CSE 431/531 (Spring 2022) Final Exam (Slightly Modified)

Solutions and Grading Criteria

Student Name	: ______________	Instructor	: Shi Li
UBITName	: ______________	Student ID	: ______________

Problem	1	2	3	4	5	6	7	8	9	Total
Maximum Score	36	5	10	10	10	10	10	10	10	100
Your Score										

The sum of the maximum scores over all problems is 111. If you get more than 100 points, then your final score will be truncated to 100.

Remarks The purpose of examples given in the problem statements is to help you understand the problems. There may be multiple (optimum) solutions for the instances. It is possible and acceptable that your algorithm outputs a different one. So, do not use the given solution to guide your algorithm design.

It is recommended that you write your solutions in the exam paper, in the space after each problem. There should be enough space for the solutions. If not, you can write down your solutions in the blue book and mention this in the exam paper.

Some Standard Notations

- $\mathbb{Z}_{\geq 0}$: set of non-negative integers.
- $\mathbb{R}, \mathbb{R}_{\geq 0}, \mathbb{R}_{>0}$: sets of real numbers, non-negative real numbers, and positive real numbers.
- $[n]$ for an integer $n \geq 0$: the set $\{1, 2, 3, \cdots, n\}$.

Problem 1 (36 Points). Indicate if each of the following statements is true or false. A true/false answer for each statement is sufficient; you do not need to give proofs/counterexamples for your answers. 2 points for each problem.

(1.1) $\lceil \log_{1.1}(n^2) \rceil + 10 = O(\log_2 n)$. [✓ True] or False?

(1.2) Let $f, g : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be two asymptotically positive functions. Then at least one of the following two statements must be true: (i) $f(n) = O(g(n))$, and (ii) $g(n) = O(f(n))$. True or [✓ False]?

(1.3) If the recurrence of a running time T is $T(n) = 3T(n/2) + O(n^2)$, then solving the recurrence using the master theorem gives $T(n) = O(n^{\log_2 3})$. True or [✓ False]?

(1.4) A directed graph $G = (V, E)$ can be topologically sorted *if and only if* it does not contain a directed triangle. (A directed triangle is a sub-graph with 3 distinct vertices $u, v, w \in V$ and 3 directed edges $(u, v), (v, w)$ and (w, u).) True or [✓ False]?

(1.5) Consider the graph $G = (V, E)$ with $V = \{a, b, c, d\}$ and $E = (a, b), (a, c), (b, d)$, and the breath-first-search (BFS) algorithm over G with a being the starting vertex. Assume b is the first vertex visited after a. Then c is visited before d in the BFS algorithm. [✓ True] or False?

(1.6) Consider an instance of the interval scheduling problem. It is safe to schedule the job with the earliest finish time. It is also safe to schedule the job with the latest starting time. True or False?

(1.7) Consider the offline caching problem. The Least-Recently-Used (LRU) algorithm will always give the optimum solution. True or False?

(1.8) Consider the Huffman codes for an alphabet, and let a, b, c, d and e be 5 letters in the alphabet. (The alphabet may contain other letters.) Then it is possible that the codes for the 5 letters have lengths 1, 2, 3, 3 and 3 respectively. True or False?

(1.9) We have 5 stones with different weights. We have an unlabeled balanced scale which, given two stones, can tell which one is heavier. Then to sort the 5 stones from lightest to heaviest, any algorithm needs to use the scale at least 7 times in the worst case. True or False?

(1.10) There exists an $O(n)$-time comparison-based sorting algorithm (n is the size of the array to be sorted). True or False?

(1.11) Consider an instance of the weighted interval scheduling problem. It is safe to schedule the job with the smallest length-to-weight ratio. True or False?

(1.12) The running time of the dynamic programming algorithm for the longest common subsequence problem is $O(n)$. True or False?

(1.13) Let $G = (V, E)$ be a connected graph with edge weights $w : E \to \mathbb{R}_{>0}$, and assume all edge weights are distinct. Let C be a simple cycle in G, and e^* be the heaviest edge on C. Then, there is a minimum spanning tree T of G that *does not* contain e^*. True or False?

(1.14) Let $G = (V, E)$ be a directed graph with edge weights $w : E \to \mathbb{R}$ (weights can be positive, 0 or negative). Let $n = |V|, m = |E|$ and $s \in V$. It is guaranteed that G does not contain negative cycles. Then we can use Bellman-Ford algorithm to compute the shortest paths from s to all vertices in V, and its running time is $O(nm)$. True or False?

(1.15) Let $G = (V, E)$ be a directed graph with edge weights $w : E \to \mathbb{R}$ (weights can be positive, 0 or negative). Let $n = |V|$ and $m = |E|$. It is guaranteed that G does not contain negative cycles. We need to compute the length of the shortest paths between all pairs of vertices. Then, Floyd-Warshall's algorithm can solve the problem and it runs in $O(n^3)$ time. True or False?

(1.16) Assume $P \neq NP$ and $X \in NP$. Then X does not have a polynomial time algorithm. True or False?

(1.17) Assume $P \neq NP$. Then the circuit-satisfiability problem is not in P. True or False?

(1.18) It is possible that $P \cap NP = \emptyset$. True or False?

Problem 2 (5 Points). Use **the definition of the O-notation** to prove $5(n+\sin(n))^2 = O(n^2)$.

For every $n \geq 1$, we have $5(n + \sin(n))^2 \leq 5(n + 1)^2 \leq 5 \times (2n)^2 = 20n^2$. Therefore $5(n + \sin(n))^2 = O(n^2)$.

You need to use the formal definition of big-O notation to prove the statement. Most importantly, you need to show that the inequality holds *for every* $n \geq 1$ (1 can be changed to any other number.) Showing by calculation that $5(n + \sin(n))^2 < 20n^2$ for $n = 100$ is not sufficient.

Problem 3 (10 points). Consider the instance of the weighted interval scheduling problem in Figure 1, with $n = 11$ jobs. Each rectangle denotes a job, where its left and right sides denote its starting time and finishing time. The two numbers inside a rectangle denote its index and weight. For example, job 1 has weight 50, starting time 0 and finishing time 2. The jobs are already sorted in non-decreasing order of finishing times. If a job completes, another job can start immediately. For example, Job 1 and Job 5 are compatible. Recall the goal is to choose a subset of mutually-compatible jobs with the maximum total weight.

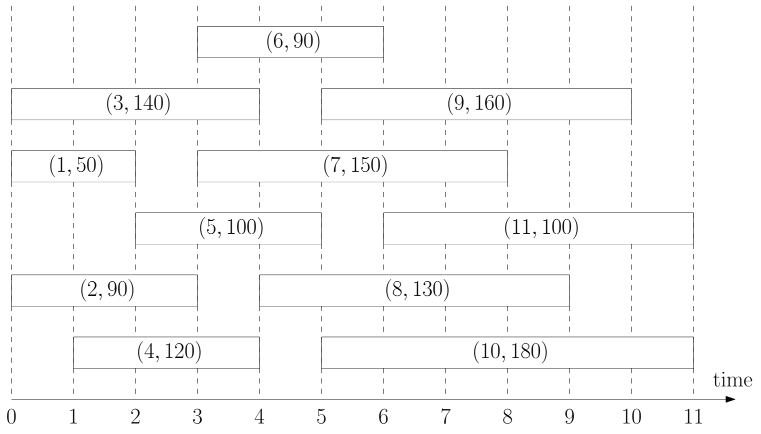

Figure 1: The Weighted Interval Scheduling Instance in Problem 3.

Give the optimum solution for the instance using the dynamic programming algorithm you learned in the class, by filling the underlined blanks below:

$opt[0] = 0$	$\pi[0] = \perp$
$opt[1] = \max\{opt[0], opt[0] + 50\} = 50$	$\pi[1] = \text{yes}$
$opt[2] = \max\{opt[1], opt[0] + 90\} = 90$	$\pi[2] = yes$
$opt[3] = \max\{opt[2], opt[0] + 140\} = 140$	$\pi[3] = yes$
$opt[4] = \max\{opt[3], opt[0] + 120\} = 140$	$\pi[4] = no$
$opt[5] = \max\{opt[4], opt[1] + 100\} = 150$	$\pi[5] = yes$
$opt[6] = \max\{opt[5], opt[2] + 90\} = 180$	$\pi[6] = yes$
$opt[7] = \max\{opt[6], opt[2] + 150\} = 240$	$\pi[7] = yes$
$opt[8] = \max\{opt[7], opt[4] + 130\} = 270$	$\pi[8] = yes$
$opt[9] = \max\{opt[8], opt[5] + 160\} = 310$	$\pi[9] = yes$
$opt[10] = \max\{opt[9], opt[5] + 180\} = 330$	$\pi[10] = yes$
$opt[11] = \max\{opt[10], opt[6] + 100\} = 330$	$\pi[11] = no$

The optimum set of jobs is {1,5,10}.

The final optimum set of jobs is worth 2 points. The remaining 8 points will be given based on which category your solution falls into:

Category	Explanation	score
Completely correct	all the *opt* and π values are computed correctly	8
Minor mistakes	there are 1-2 mistakes	6-7
Major mistakes	more than 2 mistakes, but solutions show understanding	3-5
Completely wrong	no understanding of the algorithm at all	0-2

Problem 4 (10 Points). You need to find the minimum spanning tree of the graph G in Figure 2, by filling the table in Figure 3. If an edge is not added to the MST, you do not need to mention that. So in the table, you only specify the edges that are added to the MST, in the order they are added, as well as how adding each edge changes the partition of vertices.

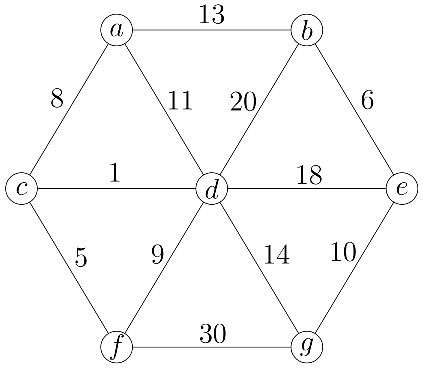

Figure 2: Instance for Kruskal's Algorithm in Problem 4.

i	i-th edge added to MST	partition of vertices after adding i-th edge
0	none	$\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{g\}$
1	(c,d)	$\{a\},\{b\},\{c,d\},\{e\},\{f\},\{g\}$
2	(c,f)	$\{a\},\{b\},\{c,d,f\},\{e\},\{g\}$
3	(b,e)	$\{a\},\{b,e\},\{c,d,f\},\{g\}$
4	(a,c)	$\{a,c,d,f\},\{b,e\},\{g\}$
5	(e,g)	$\{a,c,d,f\},\{b,e,g\}$
6	(a,b)	$\{a,b,c,d,e,f,g\}$

The total weight of the MST is 43.

Figure 3: Table for Problem 4.

The grading criteria is the same as that for Problem 3.

Problem 5 (10 Points). You are given a graph $G=(V,E)$ with $|V|=n$ and $|E|=m$. From the class you learned that the graph G is 2-colorable if and only if it is bipartite: A graph $G=(V,E)$ is 2-colorable if you can color V using two colors so that for every edge $(u,v)\in E$, the colors of u and v are different.

In this problem, some vertices $U\subseteq V$ are *pre-colored* using the two colors. Your goal is to decide if it is possible to complete the coloring to obtain a valid 2-coloring for G, without changing the colors of U. So, even if the graph G is bipartite, you may need to output "no" as the vertices in U may be incorrectly colored.

Assume the graph G is given using the linked-list-representation. The two colors are indexed by 0 and 1, and the partial coloring is given by an array *color* over V, where $\mathit{color}[u]\in\{-1,0,1\}$ for every $u\in V$: If $\mathit{color}[u]\in\{0,1\}$, then u is pre-colored with $\mathit{color}[u]$; if $\mathit{color}[u]=-1$, then u is not pre-colored.

Design an $O(n+m)$ time algorithm to solve the problem. You need to give the pseudo-code for solving the problem. **You do not need to show the correctness and the running time of the algorithm, if they are easy to see from the pseudo-code.**

Example. For the three instances in (a), (b) and (c) in Figure 4, your algorithm should output "no", "yes" and "no" respectively.

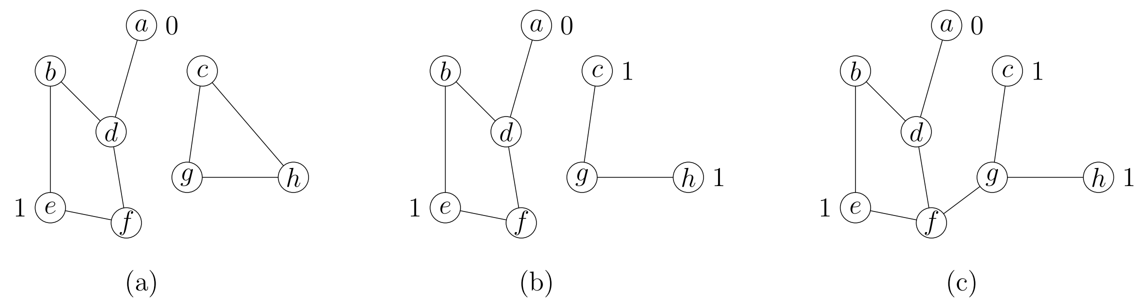

Figure 4: The three example instances for Problem 5 are denoted as (a), (b) and (c). The colors are given besides the circles. If there is no number besides a circle, then the correspondent vertex is not pre-colored. The *color* array for instance (a) is $d[a,b,c,d,e,f,g,h] = (0,-1,-1,-1,1,-1,-1,-1)$, and the *color* array for both (b) and (c) is $d[a,b,c,d,e,f,g,h] = (0,-1,1,-1,1,-1,-1,1)$.

For instance (b), we can complete the coloring to $\mathit{color}[a,b,c,d,e,f,g,h] = (0,0,1,1,1,0,0,1)$ to obtain a valid 2-coloring. So the output for instance (b) is "yes". The graph in (a) is not bipartite and so the output for (a) is "no". For the instance in (c), there is no way to extend the partial coloring to a valid 2-coloring, and so the output is also "no".

Algorithm 1 The Main Algorithm

```
1: for every $v \in V$ do $\mathit{visited}[v] \leftarrow \mathit{false}$
2: for every $v \in V$ do
3:     if $\mathit{color}[v] \neq -1$ and $\mathit{visited}[v] = \mathit{false}$ then DFS($v$)
4: for every $v \in V$ do
5:     if $\mathit{visited}[v] = \mathit{false}$ then $\mathit{color}[v] \leftarrow 0$, DFS($v$)
```

Algorithm 2 DFS(v)

```
1: $\mathit{visited}[v] \leftarrow \mathit{true}$
2: for every neighbor $u$ of $v$ in $G$ do
3:     if $\mathit{color}[u] = \mathit{color}[v]$ then
4:         output "no" and exit the whole algorithm
5:     else if $\mathit{visited}[u] = \mathit{false}$ then
6:         $\mathit{color}[u] \leftarrow 1 - \mathit{color}[v]$
7:         DFS($u$)
```

The breakup of the 10 points is as follows:

- The DFS/BFS framework is worth 5 points.
- The correctness is worth 3 points. If you follow the DFS/BFS framework and the correctness is easy to see, the 3 points will be given automatically.

- The correct running time is worth 2 points. Again, you do not need to prove the running time if this is easy to see from the pseudo-code.

If there are minor issues in the solution, 1 - 2 points will be deducted for each minor issue. Minor issue may include the following: the algorithm does not considering multiple connected components, it can not handle the case where a component does not contain any pre-colored vertices, etc.

Problem 6 (10 Points). You are given an array A of length n. For every integer $i \in [n]$, let b_i be the lower median of the sub-array $A[1..i]$. (If i is odd, the median is uniquely defined and the lower median is the median. If i is even, then there are two medians and lower median means the smaller of the two medians.) The goal of the problem is to output the medians $b_1, b_2, b_3, \cdots, b_n$ in $O(n \log n)$ time. **You do not need to show the correctness and the running time of the algorithm, if they are easy to see from the pseudo-code.**

Example. If $n = 10$ and $A = (50, 20, 10, 30, 90, 70, 40, 60, 80, 100)$. Then $b_1 = 50, b_2 = 20, b_3 = 20, b_4 = 20, b_5 = 30, b_6 = 30, b_7 = 40, b_8 = 40, b_9 = 50, b_{10} = 50$.

Hint. Use the heap data structure.

```
1: $L \leftarrow$ an empty max-heap, $L$.insert($-\infty$)
2: $R \leftarrow$ an empty min-heap, $R$.insert($\infty$)
3: for $i \leftarrow 1$ to $n$ do
4:     if $i$ is odd then
5:         if $A[i] \le R$.get_min() then
6:             $L$.insert($A[i]$)
7:         else
8:             $R$.insert($A[i]$)
9:             $L$.insert($R$.extract_min())
10:    else
11:        if $A[i] \ge L$.get_max() then
12:            $R$.insert($A[i]$)
13:        else
14:            $L$.insert($A[i]$)
15:            $R$.insert($L$.extract_max())
16:    $b_i \leftarrow L$.get_max()
17: Output $b_1, b_2, \cdots, b_n$.
```

This paragraph is not needed for getting a full score. The pseudo-code is sufficient. In the algorithm, we maintain two heaps: at the end of iteration i, the max-heap L contains the $\lceil i/2 \rceil$ smallest numbers in $A[1..i]$, and the min-heap R contains the $\lfloor i/2 \rfloor$ largest numbers in $A[1..i]$. To avoid the issue caused by empty queues, L also contains the $-\infty$ element, and R contains the ∞ element. Then at the end of iteration i, the median b_i of $A[1..i]$ is the largest element in L, which can be obtained by calling L.get_max(). The iteration i of the for loop handles the arrival of $A[i]$. Analysis of the running time: Every iteration of the for-loop takes $O(\log n)$ time. So overall the running time of the algorithm is $O(n \log n)$.

The breakup of the 10 points is as follows:

- The two-heap framework is worth 5 points.
- 3 additional points will be given if the algorithm can correctly handle all the cases.

- 2 points for achieving the correct running time.

Problem 7 (10 Points). Given a graph $G = (V, E)$, a *vertex cover* of G is a subset $S \subseteq V$ of vertices such that for every edge $(u, v) \in E$, we have $u \in S$ or $v \in S$. In the *minimum vertex cover on trees* problem, we are given a *tree* $T = (V, E)$ and the goal is to find the smallest vertex cover of T.

You need to design an efficient algorithm to solve the problem.

Example. In Figure 5, the smallest vertex cover of the tree T has size 5.

You can solve the problem using the following steps:

(7a) Design a safe strategy in which you choose some vertex v in the tree, and include v in the vertex cover.

(7b) Prove that the strategy you designed in step (7a) is safe.

(7c) Suppose you decided to put a vertex v in the vertex cover. Then, how do you reduce the remaining task into one or many sub-instances of the minimum vertex cover on trees problem?

If you do not follow the steps, and instead, you give a pseudo-code for the algorithm, then the pseudo-code will cover (7a) and (7c), but not (7b). You still need to prove the correctness by showing your strategy is safe.

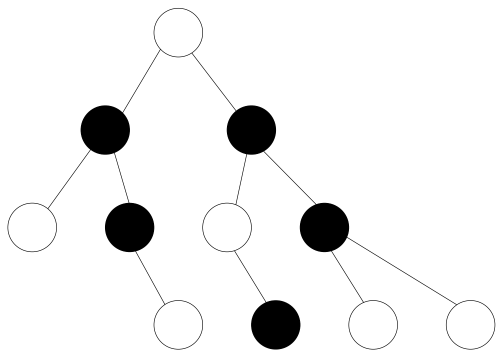

Figure 5: The minimum vertex cover of T. The minimum vertex cover is given by black circles. Every edge e in the tree has at least one black end vertex.

(7a) If T contains no edges, then nothing needs to be done. Otherwise, we take any leaf u in the tree T. Let v be the unique neighbor of u. Then we include v in the vertex cover.

(7b) The vertex cover needs to cover the edge (u, v). That is, either u is chosen, or v is chosen. Assume some optimum vertex cover S does not contain v. Then we must have $u \in S$. Then $S \setminus \{u\} \cup \{v\}$ is another optimum vertex cover for the instance. Therefore, in any case there is an optimum vertex cover that contains v.

(7c) After we choose the vertex v, we remove v and all its incident edges from the tree T. Then T becomes a forest. The remaining task is to solve the minimum vertex cover problem for each of the trees in the forest.

The breakup of the 10 points are as follows:

- The greedy strategy is worth 3 points.
- The proof of safety of the greedy strategy is worth 4 points.
- The description of the residual instance is worth 3 points.

This algorithm is based on greedy algorithm. It is recommended that you follow the guideline in the problem statement. If you give a pseudo-code, it will only cover (7a) and (7c). That is, the strategy and the residual problem. It does not cover the (7b), i.e., the proof that the greedy strategy is safe.

Problem 8 (10 Points). Suppose you are given n pictures of human faces, numbered from 1 to n. There is a face comparison program called same, that, given two different indices i and j from $1, 2, \cdots, n$, returns whether face i and face j are the same, i.e, are of the same person. A majority face is a face that appears more than $n/2$ times in the n pictures.

The problem is to find a picture with the majority face, or declare there is no majority face, by calling the program same. Your algorithm can only call same $O(n \log n)$ times. **You need to argue why your algorithm achieves this desired bounded, by giving a recurrence. If the correctness of the algorithm is easy to see, then you do not need to give a proof.**

Remark. The algorithm same can only return whether two faces i and j are the same or not. If they are not the same, it can *not* tell you whether "face $i <$ face j" or "face $i >$ face j".

Example. Suppose $n = 5$ and the function calls to same and their returned values are as follows: same$(1, 2)$ = false, same$(1, 3)$ = false, same$(2, 3)$ = true, same$(3, 4)$ = false and same$(3, 5)$ = true. Then your algorithm can return 2 (3 or 5) with confidence, since it knows that faces 2, 3 and 5 are the same.

Hint. Use divide-and-conquer and the following fact: If an element t is the majority in an array A, and we break A into two sub-arrays AL an AR, then at least one of the events happen: (a) t is the majority element in AL, and (b) t is the majority element in AR.

Algorithm 3 check-majority(l, r, t)

```
1: $count \leftarrow 0$
2: for $i \leftarrow l$ to $r$ do
3:     if same$(i, t)$ then $count \leftarrow count + 1$
4: return true if $count > (r - l + 1)/2$ and false otherwise
```

Algorithm 4 majority(l, r)

```
1: if $l = r$ then return $l$
2: $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$
3: $a \leftarrow$ majority$(l, m)$
4: $b \leftarrow$ majority$(m + 1, r)$
5: if $a \neq \perp$ and check-majority$(l, r, a)$ then return a
6: if $b \neq \perp$ and check-majority$(l, r, b)$ then return b
7: return $\perp$
```

In the main algorithm, we call majority$(1, n)$. It will return the majority face or $\perp$, which means that there is no majority face.

The recurrence for the number of times we call the *same* procedure is $T(n) = 2T(n/2) + O(n)$. Thus the algorithm calls *same* $O(n \log n)$ times.

The breakup of the 10 points is as follows:

- The divide-and-conquer framework is worth 5 points.
- 2 additional points will be given if all the details are correct.
- 3 points are for the number of calls to *same*. If the number is at most $O(n \log n)$, 1 point will be given. The remaining 2 points are given if the solution contains the correct recurrence.

There is an algorithm with $O(n)$ calls to *same*. But the correctness of the algorithm is not so obvious. So if you use that algorithm, you need to prove the correctness.

Problem 9 (10 Points). For each of the following problems, state (i) whether the problem is known to be in NP, and (ii) whether the problem is known to be in Co-NP. For the problem (9c), you need describe the certifier and certificate for the proof of the problem being in NP and/or Co-NP, if you answered yes. (For (9a) and (9b), you only give "yes/no" answers.)

(9a) Given a graph $G = (V, E)$ and an integer $k \geq 1$, the problem asks whether there is an independent set of size k in the graph G or not.

Problem known to be in NP? yes Problem known to be in Co-NP? no

(9b) Given a *tree* $T = (V, E)$ and an integer $k \geq 1$, the problem asks whether there is an independent set of size k in the tree T or not.

Problem known to be in NP? yes Problem known to be in Co-NP? yes

(9c) A boolean formula is said to be a *contradiction* if it evaluates to 0 for every assignment of 0/1 values to the variables. For example, $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$ is a contradiction. (In the formula, "$\vee$", "$\wedge$" and "$\neg$" stand for "or", "and" and "not" respectively.) Given a boolean formula with n variables (n is not a fixed constant), the problem asks whether it is a contradiction.

Problem known to be in NP? no Problem known to be in Co-NP? yes

Certifier for NP, if your answer for NP is yes: N/A

Certificate for NP, if your answer for NP is yes: N/A

Certifier for Co-NP, if your answer for Co-NP is yes:
The certifier takes the boolean formula, and an assignment of boolean values to the variables, and check if the formula evaluates to true on this assignment.

Certificate for Co-NP, if your answer for Co-NP is yes:
The certificate is the boolean assignment for which the formula evaluates to true.

Each yes/no answer is worth 1 point. Each of the certificate and certifier for NP is worth 2 points.

Problem 10 (10 Points). (This problem is based on dynamic programming and is not included in the exam. We include it here so that you understand how a DP algorithm is graded.) Given a sequence $A = (a_1, a_2, \cdots, a_n)$ of n numbers, we need to find the longest increasing sub-sequence of A. That is, we want to find a maximum-length sequence $(i_1, i_2, \cdots, i_t)$ of integers such that $1 \leq i_1 < i_2 < i_3 < \cdots < i_t \leq n$ and $a_{i_1} < a_{i_2} < a_{i_3} < \cdots < a_{i_t}$. You only need to output the length of the longest increasing sub-sequence. Design an $O(n^2)$-time dynamic programming algorithm to solve the problem.

Example. If the input $n = 11$, $A = (30, 60, 20, 25, 75, 40, 10, 50, 90, 70, 80)$, then the longest increasing sub-sequence of A is $(20, 25, 40, 50, 70, 80)$, which has length 6. The correspondent sequence of indices is $(3, 4, 6, 8, 10, 11)$.

You can solve the problem using the following steps:

(10a) Give the definition of the cells in the dynamic programming table.

(10b) Show the recursions for computing the cells.

(10c) Give the order in which you compute the cells.

(10d) Briefly state why the algorithm achieves the desired running time.

A pseudo-code in the solution will only cover (10b)-(10d), but not (10a).

(10a) We use $f[i]$ to denote the length of the longest increasing sub-sequence with the last element being $A[i]$.

(10b) The formula for computing $f[i]$ is as follows:

$$f[i] = \max_{j<i:A[j]<A[i]} f[j] + 1,$$

where we assume max is 0 if there is no $j < i$ satisfying $A[j] < A[i]$.

The final answer is $\max_{i\in\{1,2,\cdots,n\}} f[i]$.

(10c) We compute $f[i]$ for $i = 1, 2, 3, \cdots, n$, in this order.

(10d) The running time for computing each $f[i]$ is at most $O(n)$, and we need to compute n values of $f[i]$. So overall the running time is $O(n^2)$.

The breakup of the 10 points is as follows:

- The definition of cells is worth 3 points
- The recursion for computing the cells is worth 4 points.
- The order for computing $f[i]$ is worth 1 point.
- The running time is worth 2 points. If your algorithm achieves the desired running time, but you did not mention that it achieves the running time, you get 1 point.

The algorithm is based on dynamic programming. It is recommended that you use the guideline in the problem description to solve the problem. A pseudo-code in the solution will only cover (10b - 10d), but not (10a).