SPRING 2004: **COT 6405** Analysis of Algorithms

[Homework 1; Due Jan 29 at start of class]

**Sample format for writing homeworks:** See handout titled "Sample Homework Solution" on course web page. As it shows, every algorithmic solutions must show **Basic Idea**, **Algorithm**, **Proof of Correctness**, and **Time and Space Complexity Analysis**. Wherever possible, **Lower Bound** proofs should be attempted.

**General submission guidelines and policies:** On page 2. Also see separate handout on course web page.

# Problems

1. (**Exercise**) Exercise 6.2-1, page 132.
2. (**Exercise**) Exercise 6.3-1, page 135.
3. (**Exercise**) Exercise 6.4-1, page 136.
4. (**Exercise**) Exercise 6.5-1, 6.5-2, page 140.
5. (**Exercise**) Exercis 7.1-1, page 148.
6. (**Exercise**) Exercise 8.3-1, page 173.
7. (**Regular**) A sorting algorithm is said to be *stable* if the relative order of any equal items in the input list is not changed in the output list. For each of the following sorting algorithms, state which ones are **not** stable by using simple examples: Insertion Sort, Selection Sort, Bubble Sort, Merge Sort, Quick Sort, Heap Sort, Counting Sort, Bucket Sort, Radix Sort. Devise the smallest example you can build.
8. (**Regular**) The first phase of heapsort involves building a heap. The second phase involves sorting from a heap, for which we know an algorithm with time complexity $O(n \log n)$. Is it possible for another comparison-based algorithm to reduce this time complexity (of phase 2) to $O(n)$? Give reasons for your answer.
9. (**Extra Credit**) You are given a set of $n$ elements to sort using some comparison-based sorting algorithm. However, the input has a large number of duplicates, and the number of unique elements in the list are only $k$ (not a constant). Clearly, the lower bound of $\omega(n \log n)$ on the time complexity need not hold any more. Prove a reasonable lower bound on the time complexity. You may need to define some notation.
10. (**Regular**) Design a simple algorithm for the above problem. Analyze its time complexity. **Hint:** Use red-black trees.

**General submission guidelines:** Since people tend to scribble on handwritten homeworks, you are required to type up your assignment and print it out. For every algorithmic question, you must provide a **Basic Idea** in a few English sentences before you present the algorithm. As they say: *a picture is worth a thousand words*. So draw pictures to explain your idea (this may be drawn by hand). The algorithm must clearly indicate its **Input** and its **Output**. Your pseudocode must contain line numbers (like in the text) and must be properly indented. While pseudocode is generally preferred, you may write formal code using a programming language such as C++ or Java (less desirable). Variable names must be meaningful. If a section of the code is complicated, it must be commented. Every algorithm needs a proof of correctness. You should attempt one, whenever possible. If possible, write an invariant for some of the critical steps in the algoithm. An example is not a proof of correctness. Every algorithm must be analyzed for time complexity and (if relevant) for space complexity. Wherever possible, you should indicate the time complexity of each step in the pseudocode itself (as a comment on the right side of the statement). Non-trivial lower bound arguments are likely to fetch you extra credit. Pay careful attention to the final written solution. Reread your written solutions and look for typographical and logical errors. A well-written solution shows clarity of thought and is likely to receive better grades. Not all problems will be graded and not all graded problems will have equal score. If more than one correct algorithm can solve a problem, then a more efficient solution will fetch more credit.

**Collaboration Policy:** Solving an algorithmic problem is a creative process. When presented with a new problem, it is your task to "take it apart" and reach your own understanding. This is a painstaking and time-consuming process. There is much to be learned from the process of thinking out solutions to the assigned homework problems. Getting help from elsewhere destroys this process. However, discussing with others after you have spent some time with a problem can help the process and bring other aspects of the problem to light. You may discuss homework problems with me or with other students in your class, after you have given it sufficient thought. But when the time comes to write up your solution, it must be your own work, and it must be in your own words. If after working on a problem yourself, you have been unable to solve it satisfactorily, then you may get help from other people, textbooks, or the internet. If you received help from any other source, it is necessary to *cite* your source at the appropriate location in the homework, i.e., write down the URL or the name of the person or the author of the text from which your solution was acquired. GIVE THE DEVIL IT'S DUE! After getting help from some source, make an attempt to write down the solution in your own words. If you discussed with a classmate or friend and came up with a solution together, then both of you should indicate this in your homeworks. If you are helping someone or providing your solution to someone, make sure they write down you as the source of the solution. You, too, may indicate that you helped this person with the specified problem. If you do not write down where you got help from, it would be considered as cheating. Any evidence of cheating (without citing the source) will result in severe penalization of all parties involved. If this policy gets refined over the course of this semester, this will be posted on the course webpage.