1 Q1 (30 pts): Short questions

Circle the correct answer. You do not have to show your work, but it may earn you partial credit even if your answer is wrong.

1. (3 pts) Suppose algorithm FAST runs in $\Theta(n^2)$ time for all inputs of size $n$, and algorithm SLOW runs in $\Theta(2^n)$ time for all inputs of size $n$. FAST is faster than SLOW:

(a) Always  
(b) Never  
(c) On a finite number of inputs  
(d) On all but at most a finite number of inputs  
(e) None of the above is necessarily true

2. (3 pts) What is the worst-case time to search for an item in a (not necessarily balanced) binary search tree with $n$ nodes?

(a) $\Theta(1)$  
(b) $\Theta(\log(n))$  
(c) $\Theta(n)$  
(d) $\Theta(n \log(n))$  
(e) $\Theta(n!)$

3. (4 pts) The recursion $T(n) = 1$ for any $n \leq 10$ and $T(n) = n^3 + n^{2.93} + 8T(n/2)$ solves to:

(a) $\Theta(n^3)$  
(b) $\Theta(n^3 \log n)$  
(c) $\Theta(n^3 (\log n)^{2.93})$  
(d) $\Theta(n^{2.93})$  
(e) $\Theta(n^{2.93+3})$
4. (7 pts) Suppose you have a counter, represented by \( n \) binary bits. The cost of incrementation or decrementation is the number of bits you need to flip in order to represent the new number. What is the worst-case amortized cost per operation we pay for a series of increment-operations and decrement-operations? Remember, \( n \) is the number of binary bits, not the number of operations.

(a) \( \Theta(1) \)  (b) \( \Theta(\sqrt{n}) \)  (c) \( \Theta(n) \)  (d) \( \Theta(2^n) \)  (e) 0

5. (7 pts) Suppose you have a data structure that supports two operations: Insert\((x)\) and Remove\(_\text{min}\). Given that the data-structure contains \( n \) elements, Insert adds a new element to the data structure in time \( T_n \), and Remove\(_\text{min}\) removes (and returns) the smallest element in time \( O(\sqrt{\log(n)}) \).
What is the best possible value of \( T_n \)?

(a) \( O(1) \)  (b) \( O(\sqrt{\log(n)}) \)  (c) \( O(\log(n)) \)  (d) \( O((\log(n))^{3/2}) \)  (e) \( O(n) \)

6. (6 pts) Answer TRUE or FALSE with respect to the following directed graph:

(a) _____ The following is a possible DFS tour: \( (g, c, b, h, a, d, f, e) \).
(b) _____ The following is a possible BFS tour: \( (e, c, a, d, b, f, g, h) \).
2 Q2 (25 pts): Dynamic Programming

Consider a row of $n$ numbers $a_1, ..., a_n$. The numbers are all positive, and $n$ is even. We play a game against an opponent, alternating turns. In each turn, a player selects either the first or last number, removes it from the row, and collects that much money.

For example, if the numbers are 1,2,3,4, a possible game may look like:

1. Player A takes 4: row is 1,2,3.
2. Player B takes 3: row is 1,2.
3. Player A takes 1: row is 2.
4. Player B takes 2, game over.

Both players ended with a total amount of 5. Note that A could have done better.

1. (2 pts) Show an example where the greedy algorithm fails to achieve the highest possible amount of money.
2. We now examine the game starting from 1,2,10,3.
Below is a 4-level binary tree depicting all possible outcomes of the game. Each node corresponds to the current state of the row (so the root is labeled 1,2,10,3). Edges are labeled with the player’s action (taking the leftmost or the rightmost number). The levels are alternating – first and third levels (round nodes) represent your actions, second and fourth (square nodes) represent your opponent’s.

- (2 pts) Fill in the labels for edges and nodes for the rest of the tree.
- (2 pts) Under each leaf, write down the amount of money you get at the end of the game. This can be determined from the labels of the edges on the path from the root to the leaf.
- (2 pts) Work your way up the tree, starting from the leaves. Inside each node, write down the amount of money you are guaranteed to be able to collect from this point until the end of the game (no matter what your opponent does in the next turns). Note that some levels represent your action, and some represent your opponent’s!

• (2 pts) How would you play if you went first on the numbers 1,2,10,3?
3. (15 pts) Give an $O(n^2)$ algorithm to determine the maximum possible amount of money we can definitely win if we move first (no matter what the opponent does). If it is a dynamic programming algorithm, please specify (and explain) the base cases, the recursive formula, and the order in which the algorithm solves the sub-problems.
Hint: Your solution would probably include both min and max.
Hint 2: Try identifying common subproblems in the tree you drew in part 1.

4. (5 pts) **Extra credit** (ONLY IF YOU HAVE TIME): Give an $O(n)$ algorithm that guarantees you will win at least as much money than the opponent if you go first (not necessarily the maximum possible amount, just make sure you never lose).
3 Q3 (25 pts): Hashing

Let $H$ be a class of hash functions from universe $U$ of keys to $\{0, 1, ..., m - 1\}$.

1. (2 pts) Define a universal hash family.

2. (15 pts) Suppose that an adversary knows the universal hash family $H$ which you use. The adversary wants to increase the chances of a collision. We choose a hash function $h$ randomly from $H$, keeping it a secret. The adversary then chooses a key $x$ and learns the value $h(x)$. Can the adversary now make collisions more probable? In other words, can they find a $y$ s.t. $y \neq x$ and $h(x) = h(y)$ with probability greater than $1/m$?

If so, write down a particular universal hash family and describe what an adversary would do. If not, prove that the adversary cannot do this.
3. (8 pts) We say that $H$ is **2-universal** if, for every fixed pair of keys $x \neq y$, and for any $h$ chosen uniformly at random from $H$, the pair $(h(x), h(y))$ is equally likely to be any of the $m^2$ possible pairs. Show that, if $H$ is 2-universal, then it is universal.

4. (5 pts) **Extra Credit:** Answer part (1), for the case $H$ is 2-universal.
4 Q4 (20 pts): Graph Algorithms

This question refers to weighted, undirected and connected graphs only. You may assume all weights are positive.

1. (10 pts) Give an algorithm to compute MST in a graph where all edges have equal weight. Analyze your algorithm’s complexity. (Your answer should be more efficient than Prim’s Algorithm.)

2. (10 pts) One of the Towers-of-Hanoi monks got bored one day, and started drawing a Shortest Path Tree (using Dijkstra’s algorithm) on a graph of 10,000 nodes. Alas, after adding the 6626 edge to the tree, a fire alarm went off, and he had to evacuate the building.

You enter the room and see the original weighted graph. You can identify the starting node $s$ and the set of edges already added to the tree, but nothing else (the monk took all intermediate calculations with him). Can you pick up from where he left off? If so, describe how. Otherwise, describe what is missing.