Automatic Grading

All questions in this section are automatically graded. The order of questions and the options are randomized. Once you move beyond this section, you cannot go back.

For fill-in-the-blanks, please follow the format as precisely as possible. Especially for any questions on time complexity analysis, choose one of the following standard time complexities (without quotes):
"1",
"log **N**" (notice a space and capital 'N'),
"**N**",
"**N** log **N**" (notice two spaces and 'log' not 'lg' or 'ln'),
"**N**^2",
"**N**^2 log **N**" (notice a space), or
"**N**^3".
Do not write "O(" and ")" anymore as O(your-actual-answer) anymore as Big-O notation is already written. Remember that the grading by machine is very strict, so do not lose marks because of formatting issues.

1. Fill in the blanks

(5 marks)

You are given the following Python code. You can assume that the input consists of **N** positive integers that are read in the first line.

```
X = list(map(int, input().split())) # line 1
print(sum(sorted(X)))               # line 2
ans = 0                             # line 3
for i in range(len(X)):             # line 4
    for j in range(len(X)):         # line 5
        ans += sum(X[:i])*X[j]      # line 6
print(ans)                          # line 7
```

The time complexity of # line 1 is O(N).

The time complexity of # line 2 is O(N log N).

The time complexity of # line 3 to 6 is O(N^2).

Assume that the input to this Python code is "4 1 3 2"

The value printed by # line 2 is 10

The value printed by # line 7 is 170 .

For the first iteration in outer loop, the result is sum(X[:0]) * X[j]
Where sum(X[:0]) is 0.
For the second iteration of outer loop, the sum(X[:1]) is 4

Enter the correct answer below.

1

2

3

4 Please enter a number for this text box.

5 Please enter a number for this text box.

2. Select all the following Python (near) one-liner code that **really** run in O(**1**).

Partial marks based on correct and wrong.

(6 marks)

```
from collections import deque
print(Z.popleft()) # Z is a deque of N integers
```

```
print(1 if 7 in Z else 0) # Z is a list of N integers
```

```
print(Z.pop(0)) # Z is a list of N integers
```

```
from heapq import heappush, heappop
print(Z[0]) # Z is a list of N integers
# Z is modified only via heappushes and heappops
```

Equal to extractMin(), which is O(1) in min heap

```
sorted(Z) # Z is a list of N integers
```

```
print(1 if 7 in Z else 0) # Z is a set of N integers
```

In a set, searching an element costs O(1) since the set is implemented by Hash Table

Only I, IV, VI costs O(1)

3. Fill in the blanks

(4 marks)

Suppose the content of distance array/list is **D** (or Dist) = [0, 5, 7, 6, 10, 7, 11, 8] and predecessor (or parent) array/list is **p** = [-1, 0, 1, 1, 3, 3, 5, 5] after running modified Dijkstra's algorithm on a small positive-weighted graph with **V** = 8 vertices (labeled from 0, 1, ..., 7) and a certain source **s**. We know that **p[s]** = -1. Therefore, the source **s** must be vertex 0 and the shortest path from that source s to vertex **t** = 7 is therefore path: **s** - 1 - 3 - 5 - **t** = 7 with **D[t]** = 8.

Enter the correct answer below.

1 Please enter a number for this text box.

2 Please enter a number for this text box.

3 Please enter a number for this text box.

4 Please enter a number for this text box.

4.

Check all statements that are **True** about the Graph that we learned in class.

Partial marks based on correct and wrong.

(5 marks)

In an unconnected graph, it is not a tree

换句话来说，一棵有着N个顶点的树必然至少包含N-1条边，然而一个包含了N个顶点和N-1条边的图不一定是树，因为没有定义其是否是连通图

Every graph with **V** vertices and **V**-1 edges is always a tree.

False

When all vertices of a (simple) graph have in-degree and out-degree = V-1, then the graph must be a complete graph.

Every tree is also always a bipartite graph.

The Shortest Paths Spanning Tree of a directed weighted graph where all edges weight are **distinct** is always unique (assume that the source is vertex **s** = 0).

当所有边的权重都是唯一的时，对于任何给定的源顶点，都只会有一个最短路径生成树。这是因为不存在两条具有相同总权重的不同路径。

A connected graph with **V** vertices has at least **E** = **V**-1 edges.

Only II, III, IV, V are correct

5.

Select all the following Python (near) one-liner code that **really** run in O(**N** log **N**).

Partial marks based on correct and wrong.

(6 marks)

```
L.sort() # L is a list of N integers
```

```
L.reverse() # L is a list of N integers
```

reverse() need time in O(N)

False

```
from heapq import heappush
for Zi in Z: # Z is a list of N integers
    heappush(pq, Zi) # pq is a list of N integers used as min-heap
```

heappush() need time in O(log N), go through N integers, so total time in O(N log N)

```
for l in Z: # Z is a list of N integers
    if 77-l in Z:
        print(l, "+", 77-l, "=", 77)
```

for loop cost O(N), in the loop the 'in' operation cost O(N), hence total time cost O(N^2) — False

```
from heapq import heappop
Z = sorted(Z) # Z is a list of N integers
for _ in range(len(Z)):
    print(heappop(Z))
```

sorted() cost O(Nlog N)

heappop() cost O(log N), operate N times hence O(N log N)

Total time costs O(N log N)

```
from heapq import heapify
heapify(Z) # Z is a list of N integers that is heapified into a min-heap
```

False

heapify() cost O(N)

Only I, III, V are correct

6. The load factor alpha = N/M of a Hash Table can go above 1.0 if we use Separate Chaining collission resolution technique.

(2 marks)

- True
- False

7. Select all the following Python (near) one-liner code that **really** reverse the content Python list **L** containing **N** integers in O(**N**).

Partial marks based on correct and wrong.

(5 marks)

```
L = list(reversed(L))
```

```
L = L[::-1]
```

```
L.reverse()
```

```
L = reversed(L)
```

reversed() return a reverse iterator, which does not do the reverse operation immediately. Only when iterate this iterator, it will start reverse operation, like L=list(reversed(L)). For L = reversed(L) itself, it has time complexity O(1) — False

```
R = []
for Li in L:
    R = [Li]+R
L = R
```

R = [Li] +R cost O(N) since it append a new element at the front of the list, need to move all other element backword. This operation will take N times, so total time complexity is O(N^2) — False

Only I, II, III are correct

8. There are 5 names of popular sorting algorithms that are mentioned in class. One of the algorithm has one feature that is different from the other four algorithms (these four algorithms have the same other feature). Select this one algorithm.

(2 marks)

- Selection Sort
- Quick Sort
- Bubble Sort
- Counting Sort — Other four sorting algorithm is based on comparing, counting sort is based on counting
- Merge Sort

9. Fill in the blanks

(5 marks)

Complete the following Python code below with the exact required **one-word (no space)** syntax. It is supposed to read in a list **L** of **N** (**N** >= 3) **distinct** integers (positives, zeroes, and negatives) and deduce if there are 3 **distinct** integers that sums to 0 (by printing one of the possibly many 3 such integers in any order, e.g., if the input is "1 2 3 4 5 6 -7", then the answer can be "1 6 -7" - there are a few others) or not (by printing "No such triple", e.g., if the input is "1 2 3 4 5 6 7").

```
L = list(__map__(int, input().split())) # N distinct integers (+ve and -ve)
N = len(L)
found = __False__
for i in range(N-2):
    for j in range(i+1, N-1):
        for k in    range   (i+2, N):
            if not found and L[i]+L[j]+L[k] == 0:
                print(L[i], L[j], L[k])
                found = True
if not    found   :
    print("No such triple")
```

The time complexity of this algorithm is O(__N^3__).

Enter the correct answer below.

1

2

3

4

5

10. Check all statements that are **True** about the various Data Structures that we learned in class.

Partial marks based on correct and wrong.

(5 marks)

Only I, III, IV are correct

A Binary Search Tree with currently **N** = 15 elements can never be shorter than height = 3 (height = number of edges from root to the deepest leaf).	
(As of 2022), Python Standard Library has a built-in balanced Binary Search Tree implementation, called "OrderedDict" (imported from collections). OrderedDict is a subclass of dictionary, which maintains the insert element ordering, but it is implemented by hash table and DLL	False
A **Doubly** Linked List can be used to implement ADT Stack efficiently, i.e., top/push/pop all in time complexity of O(**1**).	
Collision resolution that may happen when we insert a value into a hash table can be resolved by either using Separate Chaining or Linear Probing technique.	
Inserting a new element that is greater than the maximum element currently in a Binary Max Heap with **N** > 31 elements will surely trigger more than 4 swaps.	False

When a new element is inserted into a binary max heap, the element will be inserted in to the bottom and maintain the max heap property upwards.
Consider a new element has the smallest value in the max heap, it will remain at the position that it inserted in. So no swaps trigger in this case

11. Check all statements that are **True** about Directed Acyclic Graph (DAG) and its topological ordering(s).

Partial marks based on correct and wrong.

(5 marks)

There exists a DAG with **exactly two** possible topological sorts only.	
Let {0, 1, 2, 3} be the topological order of a DAG with 4 vertices labeled with [0, 1, 2, 3]. We compute the shortest paths from source vertex **s** = 1. Without knowing the actual edge weights, we can conclude that the shortest path value from **s** = 1 to 0 is $D[0] = \delta(1, 0) = \infty$ (Infinity).	
There may have a path from 1 to 0. In topological orderings, the each vertex is ordered before its out-degree nodes.	False

The last vertex in any topological order of a DAG must have out-degree 0.

Suppose that a DAG **G** currently has **X** different topological orders. If we delete any single edge of **G**, the resulting graph DAG **G'** will definitely have **strictly greater than X** different topological orders.

False

Delete an edge may maintain the current number of topological order numbers, or decrease

The first vertex in any topological order of a DAG must have in-degree 0.

Only I, III, V are true

IQ Tests

Both questions in this section have easy and short solutions. Please think through them but do not burn too much time. There are two other longer application sections after this.

12. Earlier in the previous Section 1 (**now you cannot go back**), you were given the following O(**N**^3) Python code that read in a list **L** of **N** (**N** >= 3) **distinct** integers (positives, zeroes, and negatives) and deduce if there are 3 **distinct** integers that sums to 0 (by printing one of the possibly many 3 such integers in any order, e.g., if the input is "1 2 3 4 5 6 -7", then the answer can be "1 6 -7" - there are a few others) or not (by printing "No such triple", e.g., if the input is "1 2 3 4 5 6 7").

```
L = list(map(int, input().split())) # N distinct integers (+ve and -ve)
N = len(L)
found = False
for i in range(N-2):
    for j in range(i+1, N-1):
        for k in range(i+2, N):
            if not found and L[i]+L[j]+L[k] == 0:
                print(L[i], L[j], L[k])
                found = True
if not found:
    print("No such triple")
```

This algorithm is "too slow". Rewrite it (in Python) so that it is at least O(**N**^2) or better.

(12 marks)

Enter your answer here

See last page

Character Word 2000

13. There is an interview for a very lucrative job offer. There are **N** candidates in a queue waiting to be interviewed. This job is very competitive and everyone knows that they will not be selected if another candidate is **strictly better** than them.

So this is what each candidate does each minute: Look at the resume of candidate(s) who are currently adjacent to them in the queue (ahead and behind, possibly less than two neighbors if the candidate is at the very front or at the very back of the queue). If at least one of the adjacent neighbor has a perceived value that is greater than theirs, the candidate immediately leaves the queue (he/she knows it is not worth waiting for the interview). This process (of looking at neighbor(s)' resumes happen simultaneously and then some candidates leave the queue also simultaneously.

The question: What will be the **final** state of the list after candidate(s) no longer leave it?

Example 1: The interview queue initially has **N** = 7 candidates, and they lined up as follows [8 (front), 1, 2, 3, 5, 6, 7 (back)] - the integers describe the perceived values of each candidate. At the first minute, candidates with perceived values [1, 2, 3, 5, 6] all realized that they have at least one neighbor stronger than them, so they all left and only 8 (front) and (7) back stay at the moment. The queue is now [8 (front), 7 (back)]. At the second minute, candidate [7] realizes that candidate [8] is better, so [7] lefts, and the final queue is just [8 (front/back)].

Example 2: There are only **N** = 2 candidates, and both have the same perceived value of their own resumes. So, neither leaves the queue and both will go for the interview.

Design an algorithm (and also the required data structure(s)) to solve this problem as efficiently as you can and analyze your time complexity. You will get different marks depending on whether you can solve general case up to **N** ≤ 100,000 (the best known solution) or any other smaller range of **N**.

(7 marks)

Each candidate will check both front and back adjacency candidate's value, so it is a problem to find the global maximum. At the final state, the queue will only left candidates have the biggest value.

```python
def final_queue(candidates):
    if not candidates:
        return []

    # 找到最大值
    max_value = max(candidates)

    # 提取所有最大值候选人
    return [candidate for candidate in candidates if candidate == max_value]

# 示例
print(final_queue([8, 1, 2, 3, 5, 6, 7]))  # 输出 [8]
print(final_queue([2, 2]))  # 输出 [2, 2]

```

Enter your answer here

Character Limit: 2000

Application 1 (Second Last Section) - Dr stEVEN strange

Dr st**EVEN** is a a strange prof. He is obsessed with **even**ness. Perhaps because he is a Gemini (the Twins).

One interesting quirks of his obsession with even number is that he has to take **exactly even** number of roads to go from one place to another, despite that may cause him to detour a bit instead if he takes the normal shortest path.

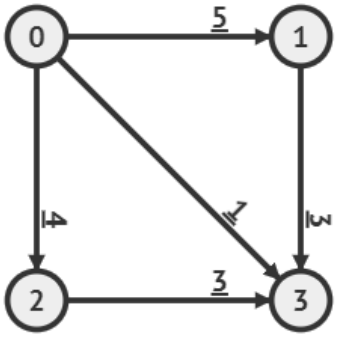
Steven has modeled the environment that he is in as a simple directed weighted graph with **V** vertices (labeled from 0 to **V**-1) and **E** directional weighted edges.

Steven wants to calculate the minimum total path length that he has to pay to go from location **s** = 0 to the last vertex **V**-1, obeying his obsession that he has to take an even number of edges along the path.

The first line of input contains two integers **V** and **E** (1 ≤ **V** ≤ 5,000, 0 ≤ **E** ≤ 10,000). The next **E** lines contains three integers: **u**, **v**, **w**, that means vertex **u** and **v** are connected with a directed edge with weight **w** (1 ≤ **w** ≤ 5,000).

Output an integer: the minimum path weight for Dr stEVEN to go from vertex 0 to vertex **V**-1, taking an even (0, 2, 4, 6, etc) number of directed edges along the path. However, if this is not possible, output '-1' instead.

For example, if given the following graph:



Then, dr stEVEN will take path 0->2->3 with minimum path weight of 4+3 = 7 as it uses an even (two) number of edges, despite there is a true shortest path 1->3 with path weight 1 (but only using 1 (odd) edge). Note than path 0->1->3 also uses an even number of edges, but it is longer (5+3 = 8).

14. In high level pseudo-code, how will you solve the general case of problem?

Mention the required (graph) data structure (if any), the name of the graph problem to be solved, the required (graph) algorithms that you will use to solve this problem (potentially with modifications), and finally analyze the time complexity of your proposed solution.

(12 marks)

It is a SSSP problem. The graph structure is Edge List, and using Dynamic Programming. The pseudo code is in the last page. The time complexity is O(VE).

Character Limit: 2000

Application 2 (The Last Section) - Special Fruit F

Warning: This is the hardest question in this paper. If you don't spend enough time on this section, you may not be able to complete them on time.

=======

There is a special fruit F that grows in a peculiar way on a field (simplified as a 2D grid with R rows and C columns and (0, 0) in the top left corner of the grid):

- F starts with four roots of length zero.
- Each of F's four roots grow a single unit in a different cardinal direction (North, East, South, and West) each day.

- When F dies (at any stage of its growth), its remains will not disappear.
- If *any* of the roots grow into another F or its roots – dead or alive – F will die at the end of that day.
- Roots cannot grow outside the bounds of the 2D grid. In other words, a fruit F will die if one of its roots tries to go outside the bounds of the 2D grid.
- If the roots of multiple Fs reach the same spot on the same day, each one of the affected roots stops growing (i.e., fight for nutrients) and in turn, those multiple Fs will die at the end of the day.
- When F dies, its roots do not grow on subsequent days.

With this information and the knowledge of where each of the seeds of fruit F were planted, output the following information: For each of the seeds of fruit F, print the day each seed dies (or a single string "STILL ALIVE") in one line if the fruits were left to grow for **D** days…

For example, if there are **X** = 5 seeds of fruit F, **D** = 3 days, and the 2D grid is of size **N**x**N** where **N** = 10, and the locations of the **X** = 5 seeds of fruit F are as of this initial grid, {seed A at (0, 0), B at (6, 3), C at (6, 4), D at (3, 6), and E at (2, 1)}. Lowercase ABCDE denote fruits that are still alive whereas the lowercase versions denote fruits that have died.

```
 0123456789
0A
1
2 E
3      D
4
5
6   BC
7
8
9
```

Then after the first day, we have this state, where:
-. lowercase a (0, 0) grows to east (0, 1) and south (1, 0) but then dies because the other two roots exit the 2D grid.
-. E and D grows to all 4 directions and still alive.
-. lowercase b and c both die as the east root of b collides with the west root of c.
-. So fruits a, b, and c all die on day 1.

```
 0123456789
0aa
1aE
2EEE   D
3 E   DDD
4      D
5 bc
6bbcc
7 bc
8
9
```

After the second day, we have this state, where:
-. lowercase a, b, c remains are still where they are.
-. lowercase e grows further to east (2, 4) and south (4, 1), but then dies as its north root collides with the remains of a and its west root exits the 2D grid.
-. So e dies on day 2.
-. D continues to grow to all 4 directions and the sole survivor so far.

```
 0123456789
0aa
1ae    D
2eeee  D
3 e  DDDDD
4 e    D
5 bc   D
6bbcc
7 bc
8
9
```

After the third (final day D = 3 of the simulation) day, we have this state, where:
-. lowercase a, b, c, e remains are still where they are.
-. D continues to grow to all 4 directions and the sole survivor so far.
-. so D still alive at Day D = 3.

```
 0123456789
0aa    D
1ae    D
2eeee  D
3 e DDDDDDD
4 e    D
5 bc   D
6bbcc  D
7 bc
8
9
```

The output for this task is therefore (in order of a, b, c, D, e):

```
1
1
1
STILL ALIVE
2
```

15. If there is only 1 seed in the entire 2D grid, this problem is **much simpler**.

Please come up with an O(**1**) algorithm to determine the required answer (that is, your algorithm should not do more than "a few constant steps").

(5 marks)

Enter your

```python
# Question 15
def solution(R, C, F, D):
    # R for num of row
    # C for num of col
    # F for pos of fruit
    # D for num of days
    if F.row()-0>=D and R-F.row()>=D and F.col()-0>=D and C-F.col()>=D:
        print("STILL ALIVE")
        return
    else:
        print(min(F.row()-0, R-F.row(), F.col()-0, C-F.col()))
        return
```

Character Limit: 2000

16. In high level pseudo-code, how will you solve the general case of problem given the following constraints: 1≤ **N** ≤ 1000 (that is, the 2D grid can be as big as 1000*1000 cells), 1≤ **D** ≤ 100 (the growth simulation can be up to 100 days), 1≤ **X** ≤ **N**^2 (there can be 1 - the easy subtask, 2, 3, ..., up to all **N**^2 seeds at the beginning).

Mention the required (graph) data structure (if any), the name of the graph problem to be solved, the required (graph) algorithms that you will use to solve this problem (potentially with modifications), and finally analyze the time complexity of your proposed solution.

(14 marks)

Enter your answer here

```python
# Question 16
def solution(R, C, F, D):
    # R for num of row
    # C for num of col
    # F for pos of fruit
    # D for num of days
    if F.row()-0>=D and R-F.row()>=D and F.col()-0>=D and C-F.col()>=D:
        print("STILL ALIVE")
        return
    else:
        print(min(F.row()-0, R-F# Question 16
def solution(N, X, D):
    # N for num of col & row
    # X for num of seeds and pos
    # D for num of days
    # 创建一个N*N的矩阵，全部初始化为未访问
    grid = create2DArray(N, N, "empty")
    # 对于每个种子，读取初始坐标放入矩阵中
    for (row, col) in X:
        grid[row][col] = "visited"
    # 记录种子的存活时间
    day = [None for i in range(len(X))]
    # 每天更新种子根的状态
    for i in range(1, D+1):
        for (row, col) in X:
            # 四个方向的位置
            right_dir = grid[row+i][col]
            left_dir = grid[row-i][col]
            up_dir = grid[row][col+1]
            down_dir = grid[row][col-1]
            all_dir = set(right_dir, left_dir, up_dir, down_dir)
            # 如果四个方向中有一个方向超出边界或触碰到其他的根，则死亡
            if not isInBoundary(all_dir) or freeBlock(all_dir):
                day[X] = i
                remove this seed from X (turn it into null)
            else:
                right_dir, left_dir, up_dir, down_dir = "visited"
    for i in range(len(X)):
        if day[X] == None:
            day[X] = "STILL ALIVE"
    print(day).row(), F.col()-0, C-F.col()))
        return
```

Character Limit: 2000

```python
# Problem 12
L = list(map(int, input().split()))
L.sort()
N = len(L)
found = False

for i in range(N-2):
    if i > 0 and L[i] == L[i-1]:  # 跳过重复元素
        continue

    left, right = i + 1, N - 1
    while left < right:
        total = L[i] + L[left] + L[right]
        if total < 0:
            left += 1
        elif total > 0:
            right -= 1
        else:
            print(L[i], L[left], L[right])
            found = True
            # 消除相同项，确保找到的是不重复的三元组
            while left < right and L[left] == L[left + 1]:
                left += 1
            while left < right and L[right] == L[right - 1]:
                right -= 1
            left += 1
            right -= 1

    if found:
        break

if not found:
    print("No such triple")
```

```python
# Question 14
def solution(EdgeList, V, E):
    i = len(V)
    j = 2
    # 创建一个dp二维数组，其中i为节点数量，j为2表示奇偶数
    dp = create2DArray with dimension [i][j]
    # 将二维数组中的每个元素初始化为无限大
    for each element in dp:
        element = INF
    # 起点到起点距离为0
    # 只初始化偶数边选项，因为起始-起始需要0条边，在后续的计算中，
    # 使用此选项加上出发的边可以得到奇数边
    dp[0][0] = 0
    # 对每个节点检测
    for i in range(1, V+1):
        for (u, v, w) in EdgeList:
            # Relax even edge
            # 即从出发节点的奇数边选项+边=偶数边
            if dp[u][1] + w < dp[v][0]:
                dp[v][0] = dp[u][1] + w

            # Relax odd edge
            if dp[u][0] + w < dp[v][1]:
                dp[v][1] = dp[u][0] + w

    if dp[V-1][0] == INF:
        return -1
    else:
        return dp[V-1][0]
```