

Algorithms

Assignment: Sorting

Name:

Id:

Grade

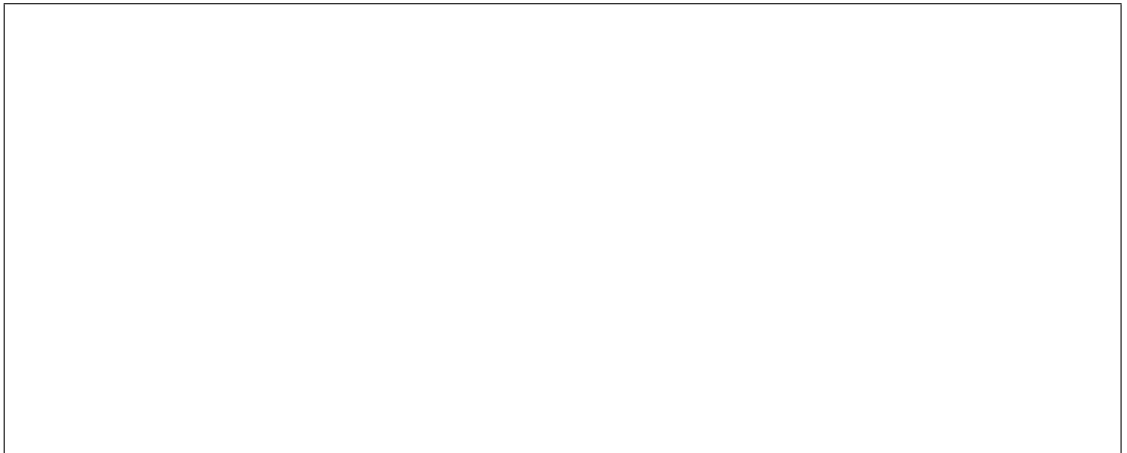
Good Luck!

1. Let $A[1], A[2], \dots, A[n]$ be an array containing n very large positive integers.

(a) Describe an *efficient* algorithm to find the *minimum positive* difference between any two integers in the array. In other words, compute $m = \min_{1 \leq i, j \leq n} \{|A[i] - A[j]|\}$.



(b) What is the complexity of your algorithm? Explain.



2. Let A be an array containing n very large positive integers not necessarily distinct.

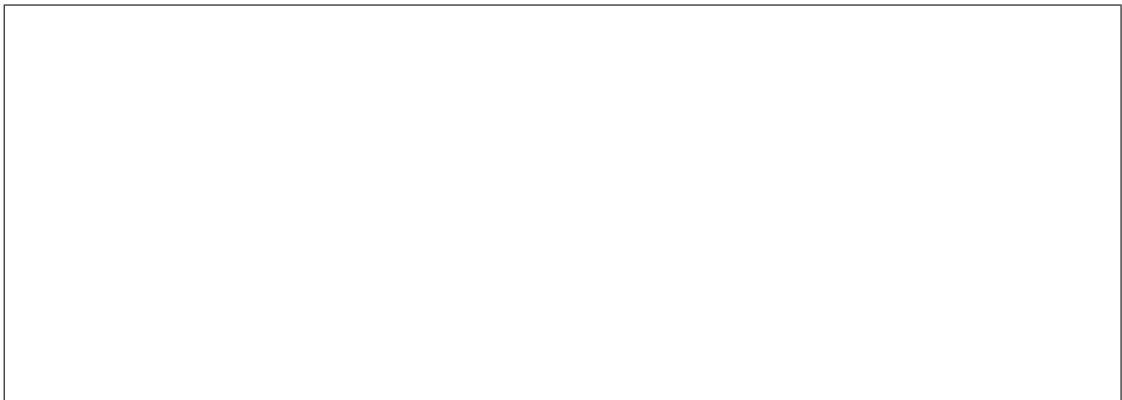
Problem \mathcal{P} :

Find the integer that appears the most times in the array A and the number of times it appears in the array. If there are several such integers it is enough to find one such integer.

(a) Describe an $O(n \log n)$ -time algorithm that solves problem \mathcal{P} with a constant-size ($O(1)$) additional memory.



(b) Explain why your algorithm has time complexity $O(n \log n)$.



3. The goal is to design efficient algorithms to sort 5 distinct (very large) keys with as few as possible comparisons in the worst-case.

(a) What is the worst-case number of comparisons of Bubble-Sort to sort 5 distinct keys.

(b) What is the worst-case number of comparisons of Merge-Sort to sort 5 distinct keys?

(c) Design a sorting algorithm for 5 keys that guarantees a better worst-case number of comparisons than Merge-Sort.

4. Let A be a **sorted** array with $n > 0$ integers and let B be an **un-sorted** array with $k > 0$ integers. The goal is to create a **sorted** array C that contains all the $n + k$ integers.

The optimization objective is to minimize the number of comparisons. All other operations are “free”. However, the integers may be very large so you **must** use comparisons and you cannot use Radix Sort like sorting algorithms.

The following 2 algorithms accomplish the task:

Algorithm P

$Sort(B)$

$Merge(A, B)$ into C .

Algorithm Q

Copy A to C .

For each integer in B : insert it into C .

- (a) What is the complexity of efficient implementations for Algorithms P and Q ? Explain.

- (b) Fix $n > 0$. In answering the following three questions use the “ O, Ω, Θ ” notation and ignore constants. For which values of k, n Algorithm P is better? For which values of k, n Algorithm Q is better? For which values of k, n the performance of both algorithm is equal? Explain.

5. An array A of $n \geq 1$ *distinct* positive integers is *bitonic* if there exists an index $1 \leq i \leq n$ such that:

- either $A[1] < A[2] < \dots < A[i-1] < A[i] > A[i+1] > \dots > A[n-1] > A[n]$
- or $A[1] > A[2] > \dots > A[i-1] > A[i] < A[i+1] < \dots < A[n-1] < A[n]$.

Describe an algorithm that uses a **linear** number ($O(n)$) of comparisons to sort bitonic arrays with **very large** integers (you **cannot** use radix-sort like algorithms). Note that if the array is bitonic the index i is not known. Justify your answer.

