HW - ~~3~~ 4

a) Analyze time complexity of insertion Sort:

- insertion-Sort (A, n)
  - for (i = 1 to n) → runs N-1 times
    - key = A[i]
    - j = i - 1,
    - while [A[j] > key and j ≥ 0]] swapping occurs.
      - A[j + 1] = A[j]
      - j = j - 1
    - A[j + 1] = key

- **Best case**: Array is already sorted.
  - since there will be no swapping needed, while loop will not run. ∴ Best case = $\Omega(n)$ [only for loop runs N-1 times]

- **Worst case**: Array in decending order
  - Example A = [5, 4, 3, 2, 1]
  - We know for loop runs N-1 times, but what about the while loop?

i	while loop [swapping] X
1	1
2	2
3	3
⋮	⋮
n	n-1

• Follows $O(n^2)$

→ $1 + 2 + \dots (n-1)$

$\frac{n(n-1)}{2}$

→ $O(n^2)$

- ~~Since Best case ≠ N~~
- $\Theta(n)$ does NOT exist since $\Omega(n) \neq O(n^2)$

→ Time complexity = $O(n^2)$

HW-4

Part 2: Time complexity of multiplication Matrix:

- we know there are 3 for loops which run from 1 to #row_A, #cols_B, #cols of A.

① **Best case:** All these 3 numbers rA, cB, cA are really small.

1st for loop would run rA times
2nd for loop will run cB × rA times
3rd for loop will run cB * rA * cA times.

→ Lower bound = $\Omega(cA \times cB * rA)$
→ Let's say $n = \min(cA, cB, rA)$
→ Then = $\boxed{\Omega(n^3)}$

② **Worst case:** rA, cB, cA are very big values
We can say that $n = \max(cA \times cB \times rA)$, then worst case = $O(n^3)$ using same logic as lower bound.

③ Since lower bound = upper bound time complexity, we can say

$\Omega(n^3) \leq \Theta(n^3) \leq O(n^3)$

$\Omega(n^3)$: $n = \min(cA \times cB \times rA)$
$O(n^3)$: $n = \max(cA \times cB \times rA)$

→ Average time: = $\boxed{\Theta(n^3)}$