
Problem Set 6

This problem set is due **Wednesday, May 2 at 11:59PM**.

Solutions should be turned in through the course website. **You must enter your solutions by modifying the solution template (in Python) which is also available on the course website.** The grading for this problem set will be largely automated, so it is important that you follow the specific directions for answering each question.

For multiple-choice and true/false questions, no explanations are necessary: your grade will be based only on the correctness of your answer. For all other non-programming questions, full credit will be given only to correct solutions which are described clearly and concisely.

Programming questions will be graded on a collection of test cases. Your grade will be based on the number of test cases for which your algorithm outputs a correct answer within time and space bounds which we will impose for the case. Please do not attempt to trick the grading software or otherwise circumvent the assigned task.

1. Dynamic programming analysis (20 points)

For each of the following recursions, all of which take exponential time to compute naively, answer the following questions. You may assume that the functions can be computed in constant time when any of the arguments are 0.

- i. In terms of n , how many distinct subproblems are ever solved to evaluate the function with arguments bounded by n ?
- ii. If we use memoization to speed up the computation of the recurrence, what is time needed to evaluate the function?

(a) A function defined by the Fibonacci recursion:

$$\text{FIB}(n) = \text{FIB}(n-1) + \text{FIB}(n-2)$$

(b) A function defined by Pascal's recursion:

$$\text{CHOOSE}(n, k) = \text{CHOOSE}(n-1, k) + \text{CHOOSE}(n-1, k-1)$$

where $\text{CHOOSE}(n, k) = 0$ if $k > n$.

(c) A function defined by the Bell numbers recursion:

$$\text{BELL}(n) = \sum_{k=0}^{n-1} \text{CHOOSE}(n, k) \cdot \text{BELL}(k)$$

where the binomial coefficients are already computed (using the recursion above).

(d) A function defined by the recursion:

$$\text{GAME}(n, k) = \max_{\frac{k}{2} \leq i \leq k} (-1)^n \cdot \text{GAME}(n-1, i)$$

where $\text{GAME}(n, k) = 0$ if $k > n$.

(e) A function defined by the recursion:

$$\text{HALF}(i, j) = \left(\sum_{k=0}^{\frac{j-i}{2}} \text{HALF}\left(i+k, i+k+\frac{j-i}{2}\right) \right)^2$$

where $0 \leq i \leq j \leq n$. (Hint: think of $[i, j]$ as an interval. What do the recursive calls look like?)

Solution Format:

Your choices for this problem are:

- A. $\Theta(1)$
- B. $\Theta(\log n)$
- C. $\Theta(n)$
- D. $\Theta(n \log n)$
- E. $\Theta(n^2)$
- F. $\Theta(n^2 \log n)$
- G. $\Theta(n^3)$

So your solution to each part of this problem should be a single character in the set `set(['A', 'B', 'C', 'D', 'E', 'F', 'G'])`.

2. A game of DAGs (30 points)

You are given a directed acyclic graph, in the same adjacency list format as the graph from Problem Set 4. Silvio and Costis are playing a game on this graph.

The game begins at a node s in the graph. The two players alternate taking turns, with Silvio going first. On a player's turn, he chooses a vertex which is a direct descendent of the vertex chosen in the previous round (e.g., on Silvio's first turn, he chooses any vertex which s has an edge to). A player loses if he has no legal moves, which happens when the other player chooses a sink.

Fill in the code for a function `find_winning_nodes(graph)` which returns a list of the start nodes s in the graph from which Silvio wins, assuming that both players play optimally.

```
graph = {0: [1, 2], 1 : [2, 3], 2 : [3], 3 : []}  
# If s is 0, Silvio chooses 1 or 2, so Costis chooses 3 and wins.  
# If s is 1 or 2, Silvio chooses 3 and wins.  
# If s is 3, Silvio immediately loses.  
set(find_winning_nodes(graph)) == set([1, 2])
```

3. Optimal parenthesization (40 points)

Given an array of n positive (but not necessarily integral) numbers, your goal is to determine the largest value that can be obtained by interspersing parentheses, multiplication signs, and addition signs between them.

Fill in the code for a function `find_largest_value(numbers)` for this problem. Your code should be able to handle a 100-element list in about a second and pass the following test cases:

```
# An optimal parenthesization: (1 + 2) * (3 * (4 * 5)) = 180
abs(find_largest_value([1, 2, 3, 4, 5]) - 180) < 0.001
```

```
# An optimal parenthesization: 0.8 + (0.5 + (0.3 + 0.5)) = 2.1
abs(find_largest_value([0.8, 0.5, 0.3, 0.5]) - 2.1) < 0.001
```

```
# An optimal parenthesization: (0.8 + 1.5) * (1.6 + 0.5) = 4.83
abs(find_largest_value([0.8, 1.5, 1.6, 0.5]) - 4.83) < 0.001
```

4. MIT's football team (40 points)

The Institute wants to develop a set of robots that can defeat Harvard's football team in a head-to-head comparison. Harvard's team is composed of n players, each of which have a strength a_i and a speed b_i .

A robot majorizes a human if it is at least as strong and at least as fast as the human. It costs MIT $a \cdot b$ thousand dollars to create a robot which has strength a and speed b , and MIT would like to have at least one robot that majorizes each player on Harvard's team. Describe an efficient algorithm to compute the minimal amount of money needed to create a set of robots that satisfies this condition.

Example: If Harvard's team has three players with strength-speed pairs $(10, 1)$, $(2, 9)$, and $(1, 10)$, the most cost-efficient team of robots is the team of two: $(10, 1)$ and $(2, 10)$. This team costs \$30,000.

Solution Format:

Fill in the string `answer_for_problem_4` with your solution.