COSC 320: Assignment 3

This assignment is due **Friday, February 28 at 7 PM**. Late submissions will not be accepted. All the submission and formatting rules for Assignment 1 apply to this assignment as well.

1 List of names of group members (as listed on Canvas)

Provide the list here. This is worth 1 mark. Include student numbers as a secondary failsafe if you wish.

1. Rin Meng, 51940633
2. Mika Panagsagan 29679552
3. Kevin Zhang 10811057

2 Statement on collaboration and use of resources

To develop good practices in doing homeworks, citing resources and acknowledging input from others, please complete the following. This question is worth 2 marks.

1. All group members have read and followed the guidelines for groupwork on assignments given in the Syllabus).

   [✓ Yes] No

2. We used the following resources (list books, online sources, etc. that you consulted):

   (a) DeepSeek R1, ChatGPT to generate ideas, explain concepts, and check if we are going in the right direction.

   (b) Google Search, Google's AI Overview

   (c) COSC 320 Lecture Slides

3. One or more of us consulted with course staff during office hours.

   [✓ Yes] No

4. One or more of us collaborated with other COSC 320 students; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   [✓ Yes] No

   If yes, please list their name(s) here:

5. One or more of us collaborated with or consulted others outside of COSC 320; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   Yes [✓ No]

   If yes, please list their name(s) here:

3 More Heat!

You are an air conditioner mechanic, and your normally temperate coastal city is expecting record-breaking heat this summer. You have agreed to install or repair an air conditioning system for n clients, and each job will take one day to complete. You must decide the order in which to complete the jobs.

None of your clients want to be hot and uncomfortable in the coming heat wave, so they would all prefer to have their project completed earlier rather than later. In particular, after n days the uncomfortable heat is expected to be over, so nobody is going to be especially happy with being the last client on your list. More precisely, if you complete the job for client i at the end of day j, then client i's satisfaction is $s_i = n - j$.

Some clients are more important to you than others (perhaps they are paying you more, are more likely to recommend you to others, or are more likely to be repeat customers in the future). You assign each client i a weight w_i. You want to schedule your clients across the n days such that you maximize the weighed sum of satisfaction, that is, $\sum_{1 \le i \le n} w_i s_i$.

In this scenario, an optimal greedy algorithm is to sort the jobs in order of w_i.

1. (3 points) Instead of assuming that each job takes exactly 1 day, suppose that it takes t_i days to complete the job i (here t_i must be greater than 0, but is not necessarily an integer). Assume that $\sum_{1 \le i \le n} t_i = n$. Again, you want to maximize the weighted sum of satisfaction $\sum_{1 \le i \le n} w_i s_i$, where $s_i = n - c_i$. and c_i is the time to completion of the project. For example, if project i is completed first, then $c_i = t_i$. If a project i' is completed second (after project i), then $c_{i'} = t_i + t_{i'}$.

   Give and briefly explain a counterexample to show that the greedy strategy of completing projects in decreasing order of w_i is not optimal for this variant of the problem.

   Consider the following instance given for $n = 4$ and two jobs, j_1 and j_2

Jobs j_i	Weight w_i	Days t_i	Completion c_i
j_1	5	3	3
j_2	4	1	4

   The greedy algorithm will sort the jobs in order of w_i, so it will complete j_1 first and then j_2. The completion times for the jobs are $c_1 = 3$ and $c_2 = 4$, respectively. The satisfaction for the clients is $s_1 = 1$ and $s_2 = 0$, respectively. The weighted sum of satisfaction is $5 \cdot (4 - 3) + 4 \cdot (4 - 4) = 5$.

   Now, consider the optimal solution where we complete j_2 first and then j_1. The completion times for the jobs are $c_2 = 1$ and $c_1 = 4$, respectively. The satisfaction for the clients is $s_2 = 3$ and $s_1 = 0$, respectively. The weighted sum of satisfaction is $4 \cdot (4 - 1) + 5 \cdot (4 - 4) = 12$.

2. (2 points) Describe a greedy strategy that is optimal on the problem variant described in part 1. A one-sentence description is sufficient (e.g., describe the order that you would use to sort the projects).

   The work and time ratio w_i/t_i is the key to the greedy strategy. We will sort the jobs in decreasing order of w_i/t_i.

3. (5 points) Prove that your greedy strategy from part 2 is optimal.

Proof. We want to show that it is optimal by using an exchange argument. Suppose that there is an optimal solution that does not follow the greedy strategy. Let j_i and j_k be two jobs in the optimal solution such that j_i is completed before j_k but j_k has a higher work and time ratio than j_i.

Let's consider the optimal schedule S with jobs $j_1, j_2, \ldots, j_n$ in that order, and suppose that there are two adjacent jobs j_i and j_{i+1} such that $\frac{w_i}{t_i} < \frac{w_{i+1}}{t_{i+1}}$.

Let T be the sum of times of all jobs scheduled before j_i. In schedule S, job j_i completes at time $c_i = T + t_i$ and job j_{i+1} completes at time $c_{i+1} = T + t_i + t_{i+1}$.

The contribution of these two jobs to the total weighted satisfaction is: $w_i(n - (T + t_i)) + w_{i+1}(n - (T + t_i + t_{i+1}))$

Now consider schedule S' obtained by swapping j_i and j_{i+1}. In S', job j_{i+1} completes at time $c'_{i+1} = T + t_{i+1}$ and job j_i completes at time $c'_i = T + t_{i+1} + t_i$.

The contribution of these two jobs to the total weighted satisfaction in S' is: $w_{i+1}(n - (T + t_{i+1})) + w_i(n - (T + t_{i+1} + t_i))$

The difference in weighted satisfaction between S' and S is:

$$[w_{i+1}(n - T - t_{i+1}) + w_i(n - T - t_{i+1} - t_i)] - [w_i(n - T - t_i) + w_{i+1}(n - T - t_i - t_{i+1})] \quad (1)$$
$$= w_{i+1}(n - T - t_{i+1}) - w_{i+1}(n - T - t_i - t_{i+1}) + w_i(n - T - t_{i+1} - t_i) - w_i(n - T - t_i) \quad (2)$$
$$= w_{i+1}t_i - w_i t_{i+1} \quad (3)$$

Since $\frac{w_{i+1}}{t_{i+1}} > \frac{w_i}{t_i}$, we have $w_{i+1}t_i > w_i t_{i+1}$, which means the difference is positive.

Therefore, swapping j_i and j_{i+1} increases the weighted satisfaction. This contradicts the optimality of the original solution S. □

4 Weekly Meeting Logistics

You're the manager of an animal shelter, which is run by a few full-time staff members and a group of n volunteers. Each of the volunteers is scheduled to work one shift during the week. There are different jobs associated with these shifts (such as caring for the animals, interacting with visitors to the shelter, handling administrative tasks, etc.), but each shift is a single contiguous interval of time. Shifts cannot span more than one week (e.g., we cannot have a shift from 10 PM Saturday to 6 AM Sunday). There can be multiple shifts going on at once.

You'd like to arrange a weekly meeting with your staff and volunteers, but you have too many volunteers to be able to find a meeting time that works for everyone. Instead, you'd like to identify a suitable subset of volunteers to instead attend the weekly meeting. You'd like for every volunteer to have a shift that overlaps (at least partially) with a volunteer who is attending the meeting. Your thinking is that you can't personally meet with every single volunteer, but you would like to at least meet with people who have been working with every volunteer (and may be able to let you know if a volunteer is disgruntled or having any difficulties with their performance, etc.). Because your volunteers are busy people, you want to accomplish this with the fewest possible volunteers.

Your volunteer shifts are given as an input list V, and each volunteer shift v_i is defined by a start and finish time (s_i, f_i). Your goal is to choose a subset of volunteer shifts of minimum size, such that every shift in V overlaps with at least one of the chosen shifts. A shift $v_i = (1, 4)$ overlaps with the shift $v_j = (3, 5)$ but not with the shift $v_k = (4, 6)$.

1. (2 points) Your co-worker proposes the following greedy algorithm to select volunteers for your meeting:

   Select the shift v that overlaps with the most other shifts, discard all shifts that overlap with v, and recurse on the remaining shifts.

Give and briefly explain a counterexample to prove that this greedy strategy is not optimal.

Consider the following instance with 5 shifts:

Shift	Start Time	End Time	Overlaps with
v_1	1	6	v_2, v_3, v_4
v_2	2	3	v_1
v_3	3	4	v_1
v_4	4	5	v_1, v_5
v_5	5	7	v_4

The greedy algorithm will select v_1 first since it overlaps with the most shifts (3 shifts). After removing v_1, v_2, v_3, v_4, we are left with just v_5, which we also have to select. So the greedy algorithm selects 2 shifts: v_1 and v_5.

However, the optimal solution is to select just v_4, which overlaps with v_1 and v_5, and then v_1 overlaps with v_2 and v_3. So one shift (v_4) covers all shifts.

2. (3 points) Give a greedy algorithm to solve this problem. Give an unambiguous specification of your algorithm using a **brief, plain English description**. Do not write pseudocode or worry about implementation details yet. (You may do this in part 3 if you feel that it's necessary to achieve a particular runtime.)

   Sort all volunteer shifts by their end times in ascending order. Do this:

   1. Start with an empty set of selected shifts.
   2. Create a set of "covered" shifts (initially empty).
   3. While there are uncovered shifts remaining:
      (a) Find the earliest-ending uncovered shift s.
      (b) Among all shifts that overlap with s (including s itself), select the shift t that overlaps with the most uncovered shifts.
      (c) Add t to our set of selected shifts.
      (d) Mark all shifts that overlap with t as "covered".
   4. Return the set of selected shifts.

   This approach prioritizes selecting shifts that can efficiently cover the remaining uncovered shifts, starting with the earliest-ending shifts to ensure we don't miss any potential overlaps.

3. (4 points) Briefly justify a good asymptotic bound on the runtime of your algorithm. If you prefer to present pseudo-code to help track the runtime incurred, you may do so.

   Let's analyze the runtime of our algorithm:

   1. Sorting all volunteer shifts by end times takes $O(n \log n)$ time, where n is the number of shifts.
   2. For each uncovered shift, we need to:
      - Find the earliest-ending uncovered shift, which takes $O(n)$ time in the worst case.
      - For this shift, we check all other shifts to find overlaps, which takes $O(n)$ time.
      - Among overlapping shifts, we count how many uncovered shifts each overlaps with, which takes $O(n^2)$ time in the worst case, as we may need to check each shift against all other shifts.
      - Marking shifts as covered takes $O(n)$ time.
   3. In the worst case, we might select only one shift per iteration, requiring $O(n)$ iterations.

   Combining these steps, the overall runtime is: $O(n \log n) + O(n) \cdot (O(n) + O(n) + O(n^2) + O(n)) = O(n \log n) + O(n^3) = O(n^3)$.

5 Fun with Recurrences

Give an asymptotic solution (which should be a Θ-bound) to each of the recurrences below. You may use whatever solution method you wish (drawing out the tree, unrolling the recurrence, proof by induction, Master Theorem, etc.), but make sure you fully justify your answer.

1. (3 points) $T(n) = 6T\left(\frac{n}{2}\right) + 2^n$ for $n > 1$, $T(1) = 1$.

   We will use the Master Theorem to solve this recurrence. The recurrence is of the form

   $$T(n) = aT\left(\frac{n}{b}\right) + f(n), \text{ where } a = 6,\ b = 2, \text{ and } f(n) = 2^n$$

   First, we compute

   $$n^{\log_b a} = n^{\log_2 6} \approx n^{2.58}$$

   Since $f(n) = 2^n$ grows faster than any polynomial, we have

   $$f(n) = \Omega(n^{\log_b a + \epsilon})$$

   for any constant $\epsilon > 0$.

   For Case 3 of the Master Theorem to apply, we also need to verify the regularity condition: $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and sufficiently large n.

   Checking this condition:

   $$af\left(\frac{n}{b}\right) = 6 \cdot 2^{n/2} = 6 \cdot 2^{-n/2} \cdot 2^n$$

   Since $6 \cdot 2^{-n/2} < 1$ for sufficiently large n, the regularity condition is satisfied.

   Therefore, by Case 3 of the Master Theorem, $T(n) = \Theta(f(n)) = \Theta(2^n)$.

2. (3 points) $T(n) = T\left(\frac{n}{4}\right) + 2T\left(\frac{n}{16}\right) + \sqrt{n}$ for $n > 16$, $T(n) = 1$ for $n \leq 16$.

   We expand the recurrence tree by repeatedly substituting the terms:

   $$T(n) = T\left(\frac{n}{4}\right) + 2T\left(\frac{n}{16}\right) + \sqrt{n}.$$

   Expanding further:

   $$T\left(\frac{n}{4}\right) = T\left(\frac{n}{16}\right) + 2T\left(\frac{n}{64}\right) + \sqrt{\frac{n}{4}},$$

   $$T\left(\frac{n}{16}\right) = T\left(\frac{n}{64}\right) + 2T\left(\frac{n}{256}\right) + \sqrt{\frac{n}{16}}.$$

   Substituting back, we get:

   $$T(n) = T\left(\frac{n}{16}\right) + 2T\left(\frac{n}{64}\right) + \sqrt{\frac{n}{4}} + 2T\left(\frac{n}{16}\right) + 4T\left(\frac{n}{64}\right) + 2\sqrt{\frac{n}{16}} + \sqrt{n}.$$

Observing the pattern, the recursive calls keep breaking down, and the sum of all extra terms is:

$$\sqrt{n} + \sqrt{\frac{n}{4}} + 2\sqrt{\frac{n}{16}} + 4\sqrt{\frac{n}{64}} + \dots$$

which forms a geometric series. The series follows:

$$S = \sqrt{n}\left(1 + \frac{1}{2} + \frac{2}{4} + \frac{4}{8} + \dots\right).$$

Observing the growth, this geometric series converges to $O(\sqrt{n})$. Since the recursion depth is logarithmic, and at each level, the dominant cost is $O(\sqrt{n})$, the final complexity is:

$$T(n) = \Theta(\sqrt{n}).$$

3. (4 points) $T(n) = mT(n-1) + 1$ for $n > 1$, $T(1) = 1$. Assume that m is an integer value greater than or equal to 2.

We will use the iterative/rollout method to solve this recurrence. We have

$$\begin{aligned}
&T(n) = mT(n-1) + 1 \\
&= m[mT(n-2) + 1] + 1 \\
&= m^2T(n-2) + m + 1 \\
&= m^2[mT(n-3) + 1] + m + 1 \\
&= m^3T(n-3) + m^2 + m + 1 \\
&= \dots \\
&= m^{n-1}T(1) + m^{n-2} + m^{n-3} + \dots + m + 1 \\
&= m^{n-1} + m^{n-2} + m^{n-3} + \dots + m + 1 \\
&= \frac{m^n - 1}{m - 1} \\
&= \Theta(m^n)
\end{aligned}$$

4. (4 points) $T(n) = mT(\frac{n}{4}) + n^2$ for $n \geq 4$; $T(n) = 1$ for $n < 4$. Assume that m is greater than or equal to 1 (but not necessarily an integer).

We will use the Master Theorem to solve this recurrence. The recurrence is of the form $T(n) = mT\left(\frac{n}{b}\right) + f(n)$, where $m = m$, $b = 4$, and $f(n) = n^2$.

First, we compute

$$n^{\log_b m} = n^{\log_4 m}.$$

Since $f(n) = n^2$ is a polynomial, we have $f(n) = \Theta(n^{\log_b m})$.

For Case 1 of the Master Theorem to apply, we also need to verify the regularity condition:

$$af\left(\frac{n}{b}\right) \leq cf(n)$$

for some constant $c < 1$ and sufficiently large n.

Checking this condition:

$$af\left(\frac{n}{b}\right) = m \cdot \left(\frac{n}{4}\right)^2 = \frac{m}{16}n^2$$

Since $\frac{m}{16} < 1$ for sufficiently large n, the regularity condition is satisfied.

Therefore, by Case 1 of the Master Theorem, $T(n) = \Theta(n^{\log_b m}) = \Theta(n^{\log_4 m})$.

6 D+C = Profit (Bonus 5%)

You own an online sales company called DCAuctions.com that sells goods both on auction and on a fixed-price basis. You want to use historical auction data to investigate your fixed price choices.

Over n minutes, you have a good's price in each minute of the auction. You want to find the largest *price-over-time stretch* in your data. That is, given an array A of n price points, you want to find the largest possible value of

$$f(i, d) = d \cdot \min(A[i], A[i+1], \ldots, A[i+d-1]),$$

where i is the index of the left end of a stretch of minutes, d is the duration (number of minutes) of the stretch, and the function f computes the duration times the minimum price over that period. (Prices are positive, $d \geq 0$, and for all values of i, $f(i, 0) = 0$ and $f(i, 1) = A[i]$.)

For example, the best stretch is underlined in the following price array: $[8, 2, \underline{9, 5, 6, 5}, 3, 1]$. Using 1-based indexing, the value for this optimal stretch starting at index 3 and running for 4 minutes is $f(3, 4) = 4 \cdot \min(9, 5, 6, 5) = 4 \cdot 5 = 20$.

1. (3 points) Describe a polynomial-time brute force algorithm to solve this problem.

2. (3 points) Write a function `MidHelper` (consider the name to be a hint for the next question!) that, given an array A, an index $1 \leq i \leq n-1$ and length $k \leq n$, finds the best stretch of length less than or equal to k that includes both $A[i]$ and $A[i+1]$. (Another hint for the next question: you will want to do this in $O(k)$ time.)

3. (5 points) Write a divide and conquer algorithm algorithm to efficiently find the best price stretch. Your algorithm should use the `MidHelper` function described in part 2.

4. (2 points) Give and briefly justify a good asymptotic bound on the runtime of your algorithm.