label the nodes with colors so that the tree is a red-black tree? Justify your answer.

\textbf{sol:} It is possible to label the nodes with colors so that the tree stays a red-black tree. Please see figure on next page.

3. \textbf{[20 points]} Consider an array \( A \) of length \( n \) indexed from 1 to \( n \) for which we know that \( A[1] \geq A[2] \) and \( A[n-1] \leq A[n] \). We say that \( A[x] \) is a \textit{local minimum} if \( A[x-1] \leq A[x] \leq A[x+1] \) where \( 1 < x < n \). Note that \( A \) must have at least one local minimum.

We can obviously find a local minimum in \( A \) in \( O(n) \) time. Describe a more efficient algorithm for finding a local minimum.

Explain your algorithm clearly and prove that it is correct. Also prove that the complexity is less than \( O(n) \).

Hint: Try halving the array.
sol: The algorithm will be as follows

\[
\text{FindLocalMin}(A, low, high)
\]

(a) If \(A[\lfloor (low + high)/2 \rfloor] \) satisfies criteria for local minimum, return \( \lfloor (low + high)/2 \rfloor \).

(b) If \( A[\lfloor (low + high)/2 \rfloor - 1] \leq A[\lfloor (low + high)/2 \rfloor] \)

return \( \text{FindLocalMin}(A, low, \lfloor (low + high)/2 \rfloor) \)

(c) return \( \text{FindLocalMin}(A, \lfloor (low + high)/2 \rfloor, high) \)

Since the array is halved at every instant, we get a recurrence of the form

\[
T(n) = T(n/2) + 1
\]

which solve to \( O(\log n) \).

Correctness: If either of statements (b) or (c) is executed, the problem reduces to a smaller problem (half the size) with the same property of the the end points as the earlier problem. Hence that section also has a local minimum.

4. [20 points] Suppose you have \( n \) keys but only \( k \) distinct possible values for each key. Assume you are doing a comparison based sort.(For simplicity you may assume that you know the \( k \) possible values.)
(a) Give an algorithm with $O(n \lg k)$ comparisons. Clearly state the algorithm, prove its correctness and give a proof of its time bound.

**sol:** The algorithm is as follows

i. Create a red-black tree from the $k$ distinct numbers by inserting into an initially empty tree one by one. Since insertion of an element in a tree with $n$ elements takes $O(\lg n)$ time, this whole operation will take $O(k \lg k)$ time. Also, keep a counter (initialised to zero) with every node in the tree.

ii. For every element in the array, look for it in the red-black tree (that will take $O(\lg k)$ time). When it is found increment a counter associated with the node. Total time for this step will be $O(n \lg k)$.

iii. Do an inorder traversal of the tree and print out the key value the counter number of times. This will take $O(n)$ time.

Hence the total time will be $O(n \lg k)$.

(b) Give a proof of a lower bound of $\Omega(n \lg k)$

**sol:** Let us calculate the number of possible configurations of $n$ numbers which will make up the leaves in the decision tree. There are $k$ choices for each slot in the array. Hence total number of configurations will be $k \times k \times \cdots \times k$ $n$ times. That is $k^n$. Hence the height of such a tree will be $\lg(k^n)$ which is $k \lg n$, giving a lower bound of $\Omega(n \lg k)$.

5. **[25 points]**

The following program determines the maximum value in an unordered array $A[1..n]$.

1. $max \leftarrow -\infty$
2. for $i \leftarrow 1$ to $n$
3. do Compare $A[i]$ to $max$
4. if $A[i] > max$
5. then $max \leftarrow A[i]$

We want to determine the average number of times the assignment in line 5 is executed. Assume that all numbers in $A$ are randomly and independently drawn from the interval $[0,1]$.

a. **[5 points]** If a number $x$ is randomly drawn from a set of $n$ distinct numbers, what is the probability that $x$ is the largest in the set?
sol: The probability is $1/n$ since there is an equal chance for each number to be the largest.

b. [6 points] For each $i$ in the range $1 \leq i \leq n$, what is the probability that line 5 is executed?
sol: The probability that step 5 is executed depends on whether the $i$th element is the maximum in the first $i$ elements. Hence,

$$P(i) = \frac{1}{i}$$

c. [7 points] Let $s_1, s_2, ..., s_n$ be $n$ random variables, where $s_i$ represents the number of times (0 or 1) that line 5 is executed during the $i$th iteration of the for loop. What is $E[s_i]$?
sol: Define $s_i = 1$ if line 5 executed in $i$th iteration, 0 otherwise. Hence

$$E[s_i] = 1 \times \frac{1}{i} + 0 \times (1 - \frac{1}{i}) = \frac{1}{i}$$

d. [7 points] $s = s_1 + s_2 + ... + s_n$ be the total number of times that line 5 is executed during some run of the program. Show that $E[s] = O(\log n)$.
sol: $E[s] = E[\sum_{i=1}^{n} s_i]$

$$= \sum_{i=1}^{n} E[s_i]$$

$$= \sum_{i=1}^{n} \frac{1}{i}$$

$$= \Theta(\log n)$$