CS2040S — Data Structures and Algorithms
School of Computing
National University of Singapore

Final Assessment

3 May 2024 **Time allowed:** 2 hours

Instructions — please read carefully:

1. Do not open the final until you are directed to do so.
2. Read **all** the instructions first.
3. The assessment is closed book. You may bring one double-sided sheet of A4 paper to the exam hall. (You may not bring any magnification equipment!) You may NOT use a calculator, your mobile phone, or any other electronic device.
4. The **QUESTION SET** comprises **SIX (6) questions** and **NINETEEN (19) pages**, and the **ANSWER SHEET** comprises of **FOURTEEN (14) pages**.
5. The time allowed for solving this test is **2 hours**.
6. The maximum score of this test is **100 marks**. The weight of each question is given in square brackets beside the question number.
7. All questions must be answered correctly for the maximum score to be attained.
8. All questions must be answered in the space provided in the **ANSWER SHEET**; no extra sheets will be accepted as answers.
9. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
10. An excerpt of the question may be provided above the answer box. It is to aid you to answer in the correct box and is not the exact question. You should refer to the original question in the question booklet.
11. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
12. Unless otherwise stated in the question, when we ask for the worst-case big-O running time of an algorithm we always mean to give the tightest possible answer.
13. Unless otherwise stated in the question, we will assume that operators behave as per Java (e.g., 5/2 will evaluate to 2). However, pseudocode does not necessarily satisfy Java syntax (unless stated otherwise) and things that do not compile as legal Java are not necessarily bugs (as long as they are clear pseudocode).
14. Unless otherwise stated in the question, we are not concerned with overflow errors, e.g., storing the value $2^{245,546,983}$ in an integer.

GOOD LUCK!

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: True or False? [19 marks]

Choose either True or False for each statement below.

A. If $h : U \to \{1, \ldots, m\}$ is a fixed hash function where $|U|$ is much larger than m, then in every set of m keys from U, there will always be two keys which collide under h.

[1 mark]

1. True
2. False

B. Let X and Y be two random variables, not necessarily independent. Then, it must be that $\mathbb{E}[X + 2Y] = \mathbb{E}[X] + 2 \cdot \mathbb{E}[Y]$.

[1 mark]

1. True
2. False

C. Let X and Y be two random variables, not necessarily independent. Then, it must be that $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$.

[1 mark]

1. True
2. False

D. Let `getTime()` be a method that returns an integer representing the current time. Then, the following is a valid implementation of `hashCode`:

```
public int hashCode() {return getTime();}
```

[1 mark]

1. True
2. False

E. Given an array A of n integers, the running time of HeapSort is $O(n)$.

[1 mark]

1. True
2. False

F. For a (binary) heap, if n is the number of elements and h is its height, then $n \geq 2^h$. (A heap with only one element has height 0.)

[1 mark]

1. True
2. False

G. Two heaps of size n can be merged in $O(n)$ time.

[1 mark]

1. True
2. False

H. Let $h : U \times \{1, \ldots, m\} \to \{1, \ldots, m\}$ be a hash function used in an open addressing scheme to map from a universe U of keys to a hash table with m slots. Suppose h satisfies the uniform hashing assumption. Then, if we define $g : U \to \{1 \ldots, m\}$ as $g(k) = h(k, 1)$, then g is a hash function that can be used for hashing with chaining, and g satisfies the simple uniform hashing assumption used for chaining.

[1 mark]

1. True
2. False

I. Consider the binary counter on k bits, where two operations, `increment`() and `read`(), are supported. We argued in class that if the counter starts from the all-zero state, then the amortized cost of `increment` is constant. The same holds true if the counter is no longer required to be initialized at all zeros.

[1 mark]

1. True
2. False

J. Depth-first search takes $\Theta(V^2)$ time on a graph with V vertices represented as an adjacency matrix.

[1 mark]

1. True
2. False

K. Given a weighted directed graph G and a shortest path p from nodes s to t, if we added 5 to the weight of each edge to produce G', then p is also a shortest path in G'.

[1 mark]

1. True
2. False

L. In a weighted undirected tree, breadth-first search from a node s finds single-source shortest paths from s (via parent pointers).

[1 mark]

1. True
2. False

M. In a weighted undirected tree, depth-first search from a node s finds single-source shortest paths from s (via parent pointers).

[1 mark]

1. True
2. False

N. Suppose for an undirected graph G, a breadth-first search from a source node s visits a vertex v at level k. Then, all paths from s to v must be of length at most k.

[1 mark]

1. True
2. False

O. Suppose G is a directed, weighted graph with non-negative weights but containing directed cycles. Then, Dijkstra's shortest-path algorithm may relax an edge more than once.

[1 mark]

1. True
2. False

P. Assuming $T(1) = 1$, the tightest possible bound for the recurrence $T(n) = 2T(n/2) + n^2$ is $T(n) = O(n^2 \log n)$.

[1 mark]

1. True
2. False

Q. Quicksort's running time is $\Omega(n \log n)$ on any array of length n. Assume randomly chosen pivot and 2-way partitioning.

[1 mark]

1. True
2. False

R. Inserting in an AVL tree may require $\Omega(\log n)$ rotations.

[1 mark]

1. True
2. False

S. Any n-node unbalanced tree can be balanced using $O(\log n)$ rotations.

[1 mark]

1. True
2. False

Question 2: Open Address-uh-Chaining [22 marks]

A. Suppose we have a hash table where collisions are resolved using chaining. The hash table has m slots, and n keys are to be inserted. Make the simple uniform hashing assumption. What is the expected number of slots with chain length exactly 1 after all n keys have been inserted.

[4 marks]

1. n/m
2. $1 + n/m$
3. $(1 - 1/m)^{n-1}$
4. $(1 - 1/m)^{m-1}$
5. $n(1 - 1/m)^{n-1}$
6. $n(1 - 1/m)^{m-1}$

B. Again, assume that we have a hash table with m slots that uses chaining to resolve collisions. Suppose that at some point, k of the m slots have a chain of length at least 1. Under the simple uniform hashing assumption, what is the probability that the next insertion does not result in a collision?

[3 marks]

1. k/m
2. $1 - k/m$
3. $(1 - k/m)^{m}$
4. $(k/m)^{m}$
5. $(1 - k/m)^{m-1}$
6. $(k/m)^{m-1}$

C. Now, suppose that we have a hash table with m slots that uses open addressing to resolve collisions. Suppose that k keys have been inserted into the hash table so far. Under the uniform hashing assumption, what is the probability that the next insertion results in less than 2 probes?

[4 marks]

1. k/m
2. $1 - k/m$
3. k^2/m^2
4. $1 - k^2/m^2$
5. $(k^2 - k)/(m^2 - m)$
6. $1 - (k^2 - k)/(m^2 - m)$

D. Again, assume that we have a hash table with m slots that uses open addressing to resolve collisions. Suppose we are inserting n keys into the hash table. Under the uniform hashing assumption, and supposing $m = 2n$, what is the expected number of insertions that require exactly one probe?

[4 marks]

1. $3n/4$
2. $(3n+1)/4$
3. $n/2$
4. $(n+1)/2$
5. $n/4$
6. $(n+3)/4$

E. Now, let's combine open addressing with chaining. Suppose that the hash table has m slots, and each slot admits a chain of length at most c. We use a hash function $h(\text{key}, i)$, just as in open addressing. For inserting a key k, at the i'th probe, we check if the $h(k,i)$ slot has $< c$ keys as a chain, and if so, k is inserted to this chain; otherwise, we probe again.

Suppose n items have already been inserted in the table, resulting in m_0 slots with 0 keys, m_1 slots with 1 key, ..., m_c slots with c keys. Under the uniform hashing assumption, what is the probability that the next insertion requires only one probe?

[3 marks]

1. m_c/m
2. $1 - m_c/m$
3. $1 - m_c m_{c-1}/m^2$
4. $m_c m_{c-1}/m^2$
5. m_0/m
6. $1 - m_0/m$

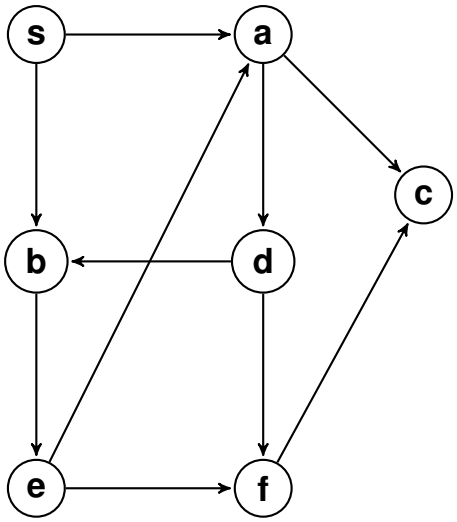
F. Continuing with the setup in part E, what is the probability that $m_2 = m_3 = \cdots = m_c = 0$ after n insertions, assuming uniform hashing for these insertions?

[4 marks]

1. $\frac{m!}{m^n(m-n)!}$
2. $\frac{m!}{m^c(m-n)!}$
3. $(1 - \frac{1}{m})^n$
4. $(1 - \frac{1}{m})^c$
5. $1 - \frac{n^2}{m}$
6. $1 - \frac{nc}{m}$

Question 3: Fighting Dragons and Spells [22 marks]

A. Akrabakra is an adventurer stationed at node s. She is tasked with checking on the towns in her realm. The towns of her realm are connected by one-way roads, and the connections can be represented as a graph as follows (where towns are nodes and roads are edges).



The graph above is a DAG.

[1 mark]

1. True
2. False

B. Akrabakra uses a pre-order DFS traversal to check on the towns, where the roads connecting the towns are depicted in part A. What is a possible order of the towns in this traversal?

[2 marks]

1. s, a, b, c, d, e, f
2. s, a, b, d, c, e, f
3. s, b, e, a, c, f, d
4. s, b, e, a, d, f, c
5. c, f, e, b, d, a, s
6. c, a, d, f, b, e, s

C. Akrabakra uses a post-order DFS traversal to check on the towns, where the roads connecting the towns are depicted in part A. What is a possible order of the towns in this traversal?

[2 marks]

1. s, a, b, c, d, e, f

2. s, a, b, d, c, e, f
3. s, b, e, a, c, f, d
4. s, b, e, a, d, f, c
5. c, f, e, b, d, a, s
6. c, a, d, f, b, e, s

D. From this part onwards, consider the general situation where Akrabakra possesses an arbitrary connected, directed graph with n nodes and m edges describing the road structure.

Dragons have invaded Akrabakra's realm! Scouts have informed her about the number of dragons n_e on each road (edge) e. For each town v, she wants to find a path from s to v that minimizes the number of dragons encountered along the way. Which algorithm would you recommend to her for this task, if she wants to minimize the algorithm's asymptotic running time as a function of n and m?

[2 marks]

1. Breadth-first search
2. Depth-first search
3. Bellman-Ford
4. Dijkstra
5. Relax in topological order

E. Now, the scouts report that dragons are also at some of the towns, not just on the roads connecting the towns. Thankfully, they haven't made it yet to Akrabakra's base s. For each town v, she knows n_v, the number of dragons at town v. Also, she knows n_e for each edge e, as in part D. Just as in part D, she wants to find a path from s to each town minimizing the number of dragons encountered along the way (counting both at the towns and roads visited, including both endpoints of the path).

How should she transform the graph before running a single-source shortest path algorithm from s?

[4 marks]

1. Each existing edge e has weight n_e. At each node v, create a new self-loop edge (v, v) with weight n_v.
2. For each edge $e = (u, v)$, let it have weight $n_e + n_u$.
3. For each edge $e = (u, v)$, let it have weight $n_e + n_v$.
4. None of the above.

F. All the dragons have left the realm finally, but before they left, they cast magic spells on some of the towns. Let X be the set of towns which are so affected. Since the dragons didn't enter s, s is not in X. There is a particular node t Akrabakra wants to reach from s, but only using a path that passes through at most one node in X. Which of the following allows her to decide whether t can be reached with this constraint or not?

[4 marks]

1. Perform a BFS traversal from s, and check if the path from s to t (induced by the parent pointers) contains at most one node in X.
2. Perform a DFS traversal from s, and check if the path from s to t (induced by the parent pointers) contains at most one node in X.
3. Create a new graph G'. For each town v, create two nodes v_0 and v_1 in G'. For each road (u, v), create three edges (u_0, v_0), (u_0, v_1) and (u_1, v_1). Check (using BFS or DFS) whether t_1 is reachable from s_0 in G'.
4. Create a new graph G'. For each town v, create two nodes v_0 and v_1 in G'. For each road (u, v), if $u \in X$, create an edge (u_0, v_1), and if $u \notin X$, create two edges (u_0, v_0) and (u_1, v_1). Check (using BFS or DFS) whether t_1 is reachable from s_0 in G'.
5. Create a new graph G'. For each town v, create two nodes v_0 and v_1 in G'. For each road (u, v), if $u \in X$, create an edge (u_0, v_1), and if $u \notin X$, create two edges (u_0, v_0) and (u_1, v_1). Check (using BFS or DFS) whether either t_0 or t_1 is reachable from s_0 in G'.
6. Create a new graph G'. For each town v, create two nodes v_0 and v_1 in G'. For each road (u, v), if $v \in X$, create an edge (u_0, v_1), and if $v \notin X$, create two edges (u_0, v_0) and (u_1, v_1). Check (using BFS or DFS) whether t_1 is reachable from s_0 in G'.
7. Create a new graph G'. For each town v, create two nodes v_0 and v_1 in G'. For each road (u, v), if $v \in X$, create an edge (u_0, v_1), and if $v \notin X$, create two edges (u_0, v_0) and (u_1, v_1). Check (using BFS or DFS) whether either t_0 or t_1 is reachable from s_0 in G'.
8. None of the above.

G. It turns out that some of the towns have more powerful spells cast on them than others. For each node v, let Z_v be the amount of energy that needs to be spent in order to fight the spell cast on v. Each Z_v is a non-negative integer and is known to Akrabakra.

Let $Z = \sum_v Z_v$ be the sum of Z_v over all nodes v. Suppose Z is at least as large as m (the number of roads). For any given town t, describe an efficient algorithm that Akrabakra can use to find a path from s to t that minimizes the amount of energy used to fight the spells at the towns on the path. Analyze the running time. For full marks, your algorithm should run in time $O(Z)$.

[7 marks]

Question 4: Where to search [8 marks]

Imagine you are given a sorted array A, and you want to perform a binary search for a key k. If the key k appears multiple times in the array, it is often okay to return any index that contains k. For example, in the array:

$$A = [3,4,5,6,6,6,6,6,6,6,6,7,9]$$

a search for 6 might return 3 or 7 or 10.

In this problem, we are going to implement a version of binary search that returns the largest index containing the key. In the example above, it would return the index 10.

```
int searchMax(int key, int[] A, int n)
    begin = 0;
    end = n-1;
    while (begin < end) {
          mid = (begin+end)/2 + [ A ] ;
          if (key [ B ] A[mid]) {
                begin = [ C ] ;
          }
          else {
                end = [ D ] ;
          }
    }
    return A[begin];
```

A. A=?

[2 marks]

1. 0
2. 1
3. -1
4. `begin`
5. `begin/2`
6. `<`
7. `>`
8. `<=`
9. `>=`
10. `=`
11. `mid-1`
12. `mid`
13. `mid+1`

B. B=?

[2 marks]

1. 0
2. 1
3. -1
4. `begin`
5. `begin/2`
6. `<`
7. `>`
8. `<=`
9. `>=`
10. `=`
11. `mid-1`
12. `mid`
13. `mid+1`

C. C=?

[2 marks]

1. 0
2. 1
3. -1
4. `begin`
5. `begin/2`
6. `<`
7. `>`
8. `<=`
9. `>=`
10. `=`
11. `mid-1`
12. `mid`
13. `mid+1`

D. D=?

[2 marks]

1. 0
2. 1
3. -1
4. `begin`
5. `begin/2`
6. `<`
7. `>`
8. `<=`
9. `>=`
10. `=`
11. `mid-1`
12. `mid`
13. `mid+1`

Question 5: Paths and Trees [16 marks]

A. For Dijkstra's algorithm, the standard implementation is to use heaps as priority queues, yielding a running time of $O((E+V)\log V)$. However, Edsger Dijkstra himself originally used a simple unsorted array as the priority queue! Specifically, in his scheme, the vertex with the minimum estimated distance was found by a brute-force scan of the array, while a vertex's estimated distance was updated by following a pointer from the vertex to its position in the array and updating the array entry. Which of the following is true?

[3 marks]

1. Dijkstra's implementation has better asymptotic running time for sparse graphs (small E)
2. Dijkstra's implementation has better asymptotic running time for dense graphs (large E)
3. Dijkstra's implementation always has better asymptotic running time but has higher memory usage.
4. The standard implementation always has better asymptotic running time than Dijkstra's version.

B. Let G be a weighted, directed graph, with V vertices, E edges, and two designated vertices s and t. Among all paths from s to t with at least 20 edges, find a path of minimum total weight (if such a path exists). Which of the following algorithms is correct?

[4 marks]

1. Run BFS on G from s. Find a vertex v which is on level 20 of the BFS tree; let P_1 be the BFS tree path from s to v. Run Bellman-Ford on G from v, and find a minimum weight path P_2 from v to t. Output the concatenation of P_1 and P_2.
2. Run Bellman-Ford on G from s. Find a vertex v which is on level 20 of the shortest-path tree; let P_1 be the minimum weight path from s to v. Run Bellman-Ford on G from v, and find a minimum weight path P_2 from v to t. Output the concatenation of P_1 and P_2.
3. Let G_2 be G but with edge orientations reversed. Run BFS on G_2 from t. Find a vertex v which is on level 20 of the BFS tree; let P_1 be the BFS tree path from t to v. Run Bellman-Ford on G_2 from v, and find a minimum weight path P_2 from v to s. Concatenate P_1 and P_2, reverse the edge orientations, and output.
4. Let G_3 be the weighted graph, which for each vertex v in G, has 21 vertices $v_0, v_1, \ldots, v_{20}$, and for each edge (u,v) in G, has 20 edges $(u_0,v_1),(u_1,v_2),\ldots,(u_{19},v_{20})$, each with the same weight as (u,v) in G. Run Bellman-Ford on G_3 from s_0, and let w in G be such that the path from s_0 to w_{20} has the smallest weight in G_3. Let P_1 be the corresponding path of 20 edges from s to w in G. Run Bellman-Ford on G from w, and let P_2 be a minimum weight path from w to t. Output the concatenation of P_1 and P_2.
5. Let G_3 be as in the 4th option above. Run Bellman-Ford on G_3 from s_0. For each vertex v in G, let $\delta_1(v)$ be the minimum weight of a path from s_0 to v_{20} in G_3 and $P_{1,v}$ the corresponding path of 20 edges from s to v in G. Let G_2 be as in the 3rd option above. Run Bellman-Ford on G_2 from t. For each vertex v in G, let $\delta_2(v)$ be the minimum

weight of a path from t to v in G_2 and let $P_{2,v}$ the corresponding path in G from v to t. Let w be the vertex that minimizes $\delta_1(w) + \delta_2(w)$; output the concatenation of $P_{1,w}$ and $P_{2,w}$.

6. None of the above.

C. Let G be a directed graph that is both node-weighted and edge-weighted. Meaning that each vertex v has a real-valued weight w_v, and each edge e has a real-valued weight w_e. The weight of a path is defined (as usual) as the sum of the weights of the edges on the path. The weights can be positive, negative, or zero, but assume that there are no negative-weight cycles.

Let s and t be two designated vertices in G. Each of the following defines a weighted graph G' on the same set of vertices and edges as G. For which is it true that if P is a minimum-weight path from s to t in G, then P must also be a minimum-weight path from s to t in G'?

[3 marks]

1. In G', the weight of an edge e is $w'_e = w_e + 1$.
2. In G', the weight of an edge e is $w'_e = w_e^2$.
3. In G', the weight of an edge $e = (u, v)$ is $w'_e = 2w_u + w_e$.
4. In G', the weight of an edge $e = (u, v)$ is $w'_e = 2w_u - 2w_v + w_e$.
5. In G', the weight of an edge $e = (u, v)$ is $w'_e = 2w_u - 3w_v + w_e$.
6. In G', the weight of an edge $e = (u, v)$ is $w'_e = 2w_u - 3w_v$.
7. In G', the weight of an edge $e = (u, v)$ is $w'_e = 2w_u w_e$.

D. Let G be as in part C above, but ignore the orientations on the edges so that it is undirected. Each of the following defines a weighted undirected graph G' on the same set of vertices and edges as G. For which is it true that if T is a minimum-weight spanning tree of G, then T must also be a minimum-weight spanning tree of G'?

[3 marks]

1. In G', the weight of an edge e is $w'_e = w_e + 1$.
2. In G', the weight of an edge e is $w'_e = w_e^2$.
3. In G', the weight of an edge $e = \{u, v\}$ is $w'_e = 2w_u + 2w_v + w_e$.
4. In G', the weight of an edge $e = \{u, v\}$ is $w'_e = 2w_u^2 + 2w_v^2 + w_e$.
5. In G', the weight of an edge $e = \{u, v\}$ is $w'_e = 2w_u + 2w_v$.
6. In G', the weight of an edge $e = \{u, v\}$ is $w'_e = 2w_u w_v w_e$.

E. Suppose G is an undirected weighted graph, and let T be a given minimum spanning tree (MST) of G. For some edge e not in T, decrease the weight of e.

Here is a proposed algorithm to update the MST. Let u be an endpoint of e. Run BFS from u on the graph $T \cup \{e\}$, and find the unique cycle C in this graph. Let e' be the maximum weight edge in C. Remove e' from $T \cup \{e\}$; the resulting tree is an MST.

Which of the following is true?

[3 marks]

(I) The proposed algorithm is correct.

(II) The proposed algorithm runs in time $O(V + E)$.

(III) Both (I) and (II) are correct.

(IV) Neither (I) and (II) is correct.

Question 6: Fun at Playtech [13 marks]

Adam, Alan, and Ellie work at Playtech, an electronics store, with their boss being Rowan.

A. There are n boxes that need to be stacked, with integer heights $z_1, \ldots, z_n$. Rowan issues Adam, Alan, and Ellie the following challenge. Each of the three has to build a stack starting from the level floor. For each box, the three have to decide among themselves whose stack the box should go on. If after all the boxes are done, the three stacks are the same height, Rowan will provide free pizza. Otherwise, they will each have to take a 20% paycut! Adam, Alan and Ellie want you to help them decide whether they should accept Rowan's challenge.

Let $T[i, h_1, h_2, h_3]$ be true if the first i boxes can be stacked so that Adam's stack has height h_1, Alan's h_2, and Ellie's h_3, and false otherwise. Which of the following gives a valid base case for T?

[2 marks]

1. $T[0, h_1, h_2, h_3]$ = True if $h_1 = h_2 = h_3 = 0$, and False otherwise.
2. $T[0, h_1, h_2, h_3]$ = True if $h_1 = 0$ or $h_2 = 0$ or $h_3 = 0$, and False otherwise.
3. $T[1, h_1, h_2, h_3]$ = True if $h_1 = h_2 = h_3 = 0$, and False otherwise.
4. $T[1, h_1, h_2, h_3]$ = True if $h_1 = 0$ or $h_2 = 0$ or $h_3 = 0$, and False otherwise.
5. None of the above.

B. Continuing the problem in part A, which of the following gives a correct recurrence for T?

[4 marks]

1. $T[i, h_1, h_2, h_3] = T[i-1, h_1 - z_i, h_2 - z_i, h_3 - z_i]$
2. $T[i, h_1, h_2, h_3] = (h_1 \geq z_i \text{ OR } h_2 \geq z_i \text{ OR } h_3 \geq z_i) \text{ AND } T[i-1, h_1 - z_i, h_2 - z_i, h_3 - z_i]$
3. $T[1, h_1, h_2, h_3] = T[i-1, h_1 - z_i, h_2, h_3] \text{ OR } T[i-1, h_1, h_2 - z_i, h_3] \text{ OR } T[i-1, h_1, h_2, h_3 - z_i]$
4. $T[i, h_1, h_2, h_3] = (h_1 \geq z_i \text{ OR } T[i-1, h_1 - z_i, h_2, h_3]) \text{ OR } (h_2 \geq z_i \text{ OR } T[i-1, h_1, h_2 - z_i, h_3]) \text{ OR } (h_3 \geq z_i \text{ OR } T[i-1, h_1, h_2, h_3 - z_i])$
5. $T[i, h_1, h_2, h_3] = (h_1 \geq z_i \text{ AND } T[i-1, h_1 - z_i, h_2, h_3]) \text{ OR } (h_2 \geq z_i \text{ AND } T[i-1, h_1, h_2 - z_i, h_3]) \text{ OR } (h_3 \geq z_i \text{ AND } T[i-1, h_1, h_2, h_3 - z_i])$
6. $T[i, h_1, h_2, h_3] = (h_1 \geq z_i \text{ AND } T[i-1, h_1 - z_i, h_2, h_3]) \text{ AND } (h_2 \geq z_i \text{ AND } T[i-1, h_1, h_2 - z_i, h_3]) \text{ AND } (h_3 \geq z_i \text{ AND } T[i-1, h_1, h_2, h_3 - z_i])$

C. Continuing the problem from above, your goal is to output $T[n, Z/3, Z/3, Z/3]$ where $Z = z_1 + z_2 + \cdots + z_n$; note that if Z is not divisible by 3, you can decide FALSE right away.

You run your algorithm in a top-down fashion, using memoization to evaluate recursive calls. What is the tightest bound among the below options for the asymptotic running time of this algorithm?

[3 marks]

1. $O(n^4)$
2. $O(nZ^3)$
3. $O(n^2Z^2)$
4. $O(nZ^2)$
5. $O(Z^4)$
6. None of the above

D. Adam, Alan, and Ellie are done with stacking boxes and the store is closed to customers; so, they want to go home early. But Rowan is not happy with this development and gives them busywork to delay them as long as possible. Can you help them so that they can be out of Rowan's sights as quickly as possible?

Rowan has provided them a long document and asked them to count how many times "STUPID" appears as a subsequence in the document. More generally, given two strings X of length n and Y of length m, the question is how many times Y appears as a subsequence of X. For example, if X is STUPIDSTUPID and Y is STUPID, then Y occurs 7 times as a subsequence in X (one such occurrence is as the first three and last three characters of X).

Which of the following yields a valid algorithm to solve the problem? (Below, the notation $X[i \dots j]$ denotes the substring of X from the i'th to the j'th characters inclusive; e.g., if X = DOG, then $X[2 \dots 3]$ = OG.) Assume $m \leq n$.

[4 marks]

1. Maintain $A[i][j]$ to be the count of the occurrences of $Y[1 \dots i]$ as a subsequence in $X[1 \dots j]$. Use base case $A[0][0] = A[0][1] = \cdots = A[0][n] = 0$, and apply recurrence $A[i][j] = A[i-1][j-1] + 1$.
2. Maintain $B[i][j]$ to be the count of the occurrences of $Y[1 \dots i]$ as a subsequence in $X[1 \dots j]$. Use base case $B[0][0] = B[1][0] = \cdots = B[m][0] = 0$, and apply recurrence: $B[i][j] = B[i][j-1]$ if $Y[i] \neq X[j]$ and $B[i][j] = B[i][j-1] + B[i-1][j-1]$ if $Y[i] = X[j]$.
3. Maintain $C[i][j]$ to be the count of the occurrences of $Y[1 \dots i]$ as a subsequence in $X[1 \dots j]$. Use base case $C[0][0] = C[0][1] = \cdots = C[0][n] = 0$, and apply recurrence: $C[i][j] = C[i-1][j]$ if $Y[i] \neq X[j]$ and $C[i][j] = C[i-1][j] + C[i-1][j-1]$ if $Y[i] = X[j]$.
4. Maintain $D[i][j]$ to be the count of the occurrences of $Y[i \dots j]$ as a subsequence in X. Let n_i be number of occurrences of $Y[i]$ in X. Use base case: for every i, $D[i][i] = n_i$. Apply recurrence: $D[i][j] = D[i+1][j-1] \cdot n_i \cdot n_j$.
5. Maintain $E[i][j]$ to be the count of the occurrences of Y as a subsequence in $X[i \dots j]$. Use base case $E[0][0] = 0$. Apply recurrence: $E[i][j] = E[i][\lfloor (i+j)/2 \rfloor] + E[\lfloor (i+j)/2 \rfloor + 1][j]$.
6. None of the above.

— E N D O F P A P E R —