Quiz 5B

CS/CE 412/471 Algorithms: Design and Analysis, Spring 2025

16 Apr, 2025. 4 questions, 25 points, 4 printed sides

Instructions:

1. Write your full name and your Hogwarts ID (or Muggle student ID) below.
2. You have 60 minutes to complete this enchanted assessment.
3. Wands away! Only your quill (pen), parchment (paper), and an optional calculator are permitted. Any magical artifacts, enchanted mirrors, or talking notebooks must be submitted with your bag at the front of the classroom.
4. Work independently — no use of Polyjuice Potion, Invisibility Cloaks, or Legilimency will be tolerated. Academic honesty is enforced by the strictest of Ministry decrees.
5. Present your spells (solutions) clearly and concisely. Illegible runes may result in lost points.
6. For your submission to be marked by the Council of Elders (your instructor), return this sheet along with your written solutions.
7. Have courage like a Gryffindor, wisdom like a Ravenclaw, diligence like a Hufflepuff, and ambition like a Slytherin. Good luck!

Student Name: _____

Student ID: _____

1. Optimize Your Spellcasting Stamina [5 points]

In your advanced spellcasting class at Hogwarts, each spell you learn consumes a certain amount of your limited magical stamina and grants you house points for your brilliance. That is, for each spell i, the stamina cost is s_i and the house points earned is h_i. You cannot spend more than S total stamina. Choose a subset of spells to maximize your total house points.

(a) [2 points] For this problem, describe the solution and the value of the solution.

Solution: This is the 0-1 knapsack problem. The solution is the set of spells, and the value of the solution is the corresponding house points.

(b) [3 points] Give a recursive formulation for computing the maximum value. Clearly indicate the base case(s) and define any notation you introduce.

Solution: Let us number the spells from 1 to n. Let $H(s, i)$ denote the maximum house points that can be obtained by consuming stamina up to s and including spells up to i. Then the required answer is $H(S, n)$ and

$$H(s,i) = \begin{cases} 0 & i = 0 \text{ or } s = 0 \\ -\infty & s < 0 \\ \max\{H(s - s_i, i-1) + h_i, H(s, i-1)\} & 1 \le i \le n \end{cases}$$

2. The Butterbeer Budget Problem [5 points]

Fred and George are stocking up on magical snacks for the Gryffindor common room. Each snack type has a cost (in Sickles) and a deliciousness score. They have unlimited supply of each snack. With a budget of B Sickles, they plan to keep buying the most delicious snack they can afford.

Will this strategy always maximize total deliciousness? Justify your answer.

Solution: This is the unbounded knapsack problem. We prove the strategy sub-optimal through a counterexample.

Proof. The strategy does not always provide the maximum deliciousness.

Consider $B = 10$ and two snacks, one with price and deliciousness both equal to 9 and the other with price and deliciousness both equal to 5.

The strategy leads to a purchase of snack 1 and a total deliciousness of 9.

A more optimal solution, 2 purchases of snack 2 leading to a total deliciousness of 10, exists. □

3. Dynamic or Not? [5 points]

Below are several magical computation problems. For each, state whether dynamic programming is a suitable approach. Justify briefly (do not solve the problem).

(a) [1 point] Spell Binding: Given a total power P and a list of spells with powers, can you find a subset that sums to exactly P?

Solution: Dynamic programming is a suitable approach. For each spell, we decide whether to include or exclude it in the subset. Multiple sequences of decisions can lead to the same subproblem, so subproblems overlap.

(b) [1 point] Broomstick Race: In a $m \times m$ race grid, move right or down to collect the most wind tokens and reach the finish line.

Solution: Dynamic programming applies. This is an optimization problem. Multiple paths from the start position can lead to a given position in the grid, so the solution will involve overlapping subproblems.

(c) 1 point Owl Name Sorting: Sort a list of magical owls alphabetically.

Solution: Dynamic programming is not a suitable approach. Sorting does not involve overlapping subproblems.

(d) 1 point Triwizard Finalist Simulation: In a dueling tournament, simulate match winners until a champion is crowned.

Solution: Dynamic programming is not a suitable approach. The duels can be directly simulated to obtain the champion. There are no overlapping subproblems.

(e) 1 point Music Box Enchantment: Given a set of tune segments, how many ways can you fill a box with tunes of total length L?

Solution: Dynamic programming applies. The explanation is similar to the first part.

4. Spells of Subproblem Sorcery [10 points]

Two directed acyclic graphs are shown below, each representing subproblem dependencies for some magical task.

For each graph, provide:

(a) a plausible recurrence matching the graph,
(b) an estimate of the time complexity of the solution,
(c) a justification whether dynamic programming is a suitable approach to solve the problem, and
(d) a bottom up pseudocode implementation of your recurrence for general n.

Solution: Left graph.

(a) a plausible recurrence matching the graph:

$$f(n) = \begin{cases} a_n & n \leq 1 \\ \min\{a_i + f(i) : n-2 \leq i < n\} & \text{otherwise} \end{cases}$$

where $a = \langle a_0, a_1, a_2, \ldots, a_n \rangle$ is an input array.

(b) an estimate of the time complexity of the solution:
all but 2 of the n nodes connect to 2 nodes each. The time complexity is therefore $\Theta(n)$.

(c) a justification whether dynamic programming is a suitable approach to solve the problem:
A dynamic programming approach is suitable. The subproblems overlap,

(d) a bottom up pseudocode implementation of your recurrence for general n:

```
F(n, a)
1  initialize dp = ⟨dp_0, dp_1, dp_2, ..., dp_n⟩
2  dp_0, dp_1 = a_0, a_1
3  for i = 2 to n
4        dp_i = min(a_{i-1} + dp_{i-1}, a_{i-2} + dp_{i-2})
```

Solution: Right graph.

(a) a plausible recurrence matching the graph:

$$f(n) = \begin{cases} a_n & n \leq 1 \\ a_n + f(n-2) & \text{otherwise} \end{cases}$$

where $a = \langle a_0, a_1, a_2, \ldots, a_n \rangle$ is an input array.

(b) an estimate of the time complexity of the solution:
Each of the n nodes connects to 1 node. The time complexity is therefore $\Theta(n)$.

(c) a justification whether dynamic programming is a suitable approach to solve the problem:
A dynamic programming approach is not suitable. The subproblems do not overlap,

(d) a bottom up pseudocode implementation of your recurrence for general n:

```
F(n, a)
1  initialize dp = ⟨dp_0, dp_1, dp_2, ..., dp_n⟩
2  dp_0, dp_1 = a_0, a_1
3  for i = 1 to n
4        dp_i = a_i + dp_{i-2}
```