SPRING 2002: **COT 6405** Analysis of Algorithms

[Homework 1; Due Feb 5 in class]

General submission guidelines and policies: Read the attached handout on this topic.

Problems

7. 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 to $O(n)$? Give reasons for your answer.

8. 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 a simple example: Insertion Sort, Selection Sort, Bubble Sort, Merge Sort, Quick Sort, Heap Sort, Counting Sort, Bucket Sort, Radix Sort.

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). We would like to know if the lower bound of $O(n \log n)$ on the time complexity can be bettered. Prove a reasonable lower bound on the time complexity. You may need to define some notation.

10. Design a simple algorithm for the above problem. Analyze its time complexity.

11. Given a set of n tennis players, design a set of matches to be organized to decide the *champion* and the *runner-up* in the set. You may assume that, unlike in the real world of tennis, a match between player i and j produces the same result, regardless of how many times the match is repeatedly played. The champion is a player who cannot be defeated by any other player. The runner-up is a player who can defeat all the other players except the champion.

12. A set of n people (numbered 1 through n) have varying amounts of assets (money in the bank) and liabilities (total debts). The *worth* of a person is measured by the assets minus the liabilities. A recent law allows them to form groups and pool their worth. The worth of a group is simply the sum of the worth of its individuals. Design an algorithm that computes the group with the maximum worth.

13. Now design a second algorithm for the above problem assuming a new law that states that if individuals numbered i and j are part of a group pooling their worth, then that group must also include all individuals numbered between i and j. Analyze your algorithm. Argue why you think you can or cannot design a more efficient algorithm. Note that the naive $O(n^3)$-time algorithm will not fetch much credit.