College of Computer & Information Science
Northeastern University
CSU690: Algorithms and Data

Fall 2006
Handout 5
22 September 2006

Problem Set 2 (due Tuesday, October 3)

1. (15 points) Computing forces in particle physics

Chapter 5, Exercise 4, page 247.

2. (15 points) Hidden surface removal in computer graphics

Chapter 5, Exercise 5, page 248.

3. (3 + 5 + 7 = 15 points) Modes and majority elements

A *mode* of an array $A[1..n]$ is an element of A that occurs most number of times in A. Thus, the mode of an array $A = [7, 4, 12, 4, 1, 1, 4]$ is 4, which occurs three times.

(a) Given an array $A[1..n]$ of n integers, show how a mode of A can be determined in $O(n \log n)$ time.

An array $A[1..n]$ is said to have a *majority element* if more than half of its entries are the same. We would like to determine whether a given array A has a majority element, and if so, find the element. Unlike in part (a), however, we will not restrict the elements to be integers. In fact, we assume that the elements of the array are not necessarily from some ordered domain like the integers, so there can be no comparisons of the form "is $A[i] > A[j]$?"; only questions of the form "is $A[i] = A[j]$?" can be answered.

(b) Show how to solve the majority element problem in $O(n \log n)$ time.

(*Hint:* Split the array into two arrays A_1 and A_2 of half the size. Use a divide-and-conquer approach that finds the majority element of A, if it exists, using the knowledge of majority elements of A_1 and A_2, if they exist.)

(c) Can you give a linear-time algorithm?

(*Hint:* Another divide-and-conquer approach is as follows: (a) pair up the elements of A arbitrarily, to get $n/2$ pairs; (b) if two elements of a pair are different, then discard both of them, else keep just one of them. Show that after this procedure there are at most $n/2$ elements left, and that they have a majority element if A does.)

4. (6 + 6 + 3 = 15 points) A new 3-way sorting algorithm

Consider the following "3-way" sorting algorithm.

ThreeWaySort(A, i, j)
1. **if** $A[i] > A[j]$

```
    exchange A[i] ↔ A[j]
if i + 1 ≥ j
    return
k ← ⌊(j − i + 1)/3⌋               // Round down.
ThreeWaySort(A, i, j − k)          // First two-thirds.
ThreeWaySort(A, i + k, j)          // Last two-thirds.
ThreeWaySort(A, i, j − k)          // First two-thirds again.
```

(a) Argue that ThreeWaySort$(A, 1, n)$ correctly sorts the input array $A[1..n]$.

(b) Give a recurrence for the worst-case running-time of ThreeWaySort and a tight asymptotic (Θ-notation) bound on the worst-case running time.

(c) Compare the worst-case running time of ThreeWaySort with that of insertion sort and mergesort.

5. (5 bonus points) A fault-tolerant OR-gate

Assume we are given an infinite supply of two-input, one-output gates, most of which are OR gates and some of which are AND gates. Unfortunately the OR and AND gates have been mixed together and we can't tell them apart. For a given integer $k \geq 0$, we would like to construct a two-input, one-output combinational "k-OR" circuit from our supply of two-input, one output gates such that the following property holds: If at most k of the gates are AND gates then the circuit correctly implements OR. Assume for simplicity that k is a power of two.

For a given integer $k \geq 0$, we would like to design a k-OR circuit that uses the smallest number of gates. Design the best possible circuit you can and derive a Θ-bound (in terms of the parameter k) for the number of gates in your k-OR circuit.