Problem Set 2

MIT students: This problem set is due in lecture on **Monday, October 3, 2005.** The homework lab for this problem set will be held 2–4 P.M. on Sunday, October 2, 2005.

SMA students:

Reading: Sections 5.1-5.3 and Chapters 6, 7, and 9.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

Exercise 2-1. Do Exercise 5.2-4 on page 98 in CLRS.

Exercise 2-2. Do Exercise 6.5-7 on page 142 in CLRS.

Exercise 2-3. Do Exercise 7.2-1 on page 153 in CLRS.

Exercise 2-4. Do Exercise 9.3-8 on page 193 of CLRS.

Exercise 2-5. Do Problem 9-1 on page 194 of CLRS.

Problem 2-1. Is this (almost) sorted?

Harry Potter, the child wizard of Hogwarts fame, has once again run into trouble. Professor Snape has sent Harry to detention and assigned him the task of sorting all the old homework assignments from the last 200 years. Being a wizard, Harry waves his wand and says, *ordinatus sortitus*, and the papers rapidly pile themselves in order.

Professor Snape, however, wants to determine whether Harry's spell correctly sorted the papers. Unfortunately, there are a large number n of papers and determining whether they are in perfect order takes $\Omega(n)$ time.

Professor Snape instead decides to check whether the papers are *almost* sorted. He wants to know whether 90% of the papers are sorted: is it possible to remove 10% of the papers and have the resulting list be sorted?

In this problem, we will help Professor Snape to find an algorithm that takes as input a list A containing n distinct elements, and acts as follows:

- If the list A is sorted, the algorithm always returns **true**.
- If the list A is *not* 90% sorted, the algorithm returns **false** with probability at least 2/3.

(a) Professor Snape first considers the following algorithm:

> Repeat k times:
>
> 1. Choose a paper i independently and uniformly at random from the open interval $(1, n)$. (That is, $1 < i < n$.)
> 2. Compare paper $A[i-1]$ and $A[i]$. Output **false** and halt if they are not sorted correctly.
> 3. Compare paper $A[i]$ and $A[i+1]$. Output **false** and halt if they are not sorted correctly.
>
> Output **true**.

Show that for this algorithm to correctly discover whether the list is almost sorted with probability at least $2/3$ requires $k = \Omega(n)$. *Hint:* Find a sequence that is not almost sorted, but with only a small number of elements that will cause the algorithm to return **false**.

(b) Imagine you are given a bag of n balls. You are told that at least 10% of the balls are blue, and no more than 90% of the balls are red. Asymptotically (for large n) how many balls do you have to draw from the bag to see a blue ball with probability at least $2/3$? (You can assume that the balls are drawn with replacement.)

(c) Consider performing a "binary search" on an unsorted list:

```
BINARY-SEARCH(A, key, left, right)          ▷ Search for key in A[left . . right].
1  if left = right
2     then return left
3     else mid ← ⌈(left + right)/2⌉
4          if key < A[mid]
5             then return BINARY-SEARCH(A, key, left, mid − 1)
6             else return BINARY-SEARCH(A, key, mid, right)
```

Assume that a binary search for key_1 in A (even though A is not sorted) returns slot i. Similarly, a binary search for key_2 in A returns slot j. Explain why the following fact is true: if $i < j$, then $key_1 \leq key_2$. Draw a picture. *Hint:* First think about why this is obviously true if list A is sorted.

(d) Professor Snape proposes a randomized algorithm to determine whether a list is 90% sorted. The algorithm uses the function RANDOM$(1, n)$ to choose an integer independently and uniformly at random in the closed interval $[1, n]$. The algorithm is presented below.

```
IS-ALMOST-SORTED(A, n, k)                   ▷ Determine if A[1 . . n] is almost sorted.
1  for r ← 1 to k
2     do i ← RANDOM(1, n)                   ▷ Pick i uniformly and independently.
3        j ← BINARY-SEARCH(A, A[i], 1, n)
4        if i ≠ j
5           then return false
6  return true
```

Show that the algorithm is correct if k is a sufficiently large constant. That is, with k chosen appropriately, the algorithm always outputs **true** if a list is correctly sorted and outputs **false** with probability at least $2/3$ if the list is not 90% sorted.

(e) Imagine instead that Professor Snape would like to determine whether a list is $1 - \epsilon$ sorted for some $0 < \epsilon < 1$. (In the previous parts $\epsilon = 0.10$.) For large n, determine the appropriate value of k, asymptotically, and show that the algorithm is correct. What is the overall running time?

Problem 2-2. Sorting an almost sorted list.

On his way back from detention, Harry runs into his friend Hermione. He is upset because Professor Snape discovered that his sorting spell failed. Instead of sorting the papers correctly, each paper was within k slots of the proper position. Hermione immediately suggests that insertion sort would have easily fixed the problem. In this problem, we show that Hermione is correct (as usual). As before, $A[1 . . n]$ in an array of n distinct elements.

(a) First, we define an "inversion." If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an

inversion of A. What permutation of the array $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?

(b) Show that, if every paper is initially within k slots of its proper position, insertion sort runs in time $O(nk)$. *Hint:* First, show that INSERTION-SORT(A) runs in time $O(n + I)$, where I is the number of inversions in A.

(c) Show that sorting a list in which each paper is within k slots of its proper position takes $\Omega(n \lg k)$ comparisons. *Hint:* Use the decision-tree technique.

(d) Devise an algorithm that matches the lower bound, i.e., sorts a list in which each paper is within k slots of its proper position in $\Theta(n \lg k)$ time. *Hint:* See Exercise 6.5-8 on page 142 of CLRS.

Problem 2-3. Weighted Median.

For n distinct elements $x_1, x_2, \ldots, x_n$ with positive weights $w_1, w_2, \ldots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the ***weighted (lower) median*** is the element x_k satisfying

$$\sum_{x_i < x_k} w_i < \frac{1}{2}$$

and

$$\sum_{x_i > x_k} w_i \leq \frac{1}{2} .$$

(a) Argue that the median of $x_1, x_2, \ldots, x_n$ is the weighted median of $x_1, x_2, \ldots, x_n$ with weights $w_i = 1/n$ for $i = 1, 2, \ldots, n$.

(b) Show how to compute the weighted median of n elements in $O(n \lg n)$ worst-case time using sorting.

(c) Show how to compute the weighted median in $\Theta(n)$ worst-case time using a linear-time median algorithm such as SELECT from Section 9.3 of CLRS.

(d) The ***post-office location problem*** is defined as follows. We are given n points $p_1, p_2, \ldots, p_n$ with associated weights $w_1, w_2, \ldots, w_n$. We wish to find a point p (not necessarily one of the input points) that minimizes the sum $\sum_{i=1}^{n} w_i \, d(p, p_i)$, where $d(a, b)$ is the distance between points a and b.

Argue that the weighted median is a best solution for the one-dimensional post-office location problem, in which points are simply real numbers and the distance between points a and b is $d(a, b) = |a - b|$.

(e) Find the best solution for the two-dimensional post-office location problem, in which the points are (x, y) coordinate pairs and the distance between points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is the ***Manhattan distance*** given by $d(a, b) = |x_1 - x_2| + |y_1 - y_2|$.