Quiz 1 Solutions

Quiz 1 Histogram

Quiz 1 Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>53</td>
</tr>
<tr>
<td>Mean</td>
<td>51.7</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>16.0</td>
</tr>
<tr>
<td>Max</td>
<td>79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Parts</th>
<th>Points</th>
<th>Grade</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. Asymptotic Running Times [12 points] (6 parts)

For each algorithm listed below, give its worst-case running time using \( \Theta \)-notation, as a function of the specified parameters. If the algorithm is randomized, give its expected running time. You need not justify your answers.

(a) [2 points] \textsc{Extract-Min} in a Min-Heap of \( n \) elements.

\textbf{Solution:} \( T(n) = \Theta(\log n) \).

(b) [2 points] \textsc{Strassen’s} algorithm to multiply \( n \times n \) matrices.

\textbf{Solution:} \( T(n) = \Theta(n^{\log_2 7}) \).

(c) [2 points] \textsc{Radix Sort} on an \( n \)-element array of \( b \)-bit integers (with optimum choice of radix).

\textbf{Solution:} \( T(n) = \Theta(n + nb/(\log n)) \).
(d) [2 points] RANDOMIZED QUICKSORT on an $n$-element array.

**Solution:** $T(n) = \Theta(n \lg n)$.

(e) [2 points] HASH-INSERT to insert an element into a chaining-based hash table with $2n$ slots, a load factor of $\alpha = 1/2$, and a hash function chosen from a universal hash family.

**Solution:** $T(n) = \Theta(1)$.

(f) [2 points] DETERMINISTIC SELECT to find the median of an $n$-element array.

**Solution:** $T(n) = \Theta(n)$. 
Problem 2. Recurrences [13 points] Professor M. Onotono has managed to come up with a new variant of Mergesort. It splits an array with \( n \) elements into three parts, two of size \( n/4 \) and one of size \( n/2 \). After sorting the subarrays, he can merge all three with just \( n \) comparisons. Thus the number of comparisons made by his algorithm, denoted \( T(n) \), satisfies the recurrence:

\[
T(n) \leq 2T(n/4) + T(n/2) + n,
\]

with \( T(1) = T(2) = 1 \). Your task is to complete his analysis by proving a tight upper bound on \( T(n) \) using \( O \)-notation.

Solution: We guess that \( T(n) \leq cn \log n \) for \( n \geq 2 \), and prove it by induction on \( n \). By sloppiness, we assume that \( n \) is a power of 2. The base case \( n = 2 \) says that \( T(2) = 1 \leq c2 \log 2 = 2c \), which holds provided \( c \geq 1/2 \). The base case \( n = 4 \) says that \( T(4) = 2T(1) + T(2) + 4 = 2 + 1 + 4 = 7 \leq c4 \log 4 = 8c \), which holds provided \( c \geq 7/8 \). For the induction step, where \( n \geq 8 \), assume that \( T(n') \leq cn' \log n' \) for all \( n' < n \) (in particular, for \( n' = n/2 \geq 4 \) and \( n' = n/4 \geq 2 \)). Now consider \( T(n) \):

\[
T(n) \leq 2c(n/4) \log(n/4) + c(n/2) \log(n/2) + n \\
= c(n/2)(\log n - 2) + c(n/2)(\log n - 1) + n \\
= cn \log n - (3/2)cn + n \\
= cn \log n - ((3/2)c - 1)n \\
\leq cn \log n
\]

provided \( c \geq 2/3 \). Thus we can choose \( c = \max\{1/2, 7/8, 2/3\} = 7/8 \). Therefore, \( T(n) = O(n \log n) \).
Problem 3. True or False, and Justify [30 points] (6 parts)  
Circle T or F for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. The more content you provide in your justification, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

T F There exist functions $f(n)$ and $g(n)$ such that $f(n) = O(g(n))$ and $f(n) = \omega(g(n))$.

Solution: False. If both are true, we find that there exist $c, n_0$ such that for all $n \geq n_0$ it is the case that $f(n) \leq cg(n)$. But we also have that for every $c'$ there exists $n_1$ such that for all $n > n_1$, $f(n) > c'g(n)$. Applying this to $c' = c$ and taking $n = \max\{n_0, n_1\}$ we get $f(n) \leq cg(n)$, but $f(n) > c'g(n) = cg(n)$ which clearly contradict each other.

T F Let $f(n) = \sqrt{n}$ and $g(n) = n \cdot (n \mod 100)$. Then $f(n) = O(g(n))$.

Solution: False. Assume otherwise. Then there should exist $c, n_0$ such that for every $n \geq n_0$ it is the case that $f(n) \leq cg(n)$. But this is clearly false if we pick $n$ to be a large multiple of 100. In this case, we have $g(n) = 0$, while $f(n) > 0$. 
T F If $T(n) = 5T(n/4) + n$, then $T(n) = O(n \log^{5/4} n)$.

Solution: False. By the Master method, we have $T(n) = \Theta(n^{\log_4 5}) \not\in O(n \log^{5/4} n)$.

T F The probability that randomized quicksort takes $\Omega(n^2)$ time to sort $n$ inputs is at least $1/(n!)$.

Solution: True. If at each step the algorithm pivots on the element of highest rank, then randomized quicksort performs like deterministic quicksort on a sorted array and takes $\Omega(n^2)$ time. The probability of this happening is $\prod_{k=1}^{n} (1/k) = 1/(n!)$. 
T F Consider an algorithm that takes an unsorted array of $3n$ distinct integers $A[1..3n]$ and produces two numbers $x$ and $y$ such that $x < y$, $n$ elements of $A$ have value smaller than $x$, $n$ elements have value greater than $y$, and $n$ elements have value between $x$ and $y$. Any such algorithm working in the comparison model must take $\Omega(n \log n)$ time.

Solution: False. This is an order-statistics problem: call $\text{SELECT}(A, n + 1)$ to compute $x$, and call $\text{SELECT}(A, 2n)$ to compute $y$. Each call takes $O(n)$ time, for a total of $O(n)$ time, which is $o(n \log n)$.

T F There exists a randomized algorithm that, in $O(n)$ expected time, determines whether an array of $n$ integers has repeated elements, i.e., whether there are two distinct indices $i, j$ such that $A[i] = A[j]$.

Solution: True. Construct a universal hash function and build a hash table of size $\Theta(n)$, so that the load factor $\alpha = \Theta(1)$. For $i = 1, 2, \ldots, n$, search for $A[i]$ in the hash table. If that key is already present, report “yes”. Otherwise, insert $A[i]$ into the hash table. Each search and possible insertion takes $O(1 + \alpha) = O(1)$ time in expectation, so by linearity of expectation, the total running time is $O(n)$ in expectation.
Problem 4. Finding common elements with “=” [25 points] (3 parts)

The Center for Disease Collection is trying to identify some common causes of the flu. To do so, they have blood samples from \( n \) infected cases, numbered 1, 2, \ldots, \( n \). Each sample \( i \) contains traces of the virus \( A[i] \) responsible for the infection. The CDC can compare two samples \( i \) and \( j \) to tell whether the same virus caused the two infections (i.e., whether \( A[i] = A[j] \)). However, they cannot perform any other type of comparison (\(<\), \(>\)), nor can they hash viruses. The CDC would like to know whether there is a virus that is responsible for at least 10\% of the cases, and if so, they would like a list of all such viruses. A test comparing two blood samples is time consuming and so they would like to minimize the number of tests performed.

We model this problem more precisely as follows. For a subarray \( A[p..q] \) and a real number \( \alpha \), \( 0 < \alpha < 1 \), define an element \( x \) to be \( \alpha \)-common if \( A[i] = x \) for at least \( \alpha \cdot (q - p + 1) \) choices of the index \( i \in \{p, p + 1, \ldots, q\} \). Our goal is to find an algorithm that, given an array \( A[1..n] \) of \( n \) elements, reports all \( \frac{1}{10} \)-common elements in \( A[1..n] \). The only operations that the algorithm is permitted to perform on values in \( A \) are equality comparisons, i.e., tests of the form “does \( A[i] = A[j] \)?”. Our goal is to devise such an algorithm for finding \( \frac{1}{10} \)-common elements that runs in \( O(n \log n) \) time.

Throughout this problem, assume that \( n \) is a power of 2.

(a) [3 points] Give an upper bound on the number of distinct \( \alpha \)-common elements in \( A[p..q] \). Write your answer as a function of \( \alpha \), \( p \), and \( q \).

Solution: In an array of size \( q - p + 1 \), the number of elements that can repeat \( \alpha(q - p + 1) \) times is at most \( 1/\alpha \). Thus, we can have at most \( 1/\alpha \) distinct \( \alpha \)-common elements in \( A[p..q] \).
(b) [7 points] Prove that there is some $\beta > 0$ such that any $\alpha$-common element in $A[1..n]$ is $\beta$-common in at least one of the subarrays $A[1..n/2]$ and $A[n/2+1..n]$. The larger the $\beta$ the better.

**Solution:** We show that one can choose $\beta = \alpha$. Assume for contradiction that some element $x$ is $\alpha$-common in $A[1..n]$, but not $\alpha$-common in either $A[1..n/2]$ or $A[n/2+1..n]$. Let $X$ be the number of occurrences of $x$ in $A[1..n]$, $X_1$ be the number of occurrences of $x$ in $A[1..n/2]$, and let $X_2$ be the number of occurrences of $x$ in $A[n/2+1..n]$.

Because $x$ is $\alpha$-common in $A[1..n]$, $X \geq \alpha n$. Because $x$ is not $\alpha$-common in either $A[1..n/2]$ or $A[n/2+1..n]$, we know that $X_1 < \alpha n/2$ and $X_2 < \alpha n/2$. But $X = X_1 + X_2 < \alpha n/2 + \alpha n/2 = \alpha n$, leading to a contradiction. Therefore, $x$ must $\alpha$-common in at least one of the subarrays $A[1..n/2]$ and $A[n/2+1..n]$.
(c) [15 points] Use the previous part to give a divide-and-conquer algorithm \textsc{TopTen} for finding the 1/10-common elements in \( A[p..q] \). Describe the divide, conquer, and combine steps of your algorithm. Analyze the running time of your algorithm. (\textit{Hint: It should satisfy the same recurrence as \textsc{Mergesort}.})

\textbf{Solution:}

\textbf{Algorithm:} Let \textsc{TopTen} \((A, p, q, \alpha)\) be the function that returns the list of all \( \alpha \)-common elements in \( A[p..q] \). We compute \textsc{TopTen} as follows:

\textit{Base case:} If \((q - p + 1) \leq 1/\alpha\), then return a list \( Y \) of all the elements in \( A[p..q] \).

\textit{Divide and conquer step:} Divide \( A[p..q] \) in half, and recursively call \textsc{TopTen} on each half with the same parameter \( \alpha \). Let \( Y_1 = \textsc{TopTen}(A, p, (p + q)/2, \alpha) \) and \( Y_2 = \textsc{TopTen}(A, (p + q)/2 + 1, q, \alpha) \).

\textit{Combine step:} Start with an empty list \( Y = \emptyset \) to store the answer. For each \( x \) in \( Y_1 \), scan through \( A[p..q] \) and count the occurrences of \( x \). If \( x \) occurs more than \( \alpha(q - p + 1) \) times, then add \( x \) to \( Y \). Repeat this loop for each \( x \) in \( Y_2 \). Return \( Y \) as the final answer.

\textbf{Correctness:} From (b), we know that any element \( x \) which is \( \alpha \)-common in \( A[p..q] \) must be \( \alpha \)-common in at least one of two subarrays we recurse on in the divide step. Thus, any candidate element \( x \) which might be \( \alpha \)-common must belong to one of \( Y_1 \) and \( Y_2 \). In the base case, if \( n = (q - p + 1) < 1/\alpha \), then every element in \( A[p..q] \) is \( \alpha \)-common.

\textbf{Runtime:} The divide step can be performed in constant time. In the combine step, for each \( x \), the scan through \( A \) requires \( O(n) \) time. From (a), we know that there at most \( 1/\alpha \) elements in \( Y_1 \) and \( 1/\alpha \) elements in \( Y_2 \). Thus, the runtime of the algorithm is given by the recurrence \( T(n) = 2T(n/2) + O(n/\alpha) \), with base case \( T(n) = O(1) \) for \( n < 1/\alpha \). For we have \( \alpha = 1/10 \), this recurrence is the same as for \textsc{Mergesort}; thus, the runtime is \( O(n \log n) \).