Here are some extra practice problems taken from past semesters. You are welcomed to check your solutions with me or discuss them in the telegram group chat.

1. **Freivalds' algorithm** (New Question). Instead of choosing $v = \{0,1\}^n$ as in lecture, we modify the algorithm to choose $v = \{0,1,\ldots,999\}^n$ instead. What is the probability that Freivalds' algorithm is successful? Prove the new bound.

   Solution. The probability is $999/1000$.
   If $AB = C$, then the algorithm always output Yes. Now suppose $AB \neq C$. Then, AB differs from C at at least one entry (i,j). Since $(AB-C)_{(i,j)}$ is non-zero, changing v_j will result in 1000 different values of $(ABv-Cv)_i$. Therefore, at least 999 of them will result in $(ABv-Cv)_i \neq 0$.

2. **Expectation Arguments** (CS5275 Tutorial 3).Let $\mathcal{M} = \{0,1,\ldots,N^2-1\}$ for some positive integer N, and let $\mathcal{A}$ be a subset of $\mathcal{M}$ of size N. Show that there exists a subset $\mathcal{B}$ of $\mathcal{M}$ of size at most N such that the set
   $$\mathcal{C} = \{a + b \bmod N^2 \mid a \in \mathcal{A} \text{ and } b \in \mathcal{B}\}$$
   has cardinality at least $\frac{N^2}{2}$.

   Solution. Construct a random subset $\mathcal{B} = \{b_1, ..., b_N\}$ by selecting N items from $\mathcal{M}$ uniformly at random with replacement. This means that during each of the N random selections, any given $j \in \mathcal{M}$ is selected with probability $1/N^2$. Note that since we are sampling with replacement, there may be "duplicates", but nevertheless $\mathcal{B}$ has size at most N as required. We now consider the random subset $\mathcal{C} \subseteq \mathcal{M}$ formed using $\mathcal{B}$. Let the random variable X be the number of integers in $\mathcal{M}$ but not in $\mathcal{C}$. By the expectation argument, it is sufficient to show that $\mathbb{E}[X] \leq N^2/2$. Let
   $$X_j = \begin{cases} 1, \text{ if } j \notin \mathcal{C} \\ 0, \text{ otherwise} \end{cases}$$
   for $j = 0, ..., N^2-1$ be the indicator random variable for the event that $j \notin \mathcal{C}$. For a fixed j and a fixed $a \in \mathcal{A}$ there is a unique $m \in \mathcal{M}$ such that
   $$a + m \equiv j \pmod{N^2}$$
   Therefore, for a fixed j, there are N possible values of $m \in \mathcal{M}$ that lead to that – one for each value of $a \in \mathcal{A}$. It follows that $j \notin \mathcal{C}$ if each of $b_1, ..., b_N$ is not any of these N possible m values, which happens with probability
   $$\mathbb{P}[j \notin \mathcal{C}] = \mathbb{P}\left[\bigcap_{i=1}^{N} \{b_i \text{ is not any of these } N \text{ possible } m \text{ values}\}\right] = \left(1 - \frac{N}{N^2}\right)^N = \left(1 - \frac{1}{N}\right)^N,$$
   where the second equality follows from the b_i being chosen independently. This implies
   $$\mathbb{E}[X_j] = 1 \cdot \mathbb{P}[X_j = 1] + 0 \cdot \mathbb{P}[X_j = 0] = \mathbb{P}[X_j = 1] = \mathbb{P}[j \notin \mathcal{C}] = \left(1 - \frac{1}{N}\right)^N < \frac{1}{e}.$$
   By linearity of expectation,
   $$\mathbb{E}[X] = \mathbb{E}\left[\sum_{j=0}^{N^2-1} X_j\right] = \sum_{j=0}^{N^2-1} \mathbb{E}[X_j] < \frac{N^2}{e} \approx 0.368N^2 \leq N^2/2$$
   as desired.

3. **Markov's Inequality** (CS5275 Tutorial 3, Simplified). Suppose that there are n nodes, and between each pair of nodes, an edge is created with probability p, independently of all the others. A random graph obtained through this procedure is called the *Erdős–Rényi random graph*. Let T_n be the random variable denoting the (random) number of triangles for a random graph obtained through the procedure, where a triangle is a set of 3 nodes that are all connected to each other.

(a) Find $\mathbb{E}[T_n]$.

(b) Using the results from (a), show that $\mathbb{P}[T_n = 0] \geq 1 - \frac{1}{6}n^3p^3$.

Solution.

(a) Let X_i be the indicator random variable that the i-th triple forms a triangle, out of the $\binom{n}{3}$ triples of nodes. Since each edge is formed with probability $\frac{1}{2}$, it follows that $\mathbb{E}[X_i] = \mathbb{P}[X_i] = p^3$. Then by linearity of expectation,

$$\mathbb{E}[T_n] = \sum_{i=1}^{\binom{n}{3}} \mathbb{E}[X_i] = \binom{n}{3} \cdot p^3$$

(b) Note that $T_n \geq 0$. By Markov's inequality,

$$\mathbb{P}[T_n \geq 1] \leq \frac{\mathbb{E}[T_n]}{1}$$

Using the fact that $\mathbb{P}[T_n = 0] = 1 - \mathbb{P}[T_n \geq 1]$, we have

$$\mathbb{P}[T_n = 0] = 1 - \mathbb{P}[T_n \geq 1] \geq 1 - \mathbb{E}[T_n] \geq 1 - \binom{n}{3} \cdot p^3 \geq 1 - \frac{n^3}{6} \cdot p^3 = 1 - \frac{1}{6}n^3p^3$$

4. **Union Bound** (AY 22/23 Sem 2, Modified). Given a permutation π over $\{1, 2, \cdots, n\}$, let $L(\pi)$ be the length of the longest increasing sequence in π. For example, in the permutation $\pi = (1, 6, 4, 5, 2, 7, 3)$, the longest increasing subsequence is $(1, 4, 5, 7)$ and thus $L(\pi) = 4$.
Now, we consider a random permutation π over $\{1, 2, \cdots, n\}$.

(a) Show that $\mathbb{P}[L(\pi) \geq 10\sqrt{n}] \leq \left(\frac{ne^2}{(10\sqrt{n})^2}\right)^{10\sqrt{n}}$.
(You may assume the inequalities $\binom{n}{k} \leq (\frac{en}{k})^k$ and $k! \geq (\frac{k}{e})^k$ without proof.)

(b) Using the results from (a), show that $\mathbb{E}[L(\pi)] = O(\sqrt{n})$.

Solution.

(a) Consider all $\binom{n}{10\sqrt{n}}$ subsequences of length $10\sqrt{n}$. Let X_i be the indicator variable that the i-th subsequence is increasing. Since all $(10\sqrt{n})!$ orderings would appear with equal probability, we have $\mathbb{P}[X_i] = \frac{1}{(10\sqrt{n})!}$. By union bound, we get

$$\begin{aligned}
\mathbb{P}\left[\bigcup X_i\right] \leq \sum \mathbb{P}[X_i] &= \binom{n}{10\sqrt{n}} \cdot \frac{1}{(10\sqrt{n})!} \\
&\leq \left(\frac{en}{10\sqrt{n}}\right)^{10\sqrt{n}} \cdot \left(\frac{e}{10\sqrt{n}}\right)^{10\sqrt{n}} \quad \blacktriangleleft \text{ apply the two inequalities} \\
&= \left(\frac{ne^2}{(10\sqrt{n})^2}\right)^{10\sqrt{n}} = \left(\frac{e^2}{10^2}\right)^{10\sqrt{n}}
\end{aligned}$$

Note that $L(\pi) \geq 10\sqrt{n}$ is equivalent to $\bigcup X_i$ (we need at least once increasing subsequence of length $10\sqrt{n}$), so we get what we desired.

(b) Note that $L(\pi)$ is between 1 and n. Hence, we have

$$\begin{aligned}
\mathbb{E}[L(\pi)] &= \sum_{k=1}^{n} k \cdot \mathbb{P}[L(\pi) = k] \\
&= \sum_{k=1}^{10\sqrt{n}-1} k \cdot \mathbb{P}[L(\pi) = k] + \sum_{k=10\sqrt{n}}^{n} k \cdot \mathbb{P}[L(\pi) = k] \\
&\leq \sum_{k=1}^{10\sqrt{n}-1} (10\sqrt{n}) \cdot \mathbb{P}[L(\pi) = k] + \sum_{k=10\sqrt{n}}^{n} n \cdot \mathbb{P}[L(\pi) = k] \\
&= (10\sqrt{n}) \cdot \mathbb{P}[L(\pi) < 10\sqrt{n}] + n \cdot \mathbb{P}[L(\pi) \geq 10\sqrt{n}] \\
&\leq 10\sqrt{n} \cdot 1 + n \cdot \left(\frac{e^2}{10^2}\right)^{10\sqrt{n}} = O(\sqrt{n})
\end{aligned}$$

5. **Dynamic Programming** (Classical Questions). Here are some simpler "quickfire" exercises for DP.

(a) There are N stones with heights $h_1, h_2, \ldots, h_N$ respectively. For each step, the frog can jump from stone i to any of stones $i+1, i+2, \ldots, i+K$ (say, stone j), incurring cost $|h_i - h_j|$. Find the minimum cost for the frog to jump from stone 1 to stone N. Your algorithm should run in $O(NK)$ time.

Sketch. Define $dp(i)$ to be the minimum cost to reach stone i from stone 1. Initially, $dp(1) = 0$. Then,

$$dp(i) = \max_{i-K \leq j \leq i-1} dp(j) + |h_j - h_i|$$

(b) The vacation consists of N days. For day i, you can either choose one of activities A, B, C, incurring happiness a_i, b_i, c_i respectively. However, you cannot pick the same activity for two consecutive days. Find the maximum happiness you can get over the N days. Your algorithm should run in $O(N)$ time.

Sketch. Define $dp(i, j)$ as the maximum happiness you can get in days $1 \ldots i$, if the activity chosen in day i is j. Initially, $dp(0, j) = 0$ for all j. Then,

$$dp(i, A) = \max\{dp(i-1, B), dp(i-1, C)\} + a_i$$

and similarly for $dp(i, B)$ and $dp(i, C)$.

(c) There are N items with weights $w_1, w_2, \ldots, w_N$ and values $v_1, v_2, \ldots, v_N$. Pick a subset of items with maximum total value, subject to the constraint where the total weight of the items must be $\leq W$. Design two algorithms, one solving this in $O(NW)$ and another solving this in $O(NV)$, where $V = \sum_{i=1}^{N} v_i$.

Sketch. Refer to lecture for the $O(NW)$ solution. For the $O(NV)$ solution, define $dp(i, j)$ as the minimum weight required to obtain a value of j in items $1 \ldots i$. Initially, $dp(0, 0) = 0$ and $dp(0, j) = \infty$ for $j > 0$. Then,

$$dp(i, j) = \min\{dp(i-1, j), dp(i-1, j - v_i) + w_i\}$$

The final answer is the maximum j such that $dp(N, j) \leq W$.

(d) You are multiplying N matrices of sizes $m_0 \times m_1$, $m_1 \times m_2$, …, $m_{N-1} \times m_N$ respectively. Multiplying a $x \times y$ matrix with a $y \times z$ matrix takes cost xyz. What is the minimum cost required for you to multiply all these N matrices together to get a single $m_0 \times m_N$ matrix? Your algorithm should run in $O(N^3)$ time.

Sketch. Define $dp(i, j)$ to be the minimum cost of multiplying the matrices $i \ldots j$ into a

single matrix. Initially, $dp(i,i) = 0$. Then,

$$dp(i,j) = \min_{i \le k \le j-1} \{dp(i,k) + dp(k+1,j) + m_{i-1} \times m_k \times m_j\}$$

The final answer is simply $dp(1,N)$. Note that we should calculate $dp(i,j)$ in ascending order of $j-i$.

(e) You are trying to distribute K candies to N children. Child i must receive between 0 and a_i candies (both inclusive). Also, no candies should be left over. Find the number of ways to distribute the candies. Your algorithm should run in $O(NK^2)$ time. (Bonus: Optimize it to $O(NK)$.)

Sketch. Define $dp(i,j)$ as the number of ways to distribute j candies to children $1 \dots i$. Initially $dp(0,0) = 1$. Then

$$dp(i,j) = \sum_{k=0}^{a_i} dp(i-1, j-k)$$

6. **Dynamic Programming** (AY 23/24 Sem 2). Given two strings x and y the edit distance, denoted by $ed(x,y)$, between them is the minimum number of character insertion, deletion and substitution operations (all of these operations are also known as edit operations) required to transform x into y. Given two strings x and y of length n and m respectively, design and analyze an algorithm that computes the value of $ed(x,y)$ in $O(mn)$ time.

Sketch. Define $dp(i,j)$ as the edit distance between $x[1 \dots i]$ and $y[1 \dots j]$. Let's make a few observations about the 2 last characters of x ($x[i]$) and y ($y[j]$) we are examining:

- $dp(0,j) = j$ and $dp(i,0) = i$ because when x or y is empty, we can quickly see that we must delete all characters in the other string to make them equal.
- If $x[i] = y[j]$, then $dp(i,j) = dp(i-1, j-1)$. This is because the last character for both strings are equal. Edit distance of the 2 strings will be equal to the edit distance of the 2 strings without the last character.
- Else, $x[i] \neq y[j]$, then to make them equal, we can
  - Insert $y[j]$ after $x[i]$, thus, $dp(i,j) = 1 + dp(i, j-1)$. (Note that this is equivalent to deleting $y[j]$)
  - Insert $x[i]$ after $y[j]$, thus, $dp(i,j) = 1 + dp(i-1, j)$. (Note that this is equivalent to deleting $x[j]$)
  - Edit $x[i]$ to be $y[j]$, thus, $dp(i,j) = 1 + dp(i-1, j-1)$. (Note that this is equivalent to editing $y[j]$ to be $x[i]$)

From the series of observations, let's build the recursive solution:

$$dp(i,j) = \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ dp(i-1, j-1) & \text{if } x[i] = y[j] \\ 1 + \min\{dp(i, j-1), dp(i-1, j), dp(i-1, j-1)\} & \text{otherwise} \end{cases}$$

7. **Dynamic Programming** (AY 23/24 Sem 2). Suppose a seller wants to cut a rod into several pieces and then sells them to his/her customers. The main objective of the seller is to maximize his/her earning. Given a rod of length n and a list of prices of rod of length i for all $1 \le i \le n$, find an optimal way to cut the rod into smaller pieces in order to maximize the earning. (Try to design a dynamic programming algorithm.)

Sketch. Define $dp(i)$ as the maximum value of a rod of length i. Then,

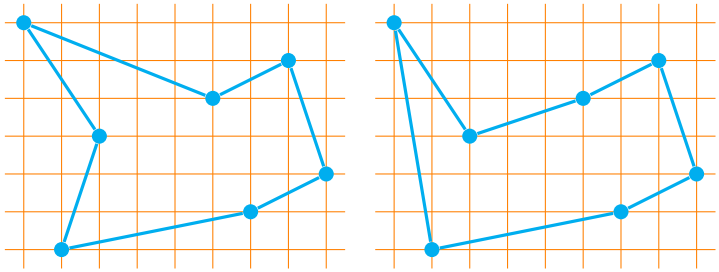
$$dp(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max_{j<i}\{dp(j) + p_{j-i}\} & \text{otherwise} \end{cases}$$

We can compute the states in the order $dp(0), dp(1), \ldots, dp(n)$ with a bottom-up approach. The final answer is $dp(n)$.
There are n distinct states and computing each state takes $O(n)$ time. Therefore, the time complexity is $O(n^2)$.

8. **Dynamic Programming** (CLRS Chapter 14 Problem 14-3). In the euclidean traveling-salesperson problem, you are given a set of n points in the plane, and your goal is to find the shortest closed tour that connects all n points. Figure 1(a) shows the solution to a 7-point problem. The general problem is NP-hard, and its solution is therefore believed to require more than polynomial time.

   J. L. Bentley has suggested simplifying the problem by considering only bitonic tours, that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point. Figure 1(b) shows the shortest bitonic tour of the same 7 points. In this case, a polynomial-time algorithm is possible.

   Describe an $O(n^2)$-time dynamic programming algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. Provide a clear description of your algorithm, including the optimal substructure, and a detailed running time analysis.



(a) **(b)**

Figure 1: Seven points in the plane, shown on a unit grid. **(a)** The shortest closed tour, with length approximately 24.89. This tour is not bitonic. **(b)** The shortest bitonic tour for the same set of points. Its length is approximately 25.58.

Solution:

- **State Definition**: Sort the points according to x-coordinate. Let $dp(i, c_F, c_B)$ be the length of the shortest "tour" to cover the remaining points $p_i, p_{i+1}, \ldots, p_n$, given we already have one forward path from p_1 to p_{c_F} and one backward path from p_{c_B} to p_1 covering the points $p_1, \ldots, p_{i-1}$. The final answer would be $dp(2, 1, 1)$.
- **Transition**: We can either assign p_i to the forward path or the backward path:

$$dp(i, c_F, c_B) = \min \begin{cases} dp(i+1, i, c_B) + d(c_F, i) \\ dp(i+1, c_F, i) + d(c_B, i) \end{cases}$$

  where $d(i, j)$ is the distance between points p_i and p_j.
- **Base Case**: When we have covered all points $p_1, p_2, \ldots, p_n$, it suffices to close the tour. Therefore, $dp(n, c_F, n) = d(c_F, n)$ and $dp(n, n, c_B) = d(n, c_B)$.
- **Time Complexity Analysis**: We evaluate $dp(2, 1, 1)$ by the top-down approach with memoization. Since either $c_F = i$ or $c_B = i$, there are only $O(n^2)$ distinct states and each state takes $O(1)$ time to compute. The total time complexity is $O(n^2)$.