Final Exam Solutions

Problem 1. [2 points] Write your **full name** on every question page!

Problem 2. True/False [30 points] (10 parts)

Circle (**T**)rue or (**F**)alse, and give a *brief* justification of your answer.

(a) T F [3 points] True or false: if $f_1(n) = O(f_2(n))$ and $f_2(n) = O(g(n))$, then $f_1(n) + f_2(n) = O(g(n))$

Solution: True.
$f_1(n) \leq c_1 f_2(n) \leq c_1 c_2 g(n)$ (1)
$f_2(n) \leq c_2 g(n)$ (2)
By (1) and (2), $f_1(n) + f_2(n) \leq (c_1 c_2 + c_2) g(n) = O(g(n))$

(b) T F [3 points] Let $A = [a_1, \ldots, a_n]$ be a one-dimensional array of integers. Define a *mega-peak* in A to be an element $a_i \in A$ such that $a_i \geq a_j$ for all a_j with $|j - i| \leq 2$. True or false: It is possible to find a mega-peak in A in $O(\log n)$ time.

Solution: True.
Run the original peak-finding algorithm from class. Instead of testing if the middle element is a peak when doing divide-and-conquer, check if the middle 3 elements are mega-peaks. If not, recurse on the side containing the largest element among the middle 5 elements.

(c) T F [3 points] True or false: Using radix sort, it is possible to sort n integers in $\{1, \ldots, n^k\}$ in only $O(kn)$ time.

Solution: True.
Represent each integer as a k-digit number in base n, and the running time is $O(n + kn) = O(kn)$.

(d) T F [3 points] In double hashing, we choose functions h_1 and h_2, and use the hash function $h(k,i) = h_1(k) + i * h_2(k) \pmod{m}$, where m is prime. In class, we studied the case where $h_2(k) \in \{1, \ldots, m-1\}$ for all k. True or false: if $h_2(k) = 0$ for some key k, then inserting k into the hash table will always work correctly (though it may be slow).

Solution: False.
If $h_2(k) = 0$ then the probe sequence consists of a single integer repeated, so we may not be able to find an empty slot, when attempting to insert an element.

(e) T F [3 points] Karatsuba's algorithm for multiplying two n-digit numbers runs in time $O(3^{\log_2(n)})$.

Solution: True.
$O(3^{\log_2(n)}) = O(n^{\log_2(3)})$, which is the running time.

(f) T F [3 points] Suppose Newton's Method is used to solve an equation of the form $f(x) = 0$, where f is a polynomial with given integer coefficients. Suppose the algorithm yields the sequence $[x_0, x_1, \ldots]$ of successive guesses, and suppose that for the initial guess x_0 we have $f(x_0) \neq 0$. Then, true or false: $f(x_i) \neq 0$ for all $i > 0$.

Solution: False.
It's entirely possible that a given tangent line collides with the x-axis at $x = 0$ at any time during the procedure.

(g) T F [3 points] When running DFS on a graph $G = (V, E)$, each vertex $v \in V$ is assigned a start time $v.d$ and a finishing time $v.f$. Let v and w be distinct vertices in V. True or false: it is impossible to have $v.d < w.d < v.f < w.f$.

Solution: True.
This is the parenthesization lemma (see CLRS).

(h) T F [3 points] Let G be a DAG with positive edge weights. Let x, y and z be distinct vertices in G such that y is on a shortest path from x to z, and suppose there are multiple shortest paths from x to y. Then, true or false: there are multiple shortest paths from x to z.

Solution: True.
Let P_1 and P_2 be two shortest paths from x to y, and let P_3 be a shortest path from y to z. Then P_1P_3 and P_2P_3 must both have the same length and hence are both distinct shortest paths from x to z.

(i) T F [3 points] True or false: in the Floyd-Warshall algorithm, we obtain an $O(V^3)$ algorithm for the all-pairs shortest paths problem by considering the subproblem "what is the shortest path from v_i to v_j that uses at most k edges?"

Solution: False.
The correct subproblem description is "what is the shortest path from v_i to v_j that uses intermediate vertices in $\{v_1, \ldots, v_k\}$"

(j) T F [3 points] Let G be an undirected graph with arbitrary (possibly negative) edge weights. Assume P $\neq$ NP. True or false: the decision problem of determining whether G has a negative-weight cycle is NP-hard.

Solution: False.
The Bellman-Ford algorithm detects negative cycles in polynomial time, so this problem is not NP-hard assuming P $\neq$ NP.

Problem 3. Short Answer [20 points] (4 parts)

Answer each problem. Justify your answer using at most two sentences.

(a) [5 points] Define a SUPER-HEAP to be a hypothetical data structure with the following properties:

- It supports $O(n)$ initialization when given an unsorted array of n numerical keys.
- It supports `EXTRACT-MIN` in $O(1)$ time.
- It is *comparison-based*—it only accesses the keys when performing key comparisons, and does no other operations on them.

Explain why it is impossible for a SUPER-HEAP to exist.

Solution: It is impossible to sort in $o(n \log n)$ time using a comparison-based algorithm. But if heapsort is done with a SUPER-HEAP (by constructing the SUPER-HEAP and then repeatedly removing the minimum element n times), then we would have an $O(n)$ comparison-based sorting algorithm, a contradiction.

(b) [5 points] Consider a hash table T of size m that employs a uniform hash function, and uses chaining for collision resolution. T is initially empty. Suppose that four distinct keys are inserted sequentially into T. What is the probability that there is a chain of size exactly 3?

Solution: $\binom{4}{3}$ possible chains of size 3. The first element can go anywhere in the table. Then, two of the other elements have to go to the same slot, which happens with probability $\frac{1}{m}$ for each element. Now, the fourth element has to go to a different slot, which happens with probability $\frac{m-1}{m}$.

The probability is: $\frac{4(m-1)}{m^3}$

(c) [5 points] Let G be an undirected graph with positive edge weights, and let s and t be two distinct vertices in G. Let d be the total number of edges that are relaxed when finding a shortest path from s to t using Dijkstra's algorithm. Let b be the total number of edges that are relaxed when finding a shortest path from s to t using the *bidirectional Dijkstra* algorithm.

Is it always true that $b \leq d$ (regardless of the choice of G, s, or t)? If so, give a short explanation. If not, provide a counterexample.

Solution: It is not true. Consider the example where G is a path $[s, u, v, t, w]$ on five vertices.

(d) [5 points] Peter Looper claims to have found a reduction f that runs in polynomial-time and transforms instances x of the Halting Problem (does a given program $\mathcal{P}$ halt?) to instances $f(x)$ of the CLIQUE Problem (does a given graph G contain a clique of a given size k?). He claims to have constructed f such that x is a YES instance (i.e. the program $\mathcal{P}$ halts) if and only if $f(x)$ is a YES instance (i.e. G has a clique of size k).

Does such a reduction f exist? If so, explain why we know f exists. If not, explain why f cannot exist.

Solution: f cannot exist. The halting problem is unsolvable, but the clique problem is solvable in exponential time. Applying the reduction would allow us to test if a given program halts by solving a clique instance, which would allow us to solve the halting problem, which is a contradiction.

Problem 4. Recurrences [10 points]

Solve the following recurrence problems. For each recurrence relation, determine a function $f(n)$ such that $T(n) = \Theta(f(n))$, where $T(n)$ is the asymptotic complexity of an algorithm with running time given by the recurrence. You may assume that $T(1) = \Theta(1)$, and that $T(n)$ is a nondecreasing function of n. If you use the master theorem, then state which case you use.

(a) [5 points]

$$T(n) = T\left(\frac{2n}{3}\right) + O(1)$$

Solution: $a = 1, b = \frac{3}{2}$.

$n^{log_b a} = n^{log_{3/2} 1} = n^0 = 1$

$\Rightarrow$ By case 2 of the Master Theorem, $T(n) = \Theta(\log n)$.

(b) [5 points]

$$T(n) = 9T\left(\frac{n}{3\sqrt{3}}\right) + O(n^2)$$

Solution: $a = 9, b = 3\sqrt{3}$.

$n^{log_b a} = n^{log_{3\sqrt{3}} 9} < n^2$

$\Rightarrow$ By case 3 of the Master Theorem, $T(n) = \Theta(n^2)$.

Problem 5. Trust But Verify (your BST) [10 points]

(a) [6 points]

Below is the code for a function `verify`. The input to `verify` is a binary tree T (The root node of T is passed to `verify`). The function `verify` then returns True if and only if T is a binary search tree, and returns False otherwise.

Fill in the missing line of code below:

```
def verify(node):
  return helper(node, -float("Inf"), float("Inf"))

def helper(node, lo, hi):
  if not (node.left is None or helper(node.left, lo, node.key)):
      return False
  if not (node.right is None or helper(node.right, node.key, hi)):
      return False

  if ____________________________________________:
      return False
  else
      return True
```

Solution:

The blank should contain (something similar to):
`(node.key < lo) or (node.key > hi)`

(b) [2 points]

If the binary tree rooted at `node` is perfectly balanced and contains n nodes, write a recurrence relation $T(n)$ for the asymptotic worst case running time of this algorithm.

Solution:

$$T(n) = 2T(n/2) + O(1)$$

(c) [2 points]

Now solve for $T(n)$. You do not need to show any work.

Solution:

$$T(n) = \Theta(n)$$

Problem 6. Have You Seen My Integer? [10 points]

Suppose you are given a **sorted** array A containing $N-1$ distinct integers, each in the range $\{1, \ldots, N\}$, where $N > 1$ is a given integer. Thus, exactly one integer $a \in \{1, \ldots, N\}$ is not in A.

Provide an $O(\log N)$ algorithm to find the missing integer a. Prove that your algorithm is correct, and analyze its running time.

Solution: Binary search for the missing value as follows: guess an index i to be the location of the missing number. If $A[i+1] - A[i-1] = 2$, then return i. Otherwise, if $A[i] = i$, the missing number must be greater than i, and if $A[i] = i+1$, the missing number must be less than i. Perform recursion on the appropriate half of the array, yielding an $O(\log n)$ binary search algorithm.

Problem 7. Power to the Network [10 points] There are microwave towers along the 2451 miles of Route 66, allowing information to be transmitted from one end to another. New towers are added from time to time; when there are n towers we denote their positions by numerical values $x_1, x_2, \ldots, x_n$, where $x_i < x_{i+1}$ for $i = 1, 2, \ldots, n-1$. The power used to transmit a signal from one end of Route 66 to the other is proportional to

$$P = \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2 \ .$$

In this problem, you must design a *Signal Tower Location* data structure that maintains the location of the towers and supports the following operations:

- `STL.insert(x)`: for a number x, add a new tower at position x into the Signal Tower Location structure STL.
- `STL.getpower()`: return the value of P, as calculated above, for all tower locations currently stored in STL.

Describe a data structure that supports `getpower()` in $O(1)$ time, and `insert(x)` in $O(\log(n))$ time, where n is the number of towers currently inserted into the Signal Tower Location structure. You may assume that no two identical values of x will ever be inserted.

Solution: Use an AVL tree, and augment it with a single integer value for P (in other words, a single number is added to the data structure; not one number per node.) Upon insdering a new tower at position x, recalculate the value of P and update it. Upon inserting x_i, we can update the value of P in $O(\log n)$ time by looking at the predecessor and successor of x_{i-1} and x_{i+1} of x_i in the AVL tree, and then adding $(x_i - x_{i-1})^2 + (x_{i+1} - x_i)^2$ and subtracting $(x_{i+1} - x_{i-1})^2$.

Problem 8. Hashing [10 points]

Consider the following two hash functions h_1 and h_2, each from $\{1, 2, 3\}$ to $\{0, 1\}$:

	1	2	3
h_1	0	0	1
h_2	1	1	0

(i.e., $h_1(1) = 0, h_1(2) = 0, h_1(3) = 1, h_2(1) = 1, h_2(2) = 1, h_2(3) = 0$)

(a) [5 points]
Is the set $H = \{h_1, h_2\}$ a universal hash family? Why or why not?

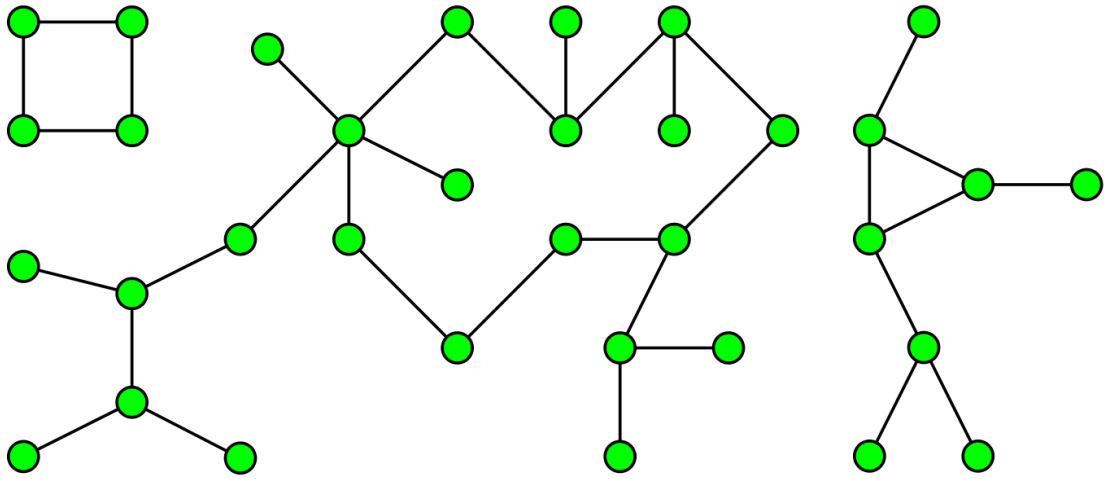
Solution: No! $h_i(1) = h_i(2)$ regardless of the choice of i, so elements 1 and 2 collide with probability greater than $\frac{1}{m} = \frac{1}{2}$.

(b) [5 points]
Suppose we modify h_1 so that $h_1(2) = 1$. Is the set $H = \{h_1, h_2\}$ a universal hash family? Why or why not?

Solution: Yes. Now elements 1 and 2 collide with probability $\frac{1}{2}$, elements 2 and 3 collide with probability $\frac{1}{2}$, and elements 1 and 3 collide with probability 0.

Problem 9. Pseudotrees [20 points]

A *pseudotree* is defined as a connected, undirected graph that contains exactly *ONE* cycle. Three sample pseudotrees are shown below:



In the following questions, suppose $T = (V, E)$ is a pseudotree.

(a) [5 points]

Suppose you want to represent T in a way that minimizes the amount of space used. Is it more efficient to use an adjacency matrix, or adjacency lists? Explain.

Solution: Adjacency lists are more efficient. T has $|V|$ edges exactly (one edge per vertex). Consequently, while an adjacency matrix uses $O(|V|^2)$ bits to represent T, an adjacency list uses only $O(|V| \log |V|)$ bits to represent T, which is a substantial savings.

(b) [5 points]

Define C to be the list of all vertices that lie in the cycle in T. Give an $O(V)$ algorithm that returns a list $[u_1, \ldots, u_k]$ containing precisely the vertices in C. Prove that your algorithm is correct and has the desired running time.

Solution: Perform a DFS on T. Upon reaching a vertex that has already been visited, simply pop back up the stack and print all of the vertices while doing so. This prints precisely the vertices along the cycle in T, because those vertices are exactly those vertices encountered when relaxing a chain of edges leading to a vertex that has been visited before. The running time guarantee follows from the running time of DFS, and the fact that $O(V + E)$ is $O(V)$ in pseudotrees.

(c) [10 points]

Now, suppose that each edge in T is given a positive edge weight, and suppose you are given a source vertex $s \in V$ that is ***NOT*** on the cycle C. Give an $O(V)$ algorithm that computes the shortest path length $\delta(s, u)$ from s to all vertices u that lie on the cycle C. Prove that your algorithm is correct and has the desired running time. You may use part (b) as a subroutine.

Solution: Run the algorithm from part (b) and mark each vertex in the cycle. Then do a BFS or DFS from s until a vertex x on C is reached. Record the value of $\delta(s, x)$. Then do two searches around the edges in C, one proceeding in each direction around the cycle. In doing so, record the distance from x to each vertex $y \in C$ if one goes from x to y walking each of the two directions of the cycle. Then assign $\delta(x, y)$ to be the minimum of the two distances. Finally, compute $\delta(s, y) = \delta(s, x) + \delta(x, y)$ for each $y \in C$. This is correct since x must lie along a shortest path from s to any $y \in C$. The running time is clearly $O(V)$ as we run $O(1)$ graph searches in a graph with $O(V)$ edges.

Problem 10. Single Negative Edge [15 points]

Let $G = (V, E)$ be a directed graph having weights on each edge; let $w(e) = w(u, v)$ denote the weight on the directed edge $e = (u, v)$. Of all the edges in G, there is only *one* edge (x, y) whose weight is negative; all other edge weights are non-negative.

(a) [5 points]

Give an $O(E \log V)$ algorithm that determines whether or not G has a negative-weight cycle. You do not need to prove correctness or verify the running time.

Solution: Run Dijkstra's algorithm from y to x to determine the length $\delta(y, x)$ of a shortest path from y to x. Then G has a negative-weight cycle if and only if $-\delta(y, x)$ is larger than $w(x, y)$.

(b) [10 points]

Suppose that you are given a starting vertex $s \in V$ and a destination vertex $t \in V$, both of which are distinct from x and y. Assume that there is a path from s to t, and that G does not contain a negative-weight cycle. Give an $O(E \log V)$ algorithm that returns the weight of a shortest path from s to t. You do not need to prove correctness or verify the running time.

Solution: There are two cases: either (x, y) is used in the shortest path, or (x, y) is not used. To check the latter case, just delete the edge from x to y and then run Dijkstra's algorithm to find $\delta^*(s, t)$, the length of a shortest path from s to t that does not use the edge (x, y). To check the case where (x, y) is used, find the length of the shortest paths from s to x and from y to t, and compute the quantity $\delta(s, x)+w(x, y)+\delta(y, t)$, which is the length of the shortest path from s to t that passes through edge (x, y). Return the $\min(\delta^*(s, t), \delta(s, x) + w(x, y) + \delta(y, t)\})$, which gives the length of the shortest path from s to t.

Problem 11. Ganon, the Barbarian [15 points]

Ganon, the legendary king of all evil, wants to protect all of the treasure hidden in his dungeon by installing security cameras. His n pieces of treasure are located at fixed positions $\mathcal{T} = \{T_1, \ldots, T_n\}$ in the plane, and Ganon is considering n different possible locations $\mathcal{C} = \{C_1, \ldots, C_n\}$ where he can potentially install security cameras. The price to install a security camera at location C_i is given by a positive integer p_i, which may be different for each camera location. Each camera can only be installed facing the lower-left direction, and has sight of all treasures both *below and left* of it. See the figure below.

Throughout this problem, you should assume that:

- No treasure T_i is below and left of any other treasure T_j
- No two locations (camera and/or treasure) share the same x-coordinate or y-coordinate
- The treasure locations are sorted from left to right (so T_i is left of T_{i+1})

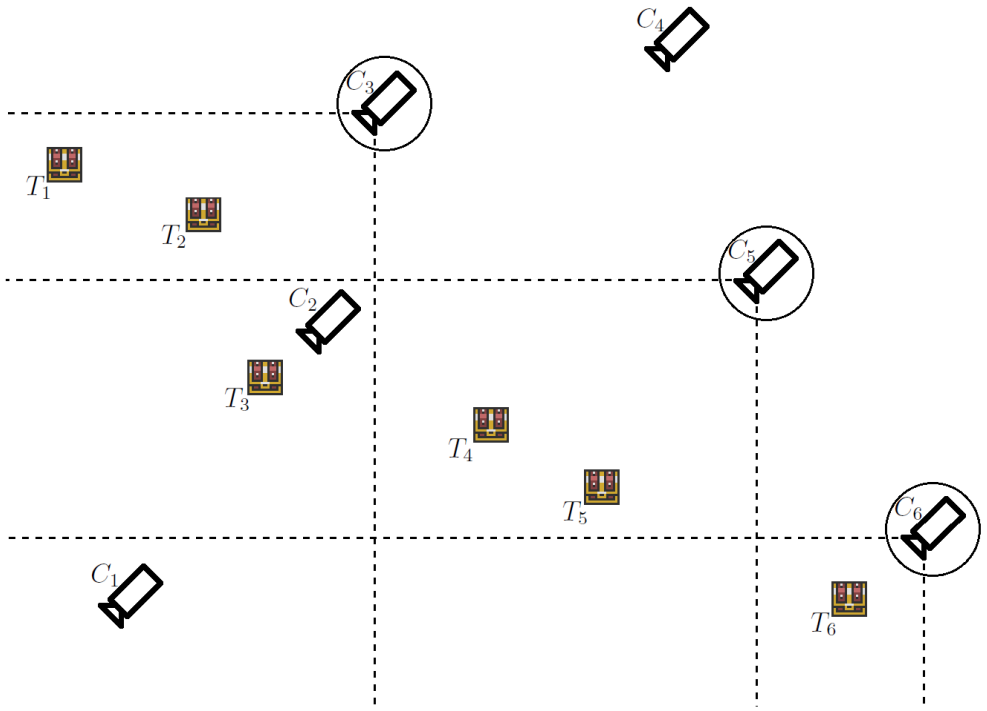


Figure 1: Example treasure and potential camera locations. The circled cameras, taken together, have vision of every treasure. Prices p_i are not shown.

For a camera location C, define $RLA(C)$ to be the *rightmost* location of any treasure $T \in \mathcal{T}$ that is both *left* and *above* C. Define $RLA(C) = \emptyset$ if no such treasure exists. In the example, $RLA(C_6) = T_5$, and $RLA(C_3) = \emptyset$.

(a) [7 points]
Define $M[T_k]$ to be the minimum price of installing cameras to guard treasures $\{T_1, \ldots, T_k\}$ (that is, the leftmost k treasures). Define $M[\emptyset] = 0$. Write down a recurrence that expresses $M[T_k]$ in terms of the values $\{M[T_j] : j < k\}$ and the camera installation prices p_i. No proof or justification is required. Hint: use the notation $RLA(C_i)$.

Solution: $M[T_k] = \min_{C \in \mathcal{C}: C \text{ guards } T_k} \{p_C + M[RLA(C)]\}$.

(b) [8 points] Evaluating your recurrence likely requires evaluating $RLA(C)$ for various camera locations C. As it turns out, there is a fast way of doing this. Read the pseudocode below, then fill in the missing portions such that the resulting algorithm correctly computes $RLA(C)$ for all $C \in \mathcal{C}$, and runs in time $O(n \log n)$:

1. Let A be an AVL tree (initially empty) where each node in A stores a location in the plane. Order A such that if N is a node containing a point (x, y), then

$$key[N] = \_\_\_\_x\_\_\_\_.$$

2. Let $\mathcal{R} = \mathcal{C} \cup \mathcal{T}$ be a set containing all cameras and treasures.
3. Sort $\mathcal{R}$ in decreasing order of y-coordinate (i.e., from top to bottom).
4. For each point $a \in \mathcal{R}$ (in top-to-bottom order), do the following:
5. If a is a treasure location, insert a into A.
6. If a is a camera location C, compute $RLA(C)$ as follows:

Solution: Do an AVL search for the rightmost node whose key is less than or equal to the x-coordinate of C.

Problem 12. Optimizing Santa [15 points]

Santa is given an ordered sequence of n children $[c_1, \ldots, c_n]$, and an ordered sequence of n toys $[t_1, \ldots, t_n]$. For each $1 \leq i \leq n$ and $1 \leq j \leq n$, the non-negative quantity h_{ij} gives the happiness that child c_i will obtain upon receiving toy t_j. Santa wishes to assign toys to children in order to maximize the total aggregate happiness:

$$H = \sum_{i=1..n} \sum_{j=1..n} x_{ij} h_{ij}$$

where $x_{ij} = 1$ if the child c_i receives the toy t_j, and $x_{ij} = 0$ otherwise. Each child may receive at most one toy, and each toy may be given to at most one child. Note that if a child receives no toy, the child obtains happiness 0.

Unfortunately, due to time constraints, Santa is only allowed to give away at most q toys, where $q \leq n$ is a known positive integer.

Additionally, due to sleigh-packing rules, Santa's toy assignment must obey the following *no-crossing constraint*: for $i < k$, if c_i receives t_j and c_k receives t_ℓ, then it must be that $j < \ell$.

(a) [10 points]

The maximum aggregate happiness H can be computed using dynamic programming. Let $H[i, j, k]$ denote the maximum aggregate happiness that is obtainable if at most k children (where $1 \leq k \leq q$) from the set $\{c_1, \ldots, c_i\}$ are assigned toys from the set $\{t_1, \ldots, t_j\}$, while obeying the *no-crossing constraint*. Write a recurrence relation for $H[i, j, k]$. Include base cases.

Solution:

$$H[0, j, k] = H[i, 0, k] = H[i, j, 0] = 0$$

$$H[i, j, k] = \max(H[i{-}1, j, k], H[i, j{-}1, k], H[i{-}1, j{-}1, k{-}1]{+}h_{ij}) \qquad \forall i, j, k \geq 1.$$

(b) [5 points]

Fill in the blanks below using big O notation:

The total number of subproblems is _____.

Solution: $O(n^2 q)$

The time required to evaluate $H[i, j, k]$ in each subproblem is _____.

Solution: $O(1)$

The total time required to compute H is _____.

Solution: $O(n^2 q)$

Problem 13. The Longest ~~Quiz~~ Path in a Tree [15 points]

Let $T = (V, E)$ be an undirected tree (a connected graph with no cycles) with non-negative edge weights. Design and analyze an efficient algorithm to determine the length of the *longest path* in T, i.e., the value of

$$\max_{u,v \in V} \delta(u, v)$$

where $\delta(u, v)$ denotes the length of the shortest path from u to v. For full marks, an $O(V)$ algorithm is required.

Hint: Start by selecting an arbitrary vertex $r \in V$ to be the *root* of T.

Solution: First, select an arbitrary vertex $r \in V$ to be the *root* of T, and run a DFS or BFS to direct all edges outwards, so that we now have a rooted tree. For any given path P, there is a unique "highest" node, call it r, in this tree. (The path starts somewhere, goes up the tree, and then goes down.) In this case, we say that P is rooted at r.

Now, for each vertex v, define $P[v]$ to be the subproblem of finding the longest path from v to some leaf in the subtree rooted at v. We then have the recurrence:

$$P[v] = \max_u \left(P[u] + w_{u,v}\right)$$

This is a DP which with $O(V)$ subproblems, where each subproblem takes $O(V)$ time to evaluate. However, notice that there is only one evaluation for each edge, and so the computation takes $O(E)$ time total. Since this is a tree, $|E| = |V| - 1$, and so $O(E) = O(V)$.

Next, we want to compute $L[v]$, the length of the longest path rooted at v. First, we claim that the longest path must go from a leaf to another leaf. If not, we could make a longer path by including another edge adjacent to the non-leaf.

Thus, to do this, for each child u, consider the quantity $P[u] + w_{u,v}$. Let u_1 and u_2 be the two children with the largest values of this. Then, we have

$$L[v] = P[u_1] + w_{u_1,v} + P[u_2] + w_{u_2,v}$$

Computing all the $L[v]$ values takes $O(V)$, by the same argument as for $P[v]$.

Finally, we have that the length of the longest path is simply

$$\max_v L[v]$$