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 7 questions, in which Q2(2), Q4(2) and Q5(2) are for graduate students only. Each question is worth 20 points. There are 140 points in total.

1. Using a decision tree method to prove that the problem of searching for a key in a given list of $n$ elements has time complexity lower bound $\Omega(\log_2 n)$ for the comparison-based models, regardless the given list is sorted or not.

2. Consider the problem SortBinaryBits that sorts a list of binary bits (where each element $x_i$ in the list is a single binary bit, either 0 or 1).

   (1) Design an algorithm (comparison-based only) for SortBinaryBits where the input is $\{x_1, \ldots, x_n\}$, with $x_i \in \{0, 1\}$. Derive a time complexity upper bound for your algorithm.

   (2) (Graduate students only, bonus for undergraduate students) Prove a time complexity lower bound for the problem SortBinaryBits using a decision tree based method. Consider the upper bound you derived for your algorithm in part (1) and explain if your algorithm is the optimal or not.
Note: you need to put in effort in algorithm design and lower bound proof. A trivial algorithm or lower bound would not deserve any point.

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 \( k < n \). Prove that this sorting problem can be solved in worst case time \( O(n \log_2 k) \).

Hint: consider a quick sort type of algorithm but with a good procedure that allows some “balanced” partition. Also consider the fact that there are only \( k \) distinct values in the given list.

4. Consider the following graph-theoretic problem:

**Minimum Vertex Cover** problem

**INPUT:** graph \( G = (V, E) \);

**OUTPUT:** a subset \( C \subseteq V \) such that

(1) \( \forall u, v \in V, \text{ if } (u, v) \in E, \text{ then } u \in C \text{ or } v \in C \)

(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

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

5. 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 $k$)?

6. The casino dice decoding problem was introduced in the classroom. Complete all the 4 steps of the dynamic programming solution to this problem:

(1) Analyze the problem to show that the problem demonstrates the property of optimal substructure and property of overlapping subproblems.

(2) Define the numerical objective function and derive a recurrence for the objective function.

7. Following the questions in Q6,

(3) Give a pseudo code for your algorithm that computes the objective function.

(4) Give a traceback algorithm to recover the sequence dice used.