Exam 2 CS-GY 6033 INET Spring 2025
April 14th 2025

Instructions:

Scheduling:

- The exam runs from 6pm to 9pm Eastern time, on April 14th 2025. Your exam is run through Gradescope, and will available to download at 5:50pm. The exam is to be completed by 9:00pm. Gradescope stops accepting uploads at 9:15pm. There are absolutely no late uploads accepted by the system after that time. The time to complete the exam will depend on your preparation: a prepared student could finish it in less than two hours, an ill-prepared student might take up to three hours. It is your responsibility to allow time for uploading your exam.

Format:

- The written exam consists of a total of 100 possible points, and will be graded out of 90 points. Your oral quiz counts for 10 points, making a total of 100 points for exam 2.
- **Submission Penalty: if you do not submit your exam on Gradescope, or do not properly assign your pages, you receive a deduction of 3 points on your exam.**
- You may write your solutions directly on the downloaded exam paper, *or* in your own format. You are responsible for providing clear and legible solutions to the problems. Your exam must be resubmitted into Gradescope electronically. Ensure that you know how to quickly upload any handwritten material. This is entirely the student's responsibility. You may assign your pages after the deadline.

Questions during the exam:

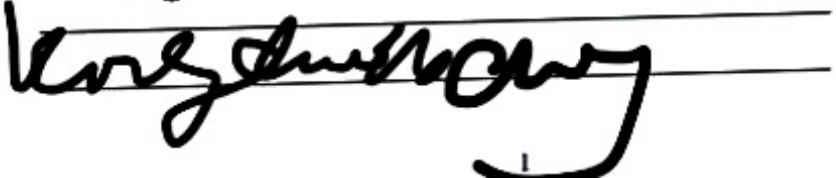
- There is a ZOOM session for questions that will be open during the entire course of the exam (microphones OFF). You may ask questions with private chat during the exam. Any announcements made by the instructor during the exam will be made over ZOOM and also be email. **It is the student's responsibility to stay connected (either by ZOOM or email) during the exam.**

Rules:

- This exam is a **take-home exam**. You may use **only** the resources from the online class (any material on NYU classes for this course) and any type of calculator (although it is not needed).
- Your work must be entirely your own. It is **forbidden to discuss any work with *any* other person**. Furthermore, your work must be done without using internet searches (although this is completely unhelpful for this exam). Any breach of academic honesty will be handled in accordance with the *Student Code of Conduct*, (a copy of which is provided), and in this particular case, taken very seriously.
- You are asked to **read** the attached Student Code of Conduct Section III subsections A,B,C,D,E and **sign** below to acknowledge that you aware of the policy. Once signed, a copy of this page must be uploaded with your exam.

I acknowledge that my submitted Exam work is entirely my own. I have read and am in accordance with the Student Code of Conduct policy of NYU Tandon and fully accept the consequences of breaching the above instructions.

Name: **Hongdao Meng**

Signature: 

Q1a. ~~Yes~~ NO, adjustment involve only local rotation and color changes, don't change the color of the root node, which means black height not change.

Q1b. Return [20, 22]

Q1c. largest black-height is 4 $\because 2^4-1 < 24 < 2^8-1$ (All black)

$15 < 24 < 255$

smallest black-height is 3 $2^b-1 < 24 < 2^{2b}-1 \quad b=3$

$7 < 24 < 63$

Q1d. BST Pre-order traversal is Unique. (root → left → right)

BST satisfied left subtree < ~~right subtree~~ root and root < right.

So the BST structure is unique. Different BST structures must correspond to different preorder traversal results;

∴ Unique.

Q1e. 4. k = BST-rank (T, x). Return BST-Select (T, k+1)

6、Successor (T, x)

Q1f. W[4] = 1 ∵ 0~6 : all True

W[2] = 2 ∵ 2, 4, 6 = True

Q1g. w[4] = 1 ∵ DP[3][1] = 6 DP[4][2] = 6 DP[4][3] = 7

V[4] = 6 ∵ weight = 1, i = 4 value = 6 from 0 (i=3)

Q1h.

$$r[8] = \max\{p_8, p_1 + r(7), p_2 + r(6), p_3 + r(5), p_4 + r(4), p_5 + r(3), p_6 + r(2), p_7 + r(1)\}$$

$$= \max\{15, 24, 25, 19, 18, 13, 25, 12\} = 25$$

$\because$ 25 is $p_2 + r(6)$

$s[8] = 2$

Q1j. $n = 15$

$s[15] = 2 \quad n = 13$

$s[13] = 2 \quad n = 11$

$s[11] = 11 \quad n = 0$

$\therefore$ 15 rod is optimally cut into 2, 2, 11.

Q1k.

2. $A[1] = B[6]$

4. $A[6] = B[3]$

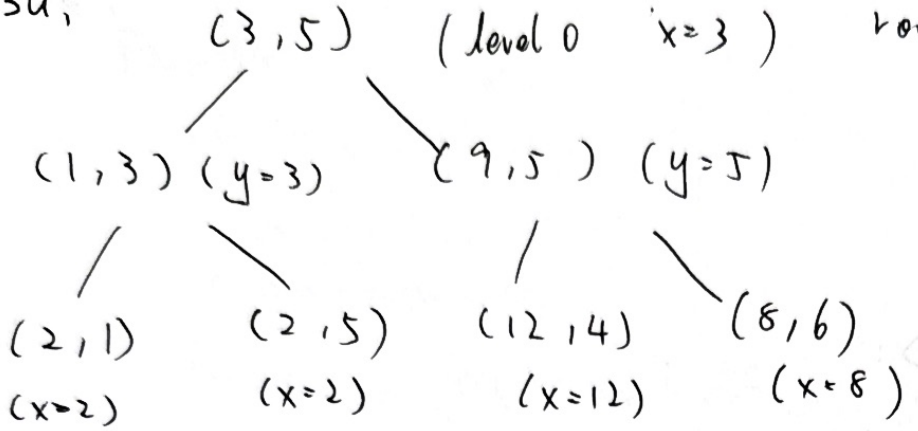
Q2.

```
CountRed(T):
    if T is NULL:
        return 0
    current_count = 0
    if T.color == RED:
        current_count = 1
    left_count = CountRed(T.left)
    right_count = CountRed(T.right)
return current_count + left_count + right_count.
```

∵ Traversal visit each node — number of nodes is n.

∴ Total Runtime : $O(n)$

Q3a.



3b.

```
Insert(T, x, level=0):
    if T is Null:
        return x
    if level % 2 == 0:
        if x.xcoord < T.xcoord:
            T.left = Insert(T.left, x, level+1)
        else:
            T.right = Insert(T.right, x, level+1)
    else:
        if x.ycoord < T.ycoord:
            T.left = Insert(T.left, x, level+1)
        else:
            T.right = Insert(T.right, x, level+1)
    return T
```

Q3c.

```
FindMinX(T, ~~level~~):
    if T is Null:
        return NIL
    ~~if level is even:~~
    left_min = FindMinX(T.left)
    right_min = FindMinX(T.right)
    current_min = T
    if left_min is not NULL and left_min.xcoord < current_min.xcoo
        current_min = left_min
    if right_min is not NULL and right_min.xcoord < current_min.xcoo
        current_min = right_min
    return current_min
```

Q4a. A pratical way to guarantee $O(\log n)$ is to implement the BST as a Blanced BST, Like a Red-Black Tree. Height is always $\log n$, so search, insert and delete takes $O(\log n)$ times.

∵ inserts take $O(\log n)$ and n nodes

∴ Total Runtime of building is $O(n \log n)$

Q4b.
```
TreeInsert(T, x):
    if T is None:
        x.left = None    x.right = None    x.size = 1
        x.totaltime = x.time    x.maxage = x.age    x.mintime = x.time
        return x
    if x.time < T.time:
        T.left = TreeInsert(T.left, x)
    else:
        T.right = TreeInsert(T.right, x)
    T.size = 1 + getsize(T.left) + getsize(T.right)
    T.totaltime = T.time + get_totaltime(T.left) + get_totaltime(T.right)
    T.maxage = max(T.age, get_maxage(T.left), get_maxage(T.right))
    T.mintime = min(T.time, get_mintime(T.left), get_mintime(T.right))
    return T.
```

Q4c.

```
FastestTime(T):
    if T is Null:
        return NULL
    if T.left == NULL:
        return T
    else:
        return FastestTime(T.left)
```

Runtime:

∵ Tree height is logn

∴ Insert and delete keep tree's balance

∴ O(logn)

Q4d.

```
MaxAge(T, k):
    if T is Null:
        return -infinity
    if T.time >= k:
        return MaxAge(T.left, k)
    else:
        left_max = T.left.maxage if T.left else -infinity
        current_max = max(T.age, left_max)
        right_max = MaxAge(T.right, k)
        return max(current_max, right_max)
```

Height of tree is O(logn)

each maxage is O(1)

∴ Runtime is O(logn)

Q4e.

```
AthletesOverTime(T, k):
    if T is NULL:
        return 0
    if T.time > k:
        return 1 + AthletesOverTime(T.left, k) + AthletesOverTime(T.right, k)
    else:
        return AthletesOverTime(T.right, k)
```

∵ Height is O(logn) size is O(1)

∴ Runtime is O(logn)

Q4f.

```
TotalTimeOver(T, k):
    if T is NULL:
        return 0
    if T.time > k:
        right_total = T.right.totaltime if T.right else 0
        return T.time + right_total + TotalTimeOver(T.left, k)
    else:
        return TotalTimeOver(T.right, k)

AverageTimeOver(T, k):
    count = AthlestesOverTime(T, k)
    if count == 0:
        return 0
    total = TotalTimeOver(T, k)
    return total / count
```

∵ Height is O(logn)
each node is visited one time.
∴ Runtime: O(logn)

Q5

$T[i,j]$ is the minimum total delay of any path from your house to traffic-light (i,j)

$\min T[i,n]$ will give the least expected total delay.

Initialization:

$T[i,1] = D[i,1]$ for each i.

Because the house has directed edges to every node in the first column.

Relation:

For each column j from 2 to n, each row i from 1 to m.

$$T[i][j] = D[i][j] + \min \begin{cases} T[i][j-1] \\ T[i-1][j-1] & \text{if } i \geq 1 \\ T[i+1][j-1] & \text{if } i \leq m \end{cases}$$

Presudocode:

```
Find MinTime(D, m, n):
    T = [[0 for _ in range(n+1)] for _ in range(m+1)]

    for i from 1 to m+1
        T[i][1] = D[i][1]

    for j in range(2, n+1):
        for i in range(1, m+1):
            min_prev = ~~float('inf')~~ infinity
            if i >= 1:
                min_prev = min(min_prev, T[i-1][j-1])
            min_prev = min(min_prev, T[i][j-1])
            if i < m:
                min_prev = min(min_prev, T[i+1][j-1])
            T[i][j] = D[i][j] + min_prev

    for i from 1 to m+1:
        result = min(result, T[i][n])
    return result.
```

Final Return the minmum of $T[i][n]$

Runtime: Initialize $O(m)$

Fill Form $O(mn)$

Recursive: $O(m)$

$\therefore O(mn)$

Q6. It's a Longest Common Subsequence Problem.

1. Sort each class by Height
2. define a 2D Table
   dp[i][j] is length of the longest common subsequence using the first i dancers from A and the first j dancers from B.
3. Recurrence
   For $i \geq 1$ and $j \geq 1$:

$$dp[i][j] = \begin{cases} dp[i-1, j-1] + 1, & \text{if } A[i].age = B[j].age, \\ \max(dp[i-1, j], dp[i, j-1]), \end{cases}$$

   initialize the zero row and column to 0.
4. result is dp[n][m]

∴ Each of n×m table is computed in O(1), total runtime is O(nm).

Q7

part a.

$d[i]$ is the maximum payment achievable by reaching mile marker i.

Initialization: $dp[0] = 0$ no markers visited

Table-Filling:

1. Don't pick rock at i:

$dp[i] = d[i-1]$

2. Pick rock at j

For j where $j + D[j] = i$ and $j + D[j] \leq n$

$dp[i] = \max(dp[i], dp[j-1] + V[j])$

Result: $dp[n]$

Pseudocode:

```
maxpay(D, V, n):
  next_j = [[] for _ in 0 to n+1]
  for j in 1 to n+1:
    i = j + D[j-1]
    if i ≤ n:
      next_j[i].append(j)
  dp = [0] * (n+1)
  for i in 1 to n+1:
    dp[i] = dp[i-1]
    for j in next_j[i]:
      dp[i] = max(dp[i], dp[j-1] + V[j-1])
  return dp[n]
```

Runtime: Each i checks $O(1)$, preprocessing takes $O(n)$ to map j to i.

$\therefore$ Total is $O(n)$

Part b.

Maintain a choice to record which rock j was picked at mile i.

```
print(D, V, n):
    next_j = [[] for _ in 0 to n+1]
    for j in 1 to n+1:
        i = j + D[j-1]
        if i ≤ n:
            next_j[i].append(j)

    dp = [0] * (n+1)
    choice = [None] * (n+1)

    for i in 1 to n+1:
        dp[i] = dp[i-1]
        for j in next_j[i]:
            if dp[j-1] + V[j-1] > dp[i]:
                dp[i] = dp[j-1] + V[j-1]
                choice[i] = j

    current = n
    picked = []
    while current > 0:
        if choice[current] is not None:
            j = choice[current]
            picked.append(j)
            current = j-1
        else:
            current -= 1
    picked.reverse()
    return dp[n], picked
```

Example output

[2, 6, 8] or

[2, 6, 9]