# ECE GY 9343 Final Exam (2023 Spring)

**Name:** **NetID:**

---

Answer ALL questions. Exam is closed book. No electronic aids. However, you are permitted two cheat sheets, two sides each sheet. Any content on the cheat sheet is permitted.

Multiple choice questions may have **multiple** correct answers. You will get **partial credits** if you only select a subset of correct answers, and will get zero point if you select one or more wrong answers.

**Requirements for in-person students ONLY**

Please answer all questions on the question book. You should have enough space. Since we scan all the submissions in a single batch, anything written ON THE BACK SIDE WON'T BE SCANNED.

Make sure you write your NetID on the bottom of each page (abc1234). Don't tear off any page.

If you need extra scrape papers, we can give to you. However, it will not be graded.

If you don't have space left for you answers: 1. First try to use extra space on other pages, and wrote below the question that part of your solution is somewhere else and give us the page number; 2. If you still need extra space, use the rear side. When you submit, tell the proctor to make a notation on the first page of your answer book. If you don't do so, we won't be able to grade anything on the rear side; 3. DON'T write on any loose pages.

**Requirements for remote students ONLY**

Each student is required to open Zoom and turn on their video camera, making sure the camera captures your hands and your computer screen/keyboard. The whole exam will be recorded. To clearly capture the video of exam taking, it is recommended that: A student can use an external webcam connected to the computer, or use another device (smartphone/tablet/laptop with power plugged in). Adjust the position of the camera so that it clearly captures the keyboard, screen and both hands. During the exam, you should keep your video on all the time. If there is anything wrong with your Zoom connection, please reconnect ASAP. If you cannot, please email ASAP, or call your proctor.

During the exam, if you need to use the restroom, please send a message using the chat function in Zoom, so we know you left.

Please use a separate page for each question. Clearly write the question numbers on top of the pages. When you submit, please order your pages by the question numbers.

Once exam is finished, please submit a single PDF on Gradescope, under the assignment named "Final Exam". DEADLINE for submission is 15 minutes after the exam ended. Please remember to parse your uploaded PDF file. Before you leave the exam, it's your responsibility to make sure all your answers are uploaded. If you have technical difficulty and cannot upload 10 minutes after exam ended, email a copy to your proctor and the professor.

1. (**20 points**) **True or False**

   (a) **T or F:** To save time on determining whether an edge $(u, v)$ exists in a graph, one should use adjacency matrix, instead of adjacency list, to represent the graph.

   (b) **T or F:** Color of all vertices are black after breadth-first search from single vertex $s$ finishes.

   (c) **T or F:** In a directed acyclic graph, only vertices with in degree of zero can be the first vertex in the topological sorted order.

   (d) **T or F:** Based on the white-path theorem, if there is a link $(u, v)$ in the graph, $v$ will become a descendant of $u$ in a DFS tree, no matter where DFS starts.

   (e) **T or F:** For any optimization problem that can be solved with divide-and-conquer, there must be a dynamic programing based algorithm that solves the same problem.

   (f) **T or F:** The dependency graph of the subproblems in a dynamic programming problem must be a directed acyclic graph(DAG);

   (g) **T or F:** The most frequent symbol must have the shortest codeword, and the least frequent symbol must have the longest codeword in Huffman code for any set of symbols.

   (h) **T or F:** For a directed graph with a negative cycle, there can be no pair of vertices that has a well-defined shortest path cost.

   (i) **T or F:** If an edge $(u, v)$ is in a minimum spanning tree of $G$, the shortest path between $u$ and $v$ must be the direct path from $u$ to $v$ over edge $(u, v)$.

   (j) **T or F:** If P1 is polynomially reducible to P2, and P2 is a NP-Complete problem, then P1 must be also NP-Complete.

2. (**4 points**) Which of the following statements about depth-first search (DFS) are true?

   (a) DFS will visit all vertices, even if the graph is not fully connected;

   (b) DFS will generate shortest paths in a graph;

   (c) In a DAG, if there is an edge $(u, v)$, then $f(u) > f(v)$;

   (d) For an undirected graph, DFS tree has no cross edges;

   (e) None of the above.

3. (**4 points**) Which of the following graph problems CANNOT be solved in linear time ($O(|V| + |E|)$)?

   (a) Determining a topological sort of the vertices for a directed acyclic graph;

   (b) Determining if an undirected graph is connected;

   (c) Determining if an undirected graph has at least a loop;

   (d) Minimum spanning tree of a connected undirected graph;

   (e) None of the above.

4. (**4 points**) Which of the following sets of codewords can be Huffman codes?

   (a) [01, 11];

   (b) [00, 01, 1];

   (c) [000,001, 01, 1];

   (d) [1, 01, 000, 010];

   (e) None of the above.

5. (**4 points**) Shortest path can be well defined between a pair of vertices in which of the following graphs?

   (a) Any strongly connected digraph;

   (b) Any DAG;

(c) Any unweighted graph;

(d) Any undirected graph if all edge weights are nonnegative;

(e) None of the above.

**6**. (**4 points**) Which of the following statements are true?

(a) Halting Problem is not in NP;

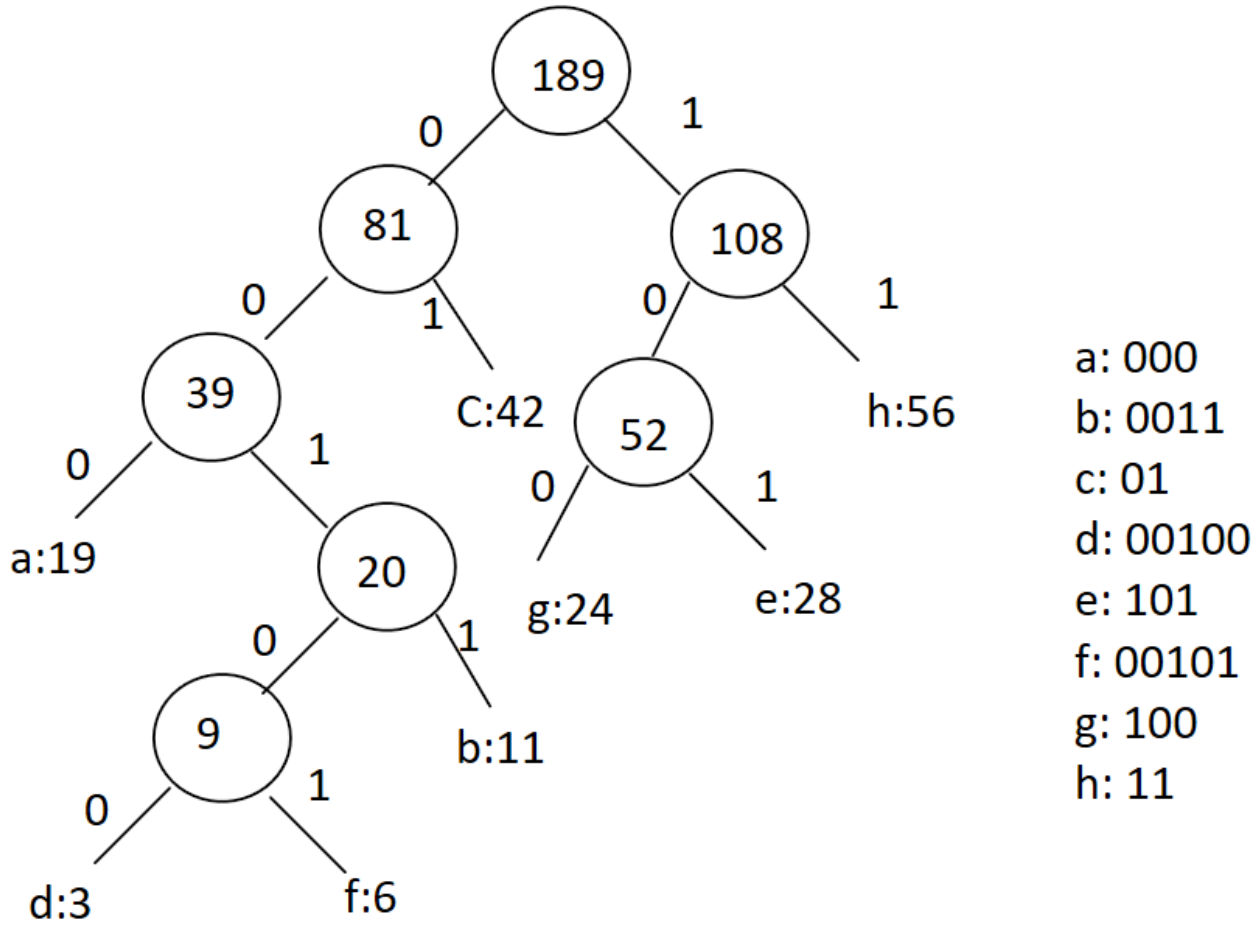
(b) NP-Complete problems are not in NP-Hard problem set;

(c) 2-SAT problem is NP-Complete;

(d) All NP-complete problems can be verified in polynomial time;

(e) None of the above.

**7**. (**5 points**) Construct the optimal Huffman code for the following set of symbols with associated frequencies: $\{a : 19, b : 11, c : 42, d : 3, e : 28, f : 6, g : 24, h : 56\}$




**8**. (**10 points**) Using Kosaraju's algorithm to find the strongly-connected-components for the graph in Figure 1. Assuming the adjacent list of each vertex is sorted in alphabetical order, and the first pass of DFS also considers vertices in alphabetical order. Show your results as the following:

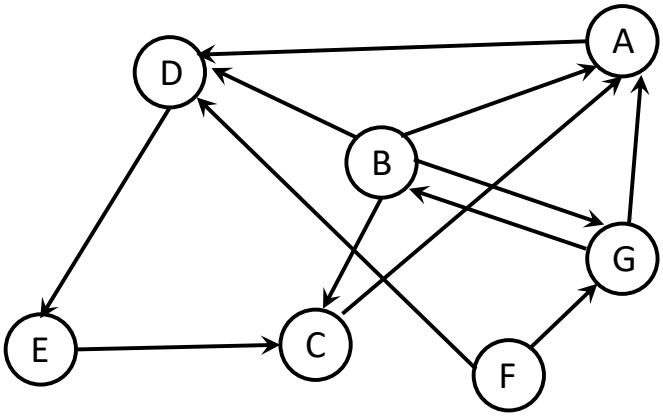


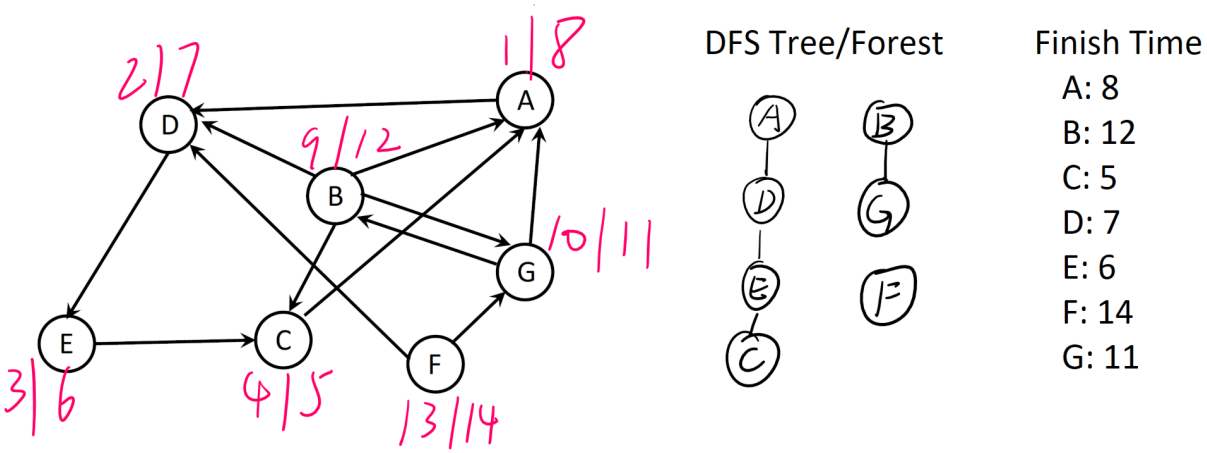

Figure 1: Directed Graph for Question 8

(a) (**4 points**) the DFS Tree/Forest after the first DFS call, the finish time for each vertex;

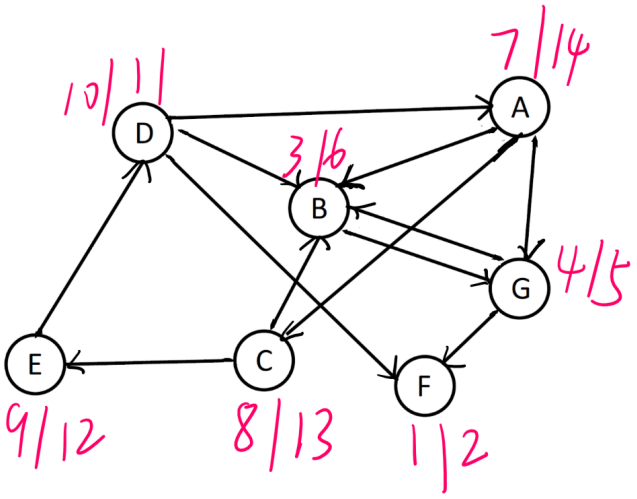
(b) (**3 points**) the DFS Tree/Forest after the second DFS call, the finish time for each vertex;

(c) (**3 points**) the Strongly-connected components and the component DAG.

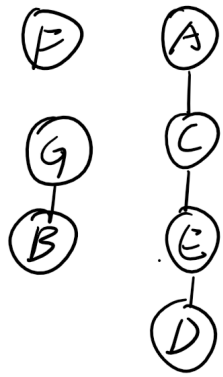
(a)




(b)




DFS Tree/Forest



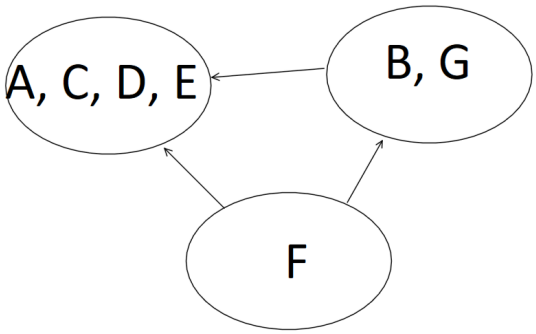

Finish Time

A: 14
B: 6
C: 13
D: 11
E: 12
F: 2
G: 5

(c)

SCCs: {A, C, D, E} {B, G} {F}

Component DAG




**9**. *(**6 points**)* Apply Dijkstra's algorithm to the graph in Figure 2 to construct the shortest paths from node A to all the other nodes.

(a) (**4 points**) show the steps of your calculation;

(b) (**2 points**) plot the shortest path tree rooted at $A$ at the end.

(a)

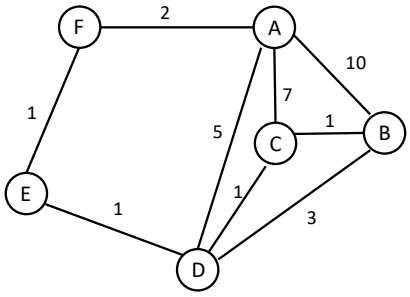
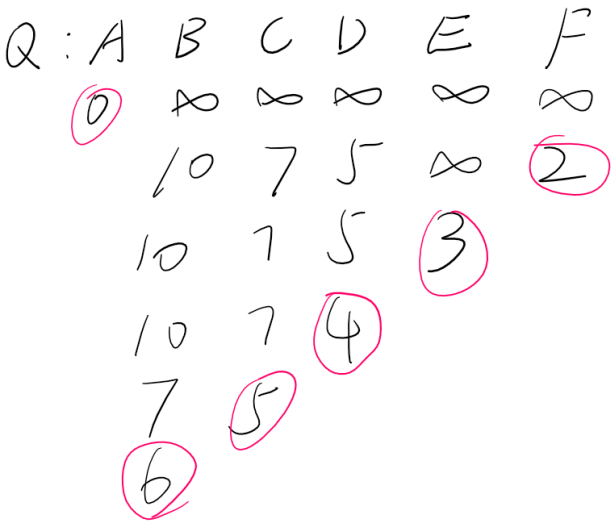



Figure 2: Undirected Graph for Question 9

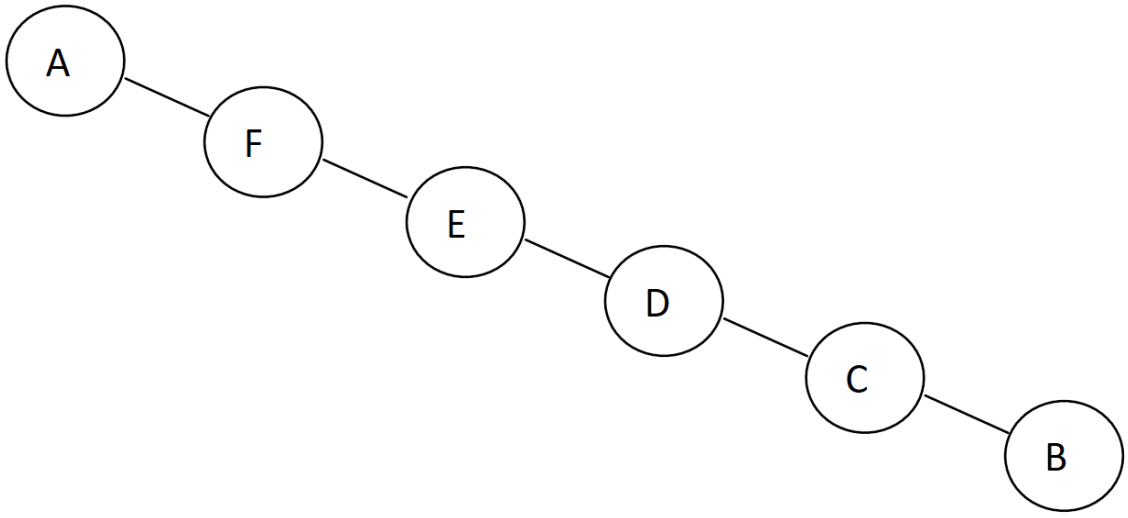



S:

{A}
{A, F}
{A, F, E}
{A, F, E, D}
{A, F, E, D, C}
{A, F, E, D, C, B}

S = {A, F, E, D, C, B}

(b)




**10**. **(7 points)** In Bellman-Ford shortest path algorithm,

(a) (**3 points**) given a network with $N$ nodes (including the source), how many iterations (the outer main loop) does it take for the Bellman-Ford algorithm to find the shortest paths from the source to all the other nodes? Briefly explain why more iterations are not needed.

(b) (**4 points**) let $d_k(A)$ be the shortest path distance estimate from the source to node $A$ after the $k$-th iteration, if $d_k(A) = 10$, what is the possible value range for $d_{k-1}(A)$? if the weight on link $\langle A \to B \rangle$ is 2, what is the possible value range for $d_{k+1}(B)$?

**Answer:**

(a) $N - 1$, the shortest path can have at most N-1 hops, each iteration increases the number of hops by 1, so at most N-1 iterations.

(b) $d_k(A)$ is a non-increasing function of $k$, so $d_{d-1}(A) \geq d_k(A) = 10$;

The estimated shortest path distance from the source to $A$ found after the $k$-th iteration has length of $d_k(A) = 10$. During the $(k+1)$-th iteration, when relaxing on link $\langle A \to B \rangle$, $d(A) \leq 10$, after the relaxation, $d(B) \leq d(A) + w(\langle A \to B \rangle) \leq 12$, so after the $k+1$-th iteration, the distance estimate for $B$ should be $d_{k+1}(B) \leq 12$.

**11**. (**12 points**) For a weighted undirected graph $G$, assume all the link weights are distinct, let $T$ be its minimum spanning tree. $AB$ and $CD$ are two links in $T$, and $A$, $B$, $C$, $D$ are four different nodes:

(a) (**2 points**) if we remove $AB$ and $CD$, how many sub-trees will $T$ be partitioned into?

(b) (**4 points**) if we group the nodes into subgraphs based on which sub-trees they belong to after $AB$ and $CD$ are removed, enumerate all the possible associations of $A, B, C, D$ with subgraphs. For example, $\{(A), (BC), (D)\}$ means $B$ and $C$ belong to the same subgraph, while $A$ and $D$ each belong to a different subgraph.

(c) (**6 points**) For the example association of $\{(A), (BC), (D)\}$, prove or disprove the following statements: *1) link AB has the lowest weight among all the links connecting the subgraph of A with the subgraph of BC; 2) link AB's weight is lower than any link connecting the subgraph of A with the subgraph of D;*

**Answer:**

(a) $T$ will be partitioned into three sub-trees;

(b) After $AB$ is removed, $A$ and $B$ will be in different subgraphs; likewise, after $CD$ is removed, $C$ and $D$ will be in different subgraphs, and there will be three subgraphs. So all the possible associations are:

- $\{(A), (BC), (D)\}$,
- $\{(A), (BD), (C)\}$,
- $\{(B), (AC), (D)\}$,
- $\{(B), (AD), (C)\}$.

(c) 1) **True.** If the association is $\{(A), (BC), (D)\}$, if only $AB$ is removed, $BCD$ will be in the same subgraph $G_{BCD}$; According to the greedy choice property of MST, link $AB$ has the lowest weight among all the links connecting the subgraph of $G_A$ with the subgraph of $G_{BCD}$. If both $AB$ and $CD$ are removed, the subgraph of $A$ is still $G_A$, while $G_{BCD}$ is further partitioned into two subgraphs $G_{BC}$ and $G_D$, the links connecting $G_A$ with $G_{BCD}$ is a subset of links connecting $G_A$ with $G_{BCD}$, therefore link $AB$ has the lowest weight among all the links connecting of $G_A$ with $G_{BC}$.

2) **True.** Proof by contradiction, if it is not true, let $T_A$, $T_{BC}$ and $T_D$ be the three spanning trees of the three partitioned sub-graphs, and $EF$ is the link connecting $G_A$ with $G_D$ and has lower weight than link $AB$, then link $EF + CD + T_A + T_{BC} + T_D$ form a spanning tree for the original graph that has lower weight than $T$, contradiction. (Similar proof can also be derived for case 1).

**12.** (**20 points**) For a sequence of numbers $\{A[1], A[2], \cdots, A[n]\}$, a subsequence is $\{A[i_1], A[i_2], \cdots A[i_k]\}$, where $1 \le i_1 < i_2 < \cdots < i_k \le n$,

(a) (**3 points**) If all the $A[i]$s are distinct, and take values from the range of $1 \cdots n$, please describe how to use the longest common subsequence algorithm we learned in Lecture 10 to find the longest subsequence such that $\{A[i_1] < A[i_2] < \cdots < A[i_k]\}$. (Note: $k$ is NOT a given constant). What is the complexity of your algorithm?

(b) (**5 points**) If $A[i]$s are not necessarily distinct, and can take arbitrary values, please describe how to use the longest common subsequence algorithm to find the longest subsequence such that $\{A[i_1] \le A[i_2] \le \cdots \le A[i_k]\}$. (Note: $k$ is NOT a given constant). What is the complexity of your algorithm?

(c) (**12 points**) If $A[i]$s are distinct, and can take arbitrary values, please design a dynamic programming algorithm to find the longest subsequence such that $\{A[i_1] < A[i_2] > A[i_3] < A[i_4] \cdots\}$, in other words, the first element of the subsequence is less than the second element, and the relative order between adjacent elements alternate between $<$ and $>$ for the rest of elements. Your algorithm should return the length of the longest subsequence and one longest subsequence. What is the complexity of your algorithm?

**Answer:**

(a) Define another sequence $B = \{1, 2, \cdots, n\}$, any increasing subsequence of $A$ is a common subsequence between $A$ and $B$, and vice versa. Therefore, the longest increasing subsequence of $A$ is just the longest common subsequence between $A$ and $B$, which can be found using the longest common subsequence algorithm in Lecture 10. The complexity is $\Theta(n^2)$.

(b) Sort $A$ into a non-decreasing sequence $A'$, any common subsequence between $A$ and $A'$ is a non-decreasing subsequence of $A$, and vice versa. Therefore, the longest non-decreasing subsequence of $A$ is just the longest common subsequence between $A$ and $A'$, which can be found using the longest common subsequence algorithm in Lecture 10. The sorting complexity is $\Theta(n \log n)$, so the total complexity is still $\Theta(n^2)$

(c) For a subsequence of $\{A[i_1] < A[i_2] > A[i_3] < A[i_4] \cdots\}$, the subsequence starts at the second element $\{A[i_2] > A[i_3] < A[i_4] \cdots\}$ also has alternating relative orders between the adjacent elements, but the first element is greater than the second element. For each position $i$, we find two types of subsequences that start with $A[i]$ and the relative orders between the adjacent elements alternate: 1) the `L-type` with $A[i]$ less than the second element; 2) the `G-type` with the $A[i]$ greater than the second element.

Let $L[i]$ and $G[i]$ be the lengths of the longest `L-type` and `G-type` subsequences starting at $A[i]$, respectively, then we have the following recurrences:

$$L[i] = 1 + \max_{j:j>i, A[j]>A[i]} G[j] \tag{1}$$

$$G[i] = 1 + \max_{j:j>i, A[j]<A[i]} H[j] \tag{2}$$

$$\tag{3}$$

Then, the problem can be solved in a bottom-up fashion:

```
L[1...n]=G[1...n]=1;
// length of the longest L and G type sequences starting at position i;
 L_next[1...n]=G_next[1...n]=n+1;
// the position of the second element for L and G type sequences

for i=n-1 downto 1
   for j=i+1 to n
      if A[j]>A[i] && L[i]<G[j]+1
         {L[i]=G[j]+1; L_next[i]=j;}
       if A[j]<A[i] && G[i]<L[j]+1
         {G[i]=L[j]+1; G_next[i]=j;}
```

```
// Find longest length
 max_len=1; max_i=n;
 for i=n-1 downto 1
      if L[i]>max_len
         {max_len=L[i]; max_i=i;}

// Print out longest subsequence
 L_Type=1; i=max_i;
 while i<=n
   {print A[i];
    if L_Type==1
              i=L_next[i];
        else
              i=H_next[i];
     L_Type=1-L_Type;
    }
  return max_len
```

Complexity: $\Theta(n^2)$,

++++++++++++++ **End of Exam** ++++++++++++++++

If you use up the space under any particular problem, you can write your answer here. On the page of the problem, tell us part of your work is here and write down the page number of this page and. Don't tear off any pages.

If you use up the space under any particular problem, you can write your answer here. On the page of the problem, tell us part of your work is here and write down the page number of this page and. Don't tear off any pages.