**CSCI2100: Midterm**

**Problem 1 (10%).** Prove: if \( f(n) = O(n \log n) \) and \( g(n) = O(\sqrt{n}) \), then there are constants \( \alpha > 0 \) and \( \beta > 0 \) such that \( f(n) + g(n) \leq \alpha \cdot n \log_2 n \) for all \( n \geq \beta \). Part of the proof has been written for you. You need to fill in the three blanks.

**Proof.** Since \( f(n) = O(n \log n) \), there exist constants \( c_1, c_2 \) such that, for all \( n \geq c_2 \), we have

\[
f(n) \leq c_1 n \log_2 n.
\]

Since \( g(n) = O(\sqrt{n}) \) there exist constants \( c_1', c_2' \) such that, for all \( n \geq c_2' \), we have

\[
g(n) \leq c_1' \sqrt{n} \leq c_1' n \log_2 n.
\]

Thus, for \( n \) satisfying __________, it holds that

\[
f(n) + g(n) \leq (c_1 + c_1') \cdot n \log_2 n.
\]

Hence, setting \( \alpha = __________ \) and \( \beta = __________ \) completes the proof. \( \square \)


**Solution.** Black 1: \( n \geq \max\{c_2, c_2'\} \). Black 2: \( \alpha = c_1 + c_1' \). Blank 3: \( \beta = \max\{c_2, c_2'\} \).

**Problem 2 (5%).** Give a counterexample to disprove the following statement: if functions \( f(n) = O(n \log n) \) and \( g(n) = O(\sqrt{n}) \), then \( f(n) + g(n) = \Omega(n \log n) \).

**Solution.** \( f(n) = g(n) = 1 \).

**Problem 3 (10%).** Let \( S \) be a set of \( n \) integers, and \( k_1, k_2 \) arbitrary integers satisfying \( 1 \leq k_1 \leq k_2 \leq n \). Suppose that \( S \) is given in an array. Give an \( O(n) \) expected time algorithm to report all the integers whose ranks in \( S \) are in the range \([k_1, k_2]\). Recall that the rank of an integer \( v \) in \( S \) equals the number of integers in \( S \) that are at most \( v \).

**Solution.** Apply the \( k \)-selection algorithm to find the integer \( p_1 \in S \) whose rank is \( k_1 \), and then apply the algorithm again to find the integer \( p_2 \in S \) whose rank is \( k_2 \). Finally, scan \( S \) to report every integer that falls in \([p_1, p_2]\).

**Problem 4 (10%).** Let \( S_1 \) and \( S_2 \) be two sets of integers (they may not be disjoint) with \( |S_1| = |S_2| = n \). We know that \( S_1 \) and \( S_2 \) have been sorted, i.e., each set is given in an array where its elements are in ascending order. Give an algorithm to compute \( S_1 \cup S_2 \) in \( O(n) \) time.

**Solution.** Let \( A_1 \) (resp., \( A_2 \)) be the array storing \( S_1 \) (resp., \( S_2 \)). Create an array \( A \) of size \( 2n \) to contain the output. Set \( i = j = 1 \). Repeat the following until \( i > n \) or \( j > n \):

- If \( A_1[i] > A_2[j] \), append \( A_1[i] \) to \( A \) and increase \( i \) by 1.

- If \( A_1[i] < A_2[j] \), append \( A_2[j] \) to \( A \) and increase \( j \) by 1.

- Otherwise, append \( A_1[i] \) to \( A \) and increase both \( i \) and \( j \) by 1.
Finally, if \( i < n \) (resp., \( j < n \)), append the remaining elements of \( A_1 \) (resp., \( A_2 \)) to \( A \).

**Problem 5 (6%).** Suppose that we use quick sort to sort the array \( A = (35, 12, 5, 55, 43, 78, 90, 82) \). Remember that the algorithm first randomly picks a pivot element from \( A \) and then solves two subproblems recursively. Let us assume that the pivot is 35. What are the input arrays of those two subproblems, respectively?

**Solution.** \( (12, 5) \), \((55, 43, 78, 90, 82)\).

**Problem 6 (6%).** Let \( A \) be the following array of 10 integers: \((8, 5, 6, 2, 12, 1, 10, 17, 11, 9)\). Suppose that we use counting sort to sort the array, knowing that all the integers are in the domain from 1 to 20. Recall that the algorithm (as described in the class) generates an array \( B \) where each element is either 0 or 1. Give the content of \( B \).

**Solution.** \((1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0)\).

**Problem 7 (10%).** Let \( S \) be a set of \( n \) integers that have been sorted in an array. Give an algorithm that, given any integers \( x \) and \( y \) with \( x \leq y \), finds the number of integers in \( S \) covered by the interval \([x, y]\). Your algorithm must finish in \( O(\log n) \) time. For example, if \( S = \{5, 12, 35, 43, 55, 78, 82, 90\} \), your algorithm should output 2 if \( x = 30 \) and \( y = 45 \).

**Solution.** Perform binary search to find the successor of \( x \) in \( A \) (which is the smallest element in \( A \) larger than or equal to \( x \)). Let \( i \) be the successor’s position index (i.e., \( A[i] \) is the successor). Perform binary search to find the predecessor of \( y \) in \( A \) (which is the largest element in \( A \) smaller than or equal to \( x \)). Let \( j \) be the predecessor’s position index. Return \( j - i + 1 \).

**Problem 8 (30%).** Let \( S_1 \) be a set of \( n \) integers that have been sorted in an array. Let \( S_2 \) be another set of \( m \) integers that have not been sorted. Answer the following questions.

1. (8%) Give an algorithm to find \( S_1 \cap S_2 \) in \( O(m \log n) \) time.

2. (10%) Give an algorithm to find \( S_1 \cap S_2 \) in \( O(n + m \log m) \) time.

3. (12%) Suppose that all the integers in \( S_1 \) are in the domain from 1 to 100\( n \) (whereas the domain for \( S_2 \) is arbitrary). Give an algorithm to find \( S_1 \cap S_2 \) in \( O(n + m) \) time.

**Solution.**

1. Let \( A_1 \) be the array storing \( S_1 \). For each integer \( e \in S_2 \), check whether \( e \in S_1 \) with binary search and, if so, output \( e \). Each binary search costs \( O(\log n) \) time. Thus, the total cost is \( O(m \log n) \).

2. Sort \( S_2 \) in \( O(m \log m) \) time; let \( A_2 \) be the sorted array \( A_2 \). Then, we perform a synchronous scan over \( A_1 \) and \( A_2 \) to output \( S_1 \cap S_2 \) as follows. First, set \( i = 1 \) and \( j = 1 \). Then, repeat the following until \( i > \lvert A_1 \rvert \) or \( j > \lvert A_2 \rvert \): if \( A_1[i] = A_2[j] \), output \( A_1[i] \) and increase both \( i \) and \( j \) by one. If \( A_1[i] > A_2[j] \), increase \( j \) by one; if \( A_1[i] < A_2[j] \), increase \( i \) by one. The synchronous scan takes \( O(m + n) \). So the overall cost is \( O(n + m \log m) \).

3. Discard from \( S_2 \) all the integers that are outside the range \([1, 100n]\). Use counting sort to sort (the remaining elements of) \( S_2 \) in \( O(m + 100n) = O(m + n) \) time. Then, perform a synchronous scan as described for Problem 8(2) to report \( S_1 \cap S_2 \). The total cost is \( O(m + n) \).
Problem 9 (13%). Let $A$ be an array of $n$ distinct integers (not necessarily sorted). We denote the $i$-th number in $A$ as $A[i]$, for $i \in [1,n]$. We call $A[i]$ a local maximum in any of the following scenarios:


For example, if $A = (35, 12, 5, 55, 43, 78, 90, 82)$, then 35, 55, and 90 are all the local maxima. Design an algorithm to find an arbitrary local maximum in $O(\log n)$ time.

Solution. Set $k = \lfloor n/2 \rfloor$. In $O(1)$ time, check if $A[k]$ is a local maximum. If not, then there are three possibilities:


In the first case, recursively look for a local maximum in the subarray $A[k+1:n]$ (i.e., everything from $A[k+1]$ to $A[n]$). In the second case, recurse in the subarray $A[1:k-1]$. In the third case, you can recurse either in $A[1:k-1]$ or $[k+1:n]$. If $f(n)$ is the running time on an input of size $n$, we have $f(n) \leq O(1) + f(\lceil n/2 \rceil)$, which yields $f(n) = O(\log n)$. 
