Algorithms
Sample Exam

First and Last Name

Instructions

- This is a closed book exam; only one sheet (one side) of handwritten notes is permitted.
- Calculators, PDAs, and computers should not be on your desk and cannot be used on this test.
- You will have 75 minutes for this exam (3.45 p.m. – 5.00 p.m.).
- You can collect 10 points for each question. You need 50 points to get a grade of 100 %.
- The problems *are not* listed in their order of difficulty. Be sure to *work the easiest for you problems first* and the harder or longer ones last.
- When writing an algorithm, a *clear* description in English will suffice. Ideally, state the invariants of your algorithms. Rambling or poorly written/organized explanations, which are difficult to follow, will receive less credit. Pseudo-code is *not* required.
- When asked for an algorithm, your algorithm should have the time complexity specified in the problem. If you cannot find such an algorithm, you will generally receive partial credit for a slower algorithm.

1) Suppose you are given an array A of n distinct and sorted numbers that has been circularly shifted k positions to the right. Give an $\mathcal{O}(\log n)$ time algorithm to determine k. For example, $\{35, 42, 5, 15, 27, 29\}$ is a sorted array that has been circularly shifted $k = 2$ positions, while $\{27, 29, 35, 42, 5, 15\}$ has been shifted $k = 4$ positions.

   We use a modified binary search. For this, we compare the middle element $A[m]$ with the first element $A[l]$. If $A[m] < A[l]$, we continue on the left side. Otherwise, if $A[m] > A[l]$, we continue on the right side. We repeat this until we have only two elements left, i. e., $r - l = 1$. Then, $k = r$.
   Invariant: $A[l] > A[r]$

2) Suppose that you have given a sorted array A of n distinct integers, drawn from 0 to m where $n < m$. Give an $\mathcal{O}(\log n)$ time algorithm to find the smallest non-negative integer that is not present in A.

   We use a modified binary search. For this, we compare the middle element $A[m]$ with m. If $A[m] = m$, continue on the right side. Otherwise, if $A[m] > m$, continue on the left side. We repeat this until we have only two elements left, i. e., $r - l = 1$. Then, $A[l] + 1 = l + 1$ is missing in A.
   Invariant: $A[l] = l$ and $A[r] > r$.

3) Find two functions $f(n)$ and $g(n)$ that satisfy the following relationship. If no such f and g exist, shortly explain why.

   (a) $f(n) \in o(g(n))$ and $f(n) \notin \Theta(g(n))$
   $f = n,\ g = n^2$

   (b) $f(n) \in \Theta(g(n))$ and $f(n) \in o(g(n))$
   Not possible. $f(n) \in \Theta(g(n))$ implies there is a constant c such that $f \leq cg$. However, $f(n) \in o(g(n))$ means that for all constants c, $f \geq cg$. Thus, both statements cannot be true at the same time.

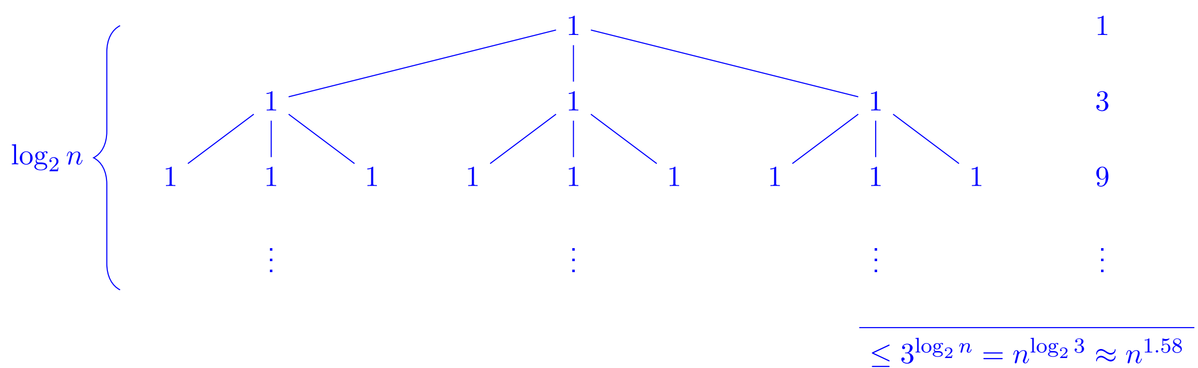
   (c) $f(n) \in \Theta(g(n))$ and $f(n) \notin \mathcal{O}(g(n))$
   Not possible. $f(n) \in \Theta(g(n))$ is defined as $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$. Thus, $f(n) \in \Theta(g(n))$ implies $f(n) \in \mathcal{O}(g(n))$ which contradicts the second statement.

   (d) $f(n) \in \Omega(g(n))$ and $f(n) \notin \mathcal{O}(g(n))$
   $f = n^2,\ g = n$

4) Use a recursion tree to determine a good asymptotic upper bound on the recurrence

$$T(n) = 3T(n/2) + 1.$$

Use the master theorem to verify your answer.



$T(n) = 3T\left(\frac{n}{2}\right) + 1$
- $a = 3,\ b = 2$
- $\log_b a = \log_2 3 \approx 1.58$
- $f(n) = 1,\ f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$ (Case 1)
- $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{1.58})$

5) Sort the following sequence using merge sort *or* quicksort.

$$22 \quad 80 \quad 18 \quad 9 \quad 90 \quad 12 \quad 24 \quad 57 \quad 86 \quad 36$$

For merge sort, show subsequences before and after merging. For quicksort, show the selected pivot element and the resulting partition.

Quicksort

22 18 9 12 24 36 80 90 57 86

18 9 12 22 24 36 57 80 90 86

9 12 18 22 24 36 57 80 86 90

Merge Sort

22 | 80 | 18 | 9 | 90 | 12 | 24 | 57 | 86 | 36

22 80 | 9 18 90 | 12 24 57 | 36 86

9 18 22 80 90 | 12 24 36 57 86

9 12 18 22 24 36 57 80 86 90

6) Let S be an *unsorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *minimizes* $|x - y|$, for $x \neq y$. What is the runtime of your algorithms.

First, sort the array. Then, for each i with $0 \leq i < n - 1$, determine the difference of the pair $x_i = A[i]$, $y_i = A[i+1]$. Output the pair x_i, y_i for which $|x_i - y_i|$ is minimal. Runtime is $\mathcal{O}(n \log n)$ for sorting and $\mathcal{O}(n)$ for finding x_i, y_i, giving an overall runtime of $\mathcal{O}(n \log n)$.