Second Midterm Exam CS413

- Due Apr 15 at 11:59pm
- Points 60
- Questions 8
- Available Apr 14 at 8am - Apr 15 at 11:59pm
- Time Limit None

Instructions

Please answer the questions carefully and precisely, ensuring you address every part of what is asked. Follow the instructions in each question and respond to **all required details. Always look for the most efficient algorithm /the tightest bound.**

Important: The exam allows **only one attempt**. You may work at your own pace, navigate between questions using the "Back" and "Next" buttons, and make edits as needed. Your answers will be saved **automatically**. **Do not click "Submit" until the exam is completely finished and you are ready to submit it.**

Once you submit, no exceptions will be made. Be sure you are fully ready before finalizing your exam.

Similar to the first midterm exam, as a reminder, your responses **will be automatically saved** (as shown in the screenshot below), allowing you to review and complete any remaining questions before submission.

sample-1.jpg

You have no time limit (as long as you submit before the deadline, Tuesday, 04/15/25, BEFORE 11:59 pm). Please note that the link to take the exam disappears sharp at 11:59 pm on Tuesday.

ONLY C++/Java coding is accepted. No points for other programming languages.

As it appears in the course syllabus, students should not discuss the questions with others and are not allowed to get help from the internet, friends, or any other resources to answer the questions. You are expected to turn in the results of your own effort (not the results of a friend's efforts). Even when not explicitly asked, you are supposed to justify your answers concisely.

No makeup exam option is available.

No deadline extension option is available.

The exam is individual.

This quiz was locked Apr 15 at 11:59pm.

ⓘ Correct answers are no longer available.

Score for this quiz: 55 out of 60
Submitted Apr 15 at 11:49pm
This attempt took 103 minutes.

Question 1
10 / 10 pts

We have designed a divide-and-conquer algorithm that runs on an input of size n. This algorithm works by spending O(n) time splitting the problem in half, then does a recursive call on each half, then spends O(1) time combining the solutions to the recursive calls. On small inputs, the algorithm takes a constant amount of time.

We want to see how long this algorithm takes, in terms of n, to perform the task.

(a) First, write a recurrence relation that corresponds to the time complexity of the above divide-and-conquer algorithm.

(b) Then solve the relation to come up with the worst-case time taken for the algorithm. Show all your work and explain.

Your Answer:

a.) Recurrence relation: **T(n) = 2*T(n/2) + O(n)**

b.) As per the Master Theorem, T(n) = a*T(n/b) + f(n)
here, a=2, b=2 and f(n) = n
where, $\log_b a = \log_2 2 = 1$
Therefore, comparing f(n) with $n^{\log_b a}$, so we get $f(n) = n^{\log_b a} = n$
Applying Case 2 of Master Theorem,
$f(n) = n^{\log_b a}\log(n)$ which is (nlogn).
Hence, the worst case time complexity is **O(n*logn).**

Question 2
10 / 15 pts

A) You are given a **connected, undirected graph G(V, E)**, where |V|=n represents the vertices and E represents the edges, each having **unique edge weights** (no two edges share the same weight). You

are asked to construct a **subgraph** of the graph that:

- Contains exactly **n nodes** (the same number of vertices as the original graph),
- Is **connected**, and
- Contains **exactly one cycle**.

Your goal is to minimize the **total weight** of this subgraph. **Total weight** is defined as the sum of the weights of all edges included in the subgraph. In this case, the subgraph must include **exactly one cycle**, and your goal is to select edges in a way that results in the smallest possible total weight while meeting the cycle constraint.

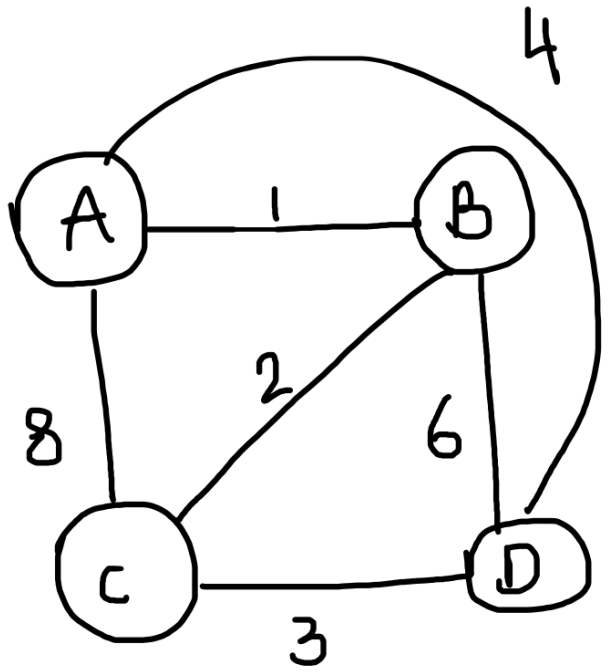
A) Write an **algorithm** to solve the given problem optimally. You may write the algorithm in **pseudocode** or provide a clear **English description** outlining the steps that your algorithm follows. Your explanation should be **clear and precise**, as incomplete or ambiguous algorithms will not earn points. Be sure to remember the key properties of an algorithm and follow them as you outline your solution.

B) For the graph provided below,

(i) Describe **how your algorithm in part A works**. Explain step by step.

(ii) What is the total weight of the obtained subgraph?

(iii) Draw the **final subgraph** and attach its **screenshot** to your answer. Make sure the **cycle** is visible in the subgraph.



Your Answer:

A) Step 1) **Find a Minimum Spanning Tree (MST)**: Utilize a standard MST algorithm, such as

Kruskal's algorithm, to select n−1 edges that connect all n vertices with the minimum total weight. Let T be this set of edges (so T is a spanning tree of G).

Step 2) **Select the Lightest Extra Edge:** Examine all remaining edges $e \in E$ (edges not already in the MST). Choose the edge e with the smallest weight among those. (Because T is spanning, any such $e*$ will connect two vertices already connected by T, thus introducing a cycle.)

Step 3) **Add the Edge to Form One Cycle:** Add e* to the tree T. The resulting edge set $T' = T \cup \{e*\}$ is our one-cycle sub-graph. By construction, T′ is connected and has exactly one cycle.

Step 4) **Return T′:** This sub-graph T′ is the solution – it has exactly one cycle and minimum possible total weight.

Pseudo Code:

```
function MinSubgraph(G):

    # Step 1: Construct MST (using Kruskal's algorithm)
    T = MST(G)  # T is the set of edges in the MST

    # Step 2: Find the lightest edge outside of the MST
    best_edge = null
    best_weight = ∞
    for each edge e in E:
        if e not in T and weight(e) < best_weight:
            best_edge = e
            best_weight = weight(e)

    # Step 3: Add that edge to T
    T' = T ∪ { best_edge }

    return T'
```

B) i) Construct the Minimum Spanning Tree (MST) using Kruskal's algorithm. The MST will be a sub-graph of the original graph G that contains all vertices with the minimum total weight and no cycles. In other words, it will have exactly |V| - 1 edges.

Next, find the lightest edge that is not in the MST. Iterate through each edge e in the original graph G. If e is not in the MST, check its weight. Keep track of the smallest weight edge among all edges that are not in the MST. Let's call this edge best_edge.

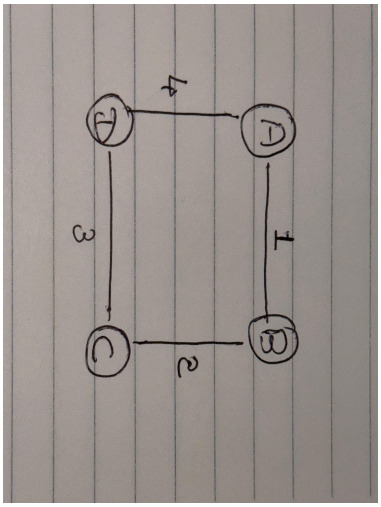
Finally, add the chosen edge to the MST. Adding this single edge to the MST will create exactly one

cycle, since the MST plus one edge equals a tree plus one edge, which equals exactly one cycle. The total weight of the resulting subgraph T' is the sum of the weight of the original tree T and the weight of the best_edge.

Therefore, T' is the final subgraph.

ii) Total weight of the obtained subgraph is **10.**

iii) Final Sub-graph:



Question 3
10 / 10 pts

You are given a set of **n** intervals $\mathbf{I=\{(s_1, f_1),(s_2, f_2),\ldots,(s_n, f_n)\}}$, where each interval $\mathbf{(s_i, f_i)}$ has a start time $\mathbf{s_i}$ and a finish time $\mathbf{f_i}$. The objective is to design an algorithm that finds the largest subset $\mathbf{S \subseteq I}$ such that for any two intervals $\mathbf{(s_i, f_i)}$ and $\mathbf{(s_j, f_j)}$ in **S**, the intervals do not overlap. That is, $\mathbf{f_i \leq s_j}$ or $\mathbf{f_j \leq s_i}$.

To solve this problem optimally, we propose the following greedy strategy: First, we sort the intervals in ascending order based on their start times, creating a new ordered set **M**. Next, we iterate through the intervals in **M**, selecting each interval if it does not overlap with any of the intervals already added to set **S**. By following this approach, we ensure that the intervals chosen for **S** are non-overlapping, and we maximize the size of **S**. Is this strategy solving our defined problem optimally? If yes, prove your claim. If not, provide a counter-example.

Your Answer:

No, Given approach(Sorting by starting time) is not optimal. The interval-scheduling problem should be solved by sorting the intervals based on their finish times, not their starting times. Sorting by starting times does not necessarily optimize the number of intervals that can fit into the schedule. Early-starting intervals may occupy a large portion of the available time, causing the algorithm to miss smaller intervals that could collectively form a larger solution set.

Counter Example:

Consider the following set of intervals:

1. I1 = {1, 8}

2. I2 = {2, 4}

3. I3 = {5, 6}

Sorting by Starting Times: If we sort the intervals in ascending order, we obtain: {1, 8}, {2, 4}, {5, 6}.

An algorithm that selects intervals in this order, ensuring that no intervals overlap with previously chosen intervals, yields the following result:

Pick I1 = {1, 8} => S = ({1, 8})

Selecting I2 and I3 will result in an overlap with the previously chosen interval I1.

Therefore, **the optimal solution is S = ({2, 4}, {5, 6}), with a size of 2**, whereas the current solution has a size of 1.

Question 4
5 / 5 pts

True/False? The merge sort algorithm solves the sorting problem asymptotically slower than the quick sort algorithm in the worst-case and as n grows. No justification is needed.

- ○ True
- ◉ False

Question 5
5 / 5 pts

Let f(n)= $n^3 \log n + 12n^4 + 25\ n$. Which one of the following cases is correct regarding f(n)?

- (●) $f(n)=\Omega(n^4)$
- ( ) $f(n)=\Theta(n^4 \log n)$
- ( ) None of the answers are correct
- ( ) $f(n)=O(n^3 \log n)$
- ( ) More than one answer is correct.
- ( ) $f(n)=\Theta(n)$
- ( ) $f(n)=\Omega(n^4 \log n)$
- ( ) $f(n)=O(n)$

Question 6
5 / 5 pts

Apply the Master Theorem to bound on the following recurrence relation using Big-O notation. Assume T(n)=1.

$T(n)= 25T(n/5) + 3n^2$

- ( ) $O(n)$
- (●) $O(n^2 \log n)$
- ( ) $O(n \log^2 n)$
- ( ) $O(1)$
- ( ) $O(n^5 \log n)$
- ( ) $O(n \log n)$
- ( ) None of the cases are correct.
- ( ) $O(n^5)$
- ( ) $O(\log^2 n)$

Question 7
5 / 5 pts

Which of the following functions grows the fastest as n grows?

- n^{400}
- n log n
- n log n^{100}
- n^3 log n
- 4^{1000} n
- 2^n

Question 8
5 / 5 pts

How many times will the operation inside the while loop be executed in the worst case? n is a positive number.

```
void fun (int n)
{
    int counter = 1;
    while (counter <n)
        counter = counter*4;
 }
```

- not countable
- O(counter)
- O(log n)
- Depends on the machine
- O(1)
- None of the cases are correct.
- O(n)

Quiz Score: 55 out of 60