Homework No. 2
CSCI 4470/6470 Algorithms, CS@UGA, Fall 2017

Due Monday September 18, 2017

The answers must be word-processed or typed. You may substitute formulae and figures with hand-writings. Your submitted algorithms should be in the pseudo-code, not in any specific programming language. Answers deviating from these requirements will be returned without grading.

The answers must be the student’s own work. Idea sharing and referencing to others’ work are not allowed. Plagiarism and other forms of academic dishonesty will be handled within the guidelines of the Student Handbook and reported to the University.

There are 6 questions, in which Q5(2) and Q6(2) are for graduate students only. Each question is worth 20 points. There are 120 points in total for graduate students and 100 for undergraduate students.

1. Consider the following problem to search a key from a sorted \( n \times n \) matrix, where all rows and columns of elements in the matrix are sorted. Design an algorithm for this problem which has a non-trivial upper bound.

2. Using a decision tree method to prove that the problem of searching for a key in a given a \( n \times n \) matrix has time complexity lower bound \( \Omega(\log_2 n) \) for the comparison-based models, regardless the given matrix is sorted or not. \textit{Hint:} consider the number of distinct outcomes that any search algorithm would have to accommodate.

3. Consider the following sorting problem where the elements in the input list \( \{x_1, \ldots, x_n\} \) always have only \( k \) distinct values, for \( k < n \). Using \textbf{the idea of quick sort} to design an algorithm that can sort such a list in the worst case time \( O(n \log_2 k) \). \textit{Hint:} You would need to make
sure to have a good procedure that allows some “balanced” partition. In addition, consider the fact that there are only $k$ distinct values in the given list.

4. Consider the SELECT that finds the $k$th smallest element in a given set. If, instead of making groups of 5 elements, we make groups of 3 elements, would the algorithm still run in linear time? How about groups of 7 elements? Justify your answers.

5. Consider the following graph-theoretic problem:

**Minimum Vertex Cover** problem

**Input:** graph $G = (V, E)$;
**Output:** a subset $C \subseteq V$ such that
(1) $\forall u, v \in V$, if $(u, v) \in E$, then $u \in C$ or $v \in C$
(2) $|C|$ is the minimum.

Intuitively, the found cover $C$ should cover every edge in $G$ on at least one of its two end vertices. The size of $C$ should be the smallest among all such covers.

Use the idea of search tree, design an exhaustive algorithm for the **Minimum Vertex Cover** problem which it has the running time $O(\gamma^n)$ for some $\gamma < 2$. The following are the requirements.

(1) Design an exhaustive search algorithm that has the time recurrence

$$T(n) \leq T(n - 1) + T(n - 2) + cn^2$$

for some constant $c > 0$;

(2) (Graduates only, bonus for undergraduates) Carefully design your algorithm so that it yields the recurrence

$$T(n) \leq T(n - 1) + T(n - 3) + cn^2$$

6. A typical dynamic programming algorithm computes the optimal solution. For example, the Assembly Line Scheduling problem is to compute the fastest path through the factory. If not only the fastest path but also the second fastest path are to be found, the dynamic programming algorithm `ASSEMBLYLINE` can be modified to accommodate this new goal. Discuss how you may revise the algorithm `ASSEMBLYLINE` for the purpose of finding both the fastest and second fastest paths through the factory.
(1) Show the details of how you would modify the algorithm to satisfy the new requirements.

(2) (Graduates only, bonus for undergraduates) Argue that the idea can be applied to finding top $k$ fastest paths through the factory, for any $k \geq 2$. What will be the time complexity of such an algorithm to find top $k$ fastest paths (in terms of both $n$ and $k$)?