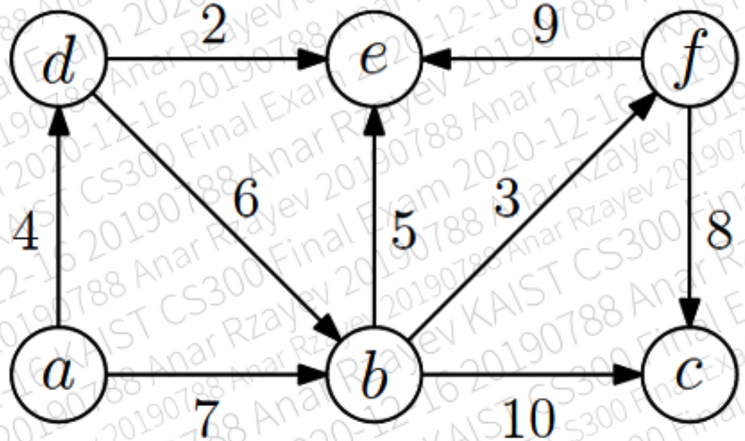
Run Dijkstra's algorithm on the weighted graph below, using vertex $a$ as the source. Then, answer the vertices in the order which they are marked.




(A) $a \to d \to b \to f \to e \to c$

(B) $a \to d \to b \to e \to f \to c$

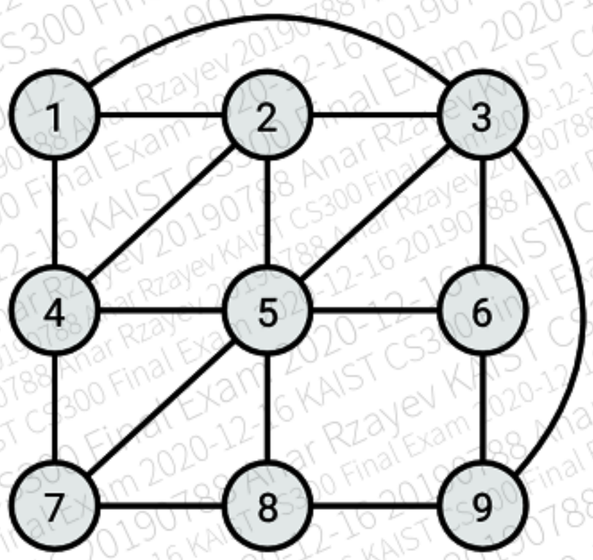
(C) $a \to d \to e \to b \to f \to c$

(D) $a \to d \to e \to b \to c \to f$

(E) $a \to b \to d \to f \to e \to c$

Suppose the following BFS pseudocode, that prints vertex numbers in order of visit, is executed with the given input graph. Which one of the following is impossible output?

```
1:  procedure BFS(G, s)                          ▷ Graph G = (V, E), undirected; vertex s ∈ V
2:      for all u ∈ V do
3:          dist(u) = ∞
4:      dist(s) = 0
5:      Q = [s]                                  ▷ (queue containing just s)
6:      while Q is not empty do
7:          u = eject(Q)
8:          print(u)
9:          for all edges (u, v) ∈ E do          ▷ (in arbitrary order)
10:             if dist(v) = ∞ then
11:                 inject(Q, v)
12:                 dist(v) = dist(u) + 1
```




(A) $1 \to 4 \to 2 \to 3 \to 7 \to 5 \to 6 \to 9 \to 8$

(B) $4 \to 2 \to 5 \to 1 \to 7 \to 3 \to 6 \to 8 \to 9$

(C) $5 \to 3 \to 2 \to 6 \to 4 \to 9 \to 1 \to 7 \to 8$

(D) $7 \to 8 \to 5 \to 4 \to 9 \to 6 \to 3 \to 2 \to 1$

For decision problems $L_1$ and $L_2$,
which of the following statement is proven to be TRUE?

(A) $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{co\text{-}NP}$

(B) $\mathbf{NP\text{-}complete} = \mathbf{NP} - \mathbf{P}$

(C) If $L_2 \in \mathbf{P}$ and $L_1 \leq_p L_2$, then $L_1 \in \mathbf{P}$.

(D) If $L_1 \in \mathbf{NP\text{-}hard}$ and $L_1 \leq_p L_2$, then $L_2 \in \mathbf{NP\text{-}complete}$.

(E) If $L_1 \in \mathbf{NP\text{-}complete}$ and $L_1 \leq_p L_2$, then $L_2 \in \mathbf{NP\text{-}complete}$.

What is the minimum number of CCTVs should we install on the road to make all cars pass the CCTV at least once?

The road's coordinate is given as -50 ~ 50. Based on the road's coordinate the routes of each car are given as follows:

**routes** = [[-16, 19], [3, 14], [12, 17], [2, 5], [11, 18], [-3, 4]]

- routes[*i*][0] is the point that car *i* enters the road
- routes[*i*][1] is the point that car *i* leave the road

Even if the CCTV is installed on the enter/leave point of the road, the car passed the CCTV.

(A) 1
(B) 2
(C) 3
(D) 4
(E) 5
(F) 6

Given a list $A$ consists of $n$ integers (i.e., $a_1, a_2, \ldots, a_n$), you need to pick a contiguous sub-list. For example, when $A$ is $10, 20, -15, 54, -3, -7$, then $10, 20, -15$ and $\phi$(an empty list) are valid sub-lists; but $10, -15, 54$ and $10, -3$ are not. The algorithm below finds the contiguous sub-list that has the largest sum of elements. For the example above, the answer would be $10, 20, -15, 54$.

Among the examples below, choose the one that best completes the blanks (a) and (b) in the algorithm.

---

**Input:** $n, a_1, a_2, \ldots, a_n$ (all inputs are positive integers)
**Output:** The contiguous sub-list that has the largest sum of elements

1: Initialize all $S[0, 1, \ldots, n] \leftarrow 0$, all $T[0, 1, \ldots, n] \leftarrow 0$
2: **for** $i = 1, 2, \ldots, n$ **do**
3: **if** $S[i-1] >$ (a) **then**
4: $S[i] \leftarrow$ (b)
5: $T[i] \leftarrow$T[i-1]
6: **else**
7: $S[i] \leftarrow a_i$
8: $T[i] \leftarrow i$
9: **end if**
10: **end for**
11: idx $\leftarrow \text{argmax}_x (S[x])$
12: **return** $a_{T[\text{idx}]}, \ldots, a_{\text{idx}}$ // 'empty list' if idx=0

| | (a) | (b) |
|---|---|---|
| A. | $S[0]$, | $S[i-1]$ |
| B. | $a_i$, | $S[i-1] + a_i$ |
| C. | $S[0]$, | $S[i - T[i-1]] + a_i$ |
| D. | $a_i$, | $S[i-1]$ |
| E. | $S[0]$, | $S[i-1] + a_i$ |
| F. | $a_i$, | $S[i - T[i-1]] + a_i$ |

Which of the following statements are TRUE?

(1) If a problem A is NP-Complete, there exists a non-deterministic polynomial time algorithm to solve A.
(2) The problem of determining whether there exists a cycle in an undirected graph is in P.

(3) The problem of determining whether there exists a cycle in an undirected graph is in NP.

(A) None of the above
(B) (1), (2), (3)
(C) Only (1)
(D) Only (2)
(E) Only (3)
(F) Only (1), (2)
(G) Only (1), (3)
(H) Only (2), (3)

Consider a Kruskal's algorithm which is implemented by using disjoint-set data structure. Which one of the following is **not** true?

(A) The height of the tree which the algorithm returns is $O(\log |E|)$.

(B) The algorithm doesn't allow a directed graph as an input.

(C) The worst-case running time of the algorithm is $O(|V| \log |E|)$.

(D) The algorithm works well with an undirected graph when weights of all edges are negative.

(E) The behavior of the algorithm depends on relative ordering of the weights of edges given on the input graph, not on the particular values of weights.

A *vertex cover* of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v)$ is an edge of G, then either $u \in V'$ or v $\in V'$ (or both). The size of a vertex cover is the number of vertices in it. The *vertex-cover problem* is to find a *vertex cover* of minimum size in a given undirected graph. There exists polynomial-time ___-approximation algorithm for vertex cover problem.

***Fill out the blank.***

(A) 3/2
(B) 8/7
(C) 2

**Analyze the time complexity of the given algorithms.**

---

function foo ( n )
if n == 1 : return 1
M( n ) = foo( n – 1 ) + foo( n – 1 )
return M( n )

---

A hash table, initially empty, holds values of M( n ) indexed by n

function bar ( n )
if n is in hash table: return M( n )
if n == 1 : return 1
M( n ) = bar( n – 1 ) + bar( n – 1 )
insert M( n ) into hash table, with key n
return M( n )

---

| | foo | bar |
|---|---|---|
| (A) | $\Theta(n^2)$ | $\Theta(n)$ |
| (B) | $\Theta(2^n)$ | $\Theta(n)$ |
| (C) | $\Theta(n)$ | $\Theta(2^n)$ |
| (D) | $\Theta(2^n)$ | $\Theta(2^n)$ |
| (E) | $\Theta(n)$ | $\Theta(n^2)$ |

Solving by referring to stored values to overlapping subproblems is known as:

(a)Dynamic programming
(b)Greedy
(c)Divide and conquer
(d)Recursion

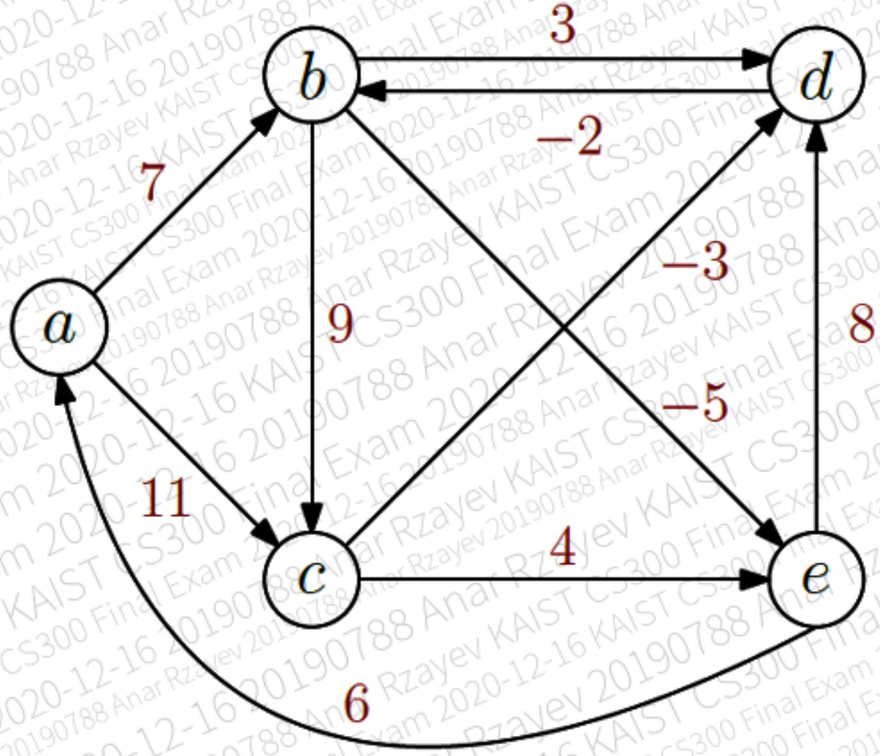
Choose all NP-complete problems among (A)-(C) given below. (P≠NP)

(A) : Problem is to find a truth assignment to variables that makes the formula in the form of 2-CNF true. For example, find assignments to make $f = (\neg x_1 \vee x_2) \wedge (\neg x_2 x_3) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2)$ true.

(B) : Given $n$ non-negative integers $w_1, ..., w_n$ and a target sum $S$, the problem is to decide whether such a subset of the items exists with total weight of $S$.

(C) : We have a set of n items $x_1, ..., x_n$, each with a weight $w_i$ and a cost $c_i$, given a maximum weight W, and a budget C. The problem is to decide whether a set of n variables $x_1, ..., x_n$ with $x_i \in \{0, 1\}$ exists such that $\sum_{i=1}^{n} x_i c_i \geq C, \sum_{i=1}^{n} x_i w_i \leq W$.

Assume that we apply the Bellman-Ford algorithm to the following graph $G = (V, E)$ with positive and negative weights $w(x)$ for each $x \in E(G)$:




| $v \backslash m$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $a$ | 0 | 0 | · | $L^{(4)}(a)$ |
| $b$ | 7 | 7 | · | $L^{(4)}(b)$ |
| $c$ | 11 | 11 | · | $L^{(4)}(c)$ |
| $d$ | $\infty$ | 8 | · | $L^{(4)}(d)$ |
| $e$ | $\infty$ | 2 | · | $L^{(4)}(e)$ |

Let $L^{(m)}(v)$ be the length of a shortest path from $a$ to $v$ where at most $m$ arcs are used (in other words, the length of a shortest $a, v$-path after the $m$ iterations). The values $L^{(m)}(v)$ can be found in the table above when $m = 1$ and $m = 2$. Then, answer the $L^{(4)}(a), L^{(4)}(b), L^{(4)}(c), L^{(4)}(d), L^{(4)}(e)$. (For example, if we answer $L^{(1)}(a), L^{(1)}(b), L^{(1)}(c), L^{(1)}(d), L^{(1)}(e)$, just enter "0, 7, 11, +infty, +infty")

Final exam of CS999 has $n$ questions, and you should solve the questions in order, similar to this final exam. If you try to solve a question $i$, you can get $p_i$ points. However, you will be exhausted after solving it and will be unable to solve any of the following $u_i$ questions. (i.e., if you solved question $\alpha(p_\alpha = 3, u_\alpha = 2)$, you can earn 3 points but you cannot solve the following 2 questions.) For this reason, you **can decide to skip some questions**.

Below algorithm is an algorithm to find the maximum point you can earn from the final exam of CS999. Select proper entries from the table below to complete the algorithm. **Write only the indices in order**. Since the answer for (a) is A and for (b) is C, **your answer must start with A,C,...** (e.g., A,C,B,D,E,F,G).

---

**Input:** $n, p_1, p_2, \ldots, p_n, u_1, u_2, \ldots, u_n$ (all inputs are positive integers)
**Output:** The maximum points you can earn from the final exam of CS999

1: Initialize all entries in [ (a) ] to [ (b) ] // Assume the length of $t$ is much larger than $n$
2: **for** $i =$ [ (c) ] **do**
3: $t[i] \leftarrow \max($ [ (d) ] $+$ [ (e) ] , [ (f) ] $)$
4: **end for**
5: **return** [ (g) ]

| | | | | | |
|---|---|---|---|---|---|
| **A.** $t$ | **B.** $z$ | **C.** $0$ | **D.** $1$ | **E.** $t[1]$ | **F.** $n-1, n-2, ..., 1$ |
| **G.** $t[i+1]$ | **H.** $p[i+1]$ | **I.** $u[i]$ | **J.** $t[i-u[i]]$ | **K.** $t[0]$ | **L.** $1, 2, ..., n-1$ |
| **M.** $t[i-1]$ | **N.** $p[i-1]$ | **O.** $t[i-u[i]-1]$ | **P.** $t[i+u[i]]$ | **Q.** $t[n]$ | **R.** $n, n-1, ...1$ |
| **S.** $t[i]$ | **T.** $p[i]$ | **U.** $t[i+u[i]+1]$ | **V.** $t[i+u[i]-1]$ | **W.** $t[n-1]$ | **X.** $1, 2, ..., n$ |

A boolean formula $\phi$ consists of $n$ variables, $\wedge$, $\vee$, $\neg$, and parentheses.
Define the following decision problems.

**P1** Given a boolean formula $\phi$, is there an assignment such that $\phi = \texttt{TRUE}$?

**P2** Given a boolean formula $\phi$, is there an assignment such that $\phi = \texttt{FALSE}$?

**P3** Given a boolean formula $\phi$, is $\phi = \texttt{TRUE}$ for all assignment?

**P4** Given a boolean formula $\phi$, is $\phi = \texttt{FALSE}$ for all assignment?

Assuming **P** = **NP** ∩ **co-NP** and **NP** ≠ **co-NP**, choose all TRUE statements among the following.
No explanation is necessary.

a **P1** ∈ **NP**

b **P2** ∈ **co-NP**

c **P3** ∈ **NP**

d **P4** ∈ **co-NP**

The code below uses dynamic programming to find the length of the longest common subsequence, which is defined as the longest subsequence that is common to sequences X and Y. The code below makes comparisons and fill the table accordingly. What built-in python function is f1?

```
1  def lcs(X , Y):
2      m = len(X)
3      n = len(Y)
4      L = [[None]*(n+1) for l in xrange(m+1)]
5      for i in range(m+1):
6          for j in range(n+1):
7              if i == 0 or j == 0 :
8                  L[i][j] = 0
9              elif X[i-1] == Y[j-1]:
10                 L[i][j] = L[i-1][j-1]+1
11             else:
12                 L[i][j] = f1(L[i-1][j] , L[l][j-1])
13     return L[m][n]
```

The weights and values of each item are given. You are finding the maximum total value in the knapsack of capacity W with the items. The number of each item is unbounded. Find the values a, b, c, d, e and f in the table. Write your answer in "a, b, c, d, e, f" format. (Write only commas between numbers, no alphabets and no special characters are allowed except for commas.)

| Item | Weight | Value |
|---|---|---|
| 1 | 6 | 30 |
| 2 | 3 | 14 |
| 3 | 5 | 24 |
| 4 | 2 | 9 |

| weight | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| max value | 0 | 0 | 9 | 14 | 18 | 24 | 30 | 33 | 39 | a | b | c | d | e | f |

Write TRUE or FALSE for the following statements.

(a) Dijkstra's algorithm does **not** work when there is one or more edges with zero weight.

( TRUE, FALSE )

(b) Floyd-Warshall algorithm runs in $O(V^3)$ time.

( TRUE, FALSE )

(c) Floyd-Warshall algorithm does **not** work for undirected graphs.

( TRUE, FALSE )

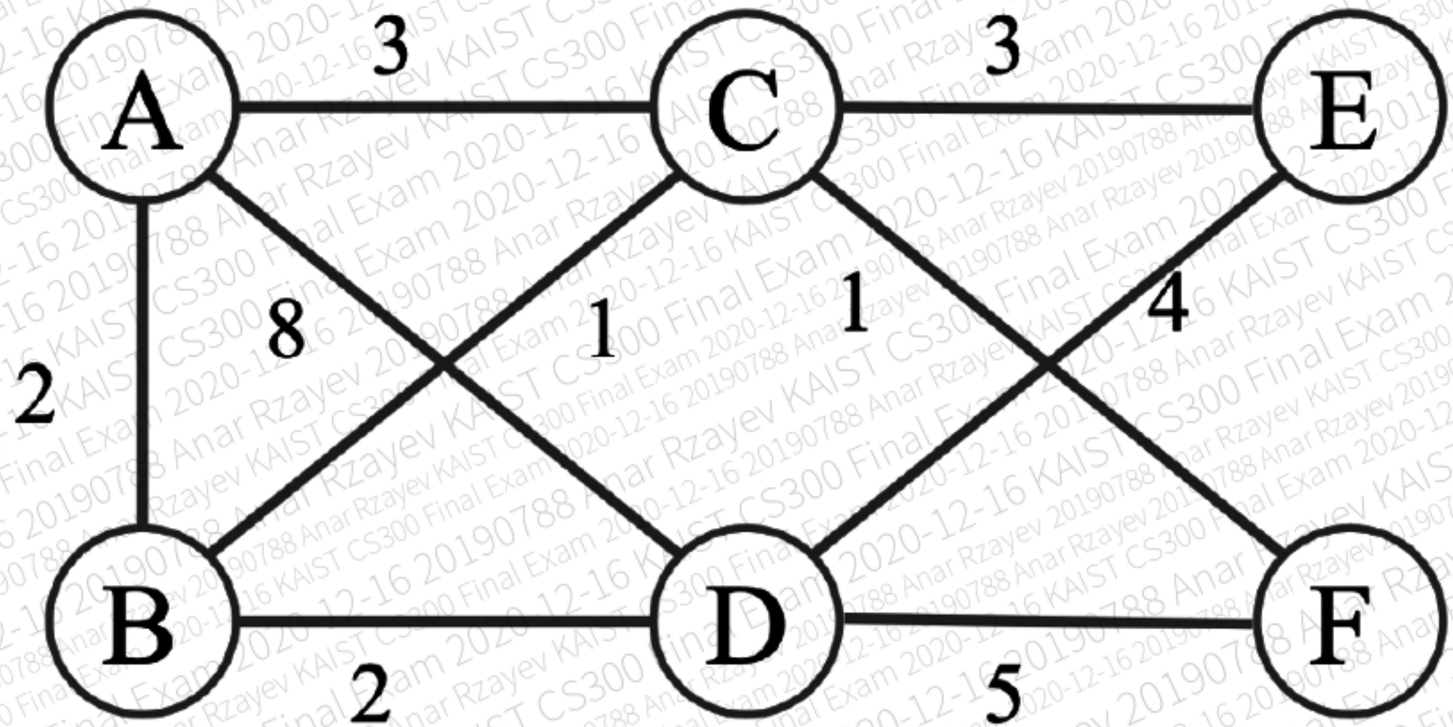
(d) Dijkstra's algorithm can be implemented without priority queue.

( TRUE, FALSE )

Specify if the below statement is TRUE or FALSE or NOT KNOWN. (You don't need to prove)

*Given n cities and intercity distances, the traveling salesman problem is to compute a tour that visits every city exactly once with minimum total length. There is no polynomial-time ρ-approximation algorithm for any constant $\rho \geq 1$ for the traveling-salesman problem.*

What is the MST (Minimum Spanning Tree) 's sum of edge weights?



Based on our lecture, select **all** the problems where the greedy approach can always find the optimal solution.

(A) Coin change problem

(B) Huffman coding problem

(C) Longest increasing subsequences

(D) Scheduling problem

(E) Fractional knapsack problem

Gridcity, a small and peaceful city, is well known for the coexistence of cats and people and its perfectly rectangular urban plan.

Since cats are serious about social distancing these days, there are frequent conflicts between cats due to overlapping territories. Gridcity's cats are so independent that they are ready to fight whenever the area of intersection of any two cats' territories is positive.

Meanwhile, as more people couldn't sleep because of the fights, the mayor of Gridcity decided to select some of the cats and move them to neighboring cities. Not surprisingly, the cats were opposed to leaving their hometown. After lengthy discussions, the mayor and cats agreed that only a minimal number of cats have to be moved, and one leader cat is always guaranteed to remain in Gridcity.

Suppose that Gridcity has width $W$ and height $H$, and the lower-left and upper-right coordinate of Gridcity is $(0, 0)$ and $(W, H)$, respectively. There are $n$ cats from index 1 to $n$ (inclusive), and the $i$th cat's territory is a rectangle whose lower-left corner is at $(A_i, 0)$, and upper-right corner is at $(B_i, H)$, and $0 \leq A_i < B_i \leq W$. Cats are sorted by $A_i$ in ascending order, and there are no duplicate coordinates. The leader cat is at index $k$, and $1 \leq k \leq n$. Coordinate values, lengths, and indices are all integers.

(1) [10 points] Find an $O(n)$ algorithm that returns a list of cats to move. Of course, your algorithm should comply with the agreement between cats and people. Write your algorithm in English, Korean, or pseudocode. You don't have to prove correctness.

*Input*: $n$ $k$ $W$ $H$ $A_{1\ldots n}$ $B_{1\ldots n}$ *Output*: an array of indices

(2) [5 points] Show that your algorithm runs in $O(n)$ time.

You are a broker who receives programming requests from companies and hires programmers to produce the requested programs. Your profit is *the money you get from companies minus the money to hire programmers*, and you should maximize this profit.

There are $n$ programmers. Each programmer $i$ has his/her rating of programming skill, $r_i$, and you have to pay $p_i$ dollars to hire him/her.

There are $m$ requests. Each request $j$ requires you to hire **a programmer** who has a rating of at least $R_j$, and you can earn $P_j$ dollars from it. You can decline some requests if you think they do not seem to bring money for you. Therefore, you should pick **a subset $\mathcal{A}$ of programmers and a subset $\mathcal{B}$ of requests to maximize your profit** $\sum_{j\in\mathcal{A}} P_j - \sum_{i\in\mathcal{B}} p_i$. Note that you can't assign more than one request to a programmer.

Give an $\mathbf{O}((\mathbf{n}+\mathbf{m})\mathbf{n}+\mathbf{m}\log\mathbf{m})$ **time algorithm to maximize your profit**. You must solve this question incrementally (**based on the given pseudo-code and the previous subquestions**). If you make a mistake in a subquestion, you won't be able to earn points on the subquestion after that.

---

**Input:** $n$; (for each $1 \le i \le n$) $p_i, r_i$; $m$; (for each $1 \le j \le m$) $P_j, R_j$; (all inputs are positive integers)
**Output:** Maximum profit you can earn (you don't have to return the subset $\mathcal{A}, \mathcal{B}$)

```
1: T, V ← an 1-D array of length (n + m + 1)
2: [ (a) ]  // You don't have to write pseudo-code for this one
3: [ (c) ]  // You don't have to write pseudo-code for this one
4: for i = 1, 2, ..., n + m do
5:     [ (d) ]
6: end for
7: return [ (e) ]
```

---

(a) (3 points) You can easily solve the **rating requirement** by merging and manipulating $p_i$ and $P_i$ on the array `V`. Write **how to merge and manipulate** the inputs within **two sentences**. You can use any temporal arrays if you need them. (Hint: look at the optimal time complexity)

(b) (4 points) You can use **only one 1-D array**, `T`, as a Dynamic Programming memoization table. Define the entries in your table $T$ within **one sentence**. (e.g., "`T[i]` is (tentative) largest number of programmers we can hire with `i` dollars")

(c) (1 point) Initialize your 1-D array `T`. Write you answer within **one sentence**.

(d) (6 points) Fill in the blank `(d)` to complete the pseudo-code. You can't use any arrays other than `T` and `V`. (n.b. the solution has multiple lines.)

(e) (1 point) Fill in the blank `(e)` to complete the pseudo-code.

Consider a set $X = \{x_1, \ldots, x_n\}$ and a collection $Y_1, Y_2, \ldots, Y_m$ of subsets of $X$ (i.e. $Y_i \subseteq X$ for each $i$). We say that a set $S \subseteq X$ is a *Special Set* for the collection $Y_1, Y_2, \ldots, Y_m$ if $S$ contains at least one element from each $Y_i$ (in other words, $S \cap Y_i \neq \emptyset$ for each $i$).

We now define the *Special Set Problem* as follows. We are given a set $X = \{x_1, \ldots, x_n\}$, a collection $Y_1, Y_2, \ldots, Y_m$ of subsets of $X$, and a positive integer $k$. We are asked: Is there a *special set* $S \subseteq X$ for $Y_1, Y_2, \ldots, Y_m$ so that the size of $S$ is at most $k$?

We want to show that *Special Set Problem* is NP-complete.

(a) (3 pts) Prove that *Special Set Problem* is NP.

(b) (12 pts) Prove that *Special Set Problem* is NP-hard. (Hint: You may use *Vertex Cover* for reduction).