Luis Gascon, Ethan Webb, Femi Dosumu
COSC 336
February 16, 2023

Assignment 1

Exercise
Consider the following three program fragments (a), (b), and (c).

(a)
```
sum = 0;
for (int i = 0; i < n ; i++) {
        sum++;
}
```

(b)
```
sum = 0;
for (int i = 0; i < 2 * n ; i++) {
        sum++;
}
```

(c)
```
sum = 0;  i = n * n;
while (i > 1) {
        sum++;
        i= i/2;
}
```

We denote by $T_a(n)$, $T_b(n)$, $T_c(n)$ the running time of the three fragments.

1. Give Θ evaluations for $T_a(n), T_b(n), T_c(n)$.
   - $T_a(n) = \Theta(n)$
   - $T_b(n) = \Theta(n)$
   - $T_c(n) = \Theta(\log n)$

2. Is $T_b(n) = O(T_a(n))$? Answer YES or NO and justify your answer.

   Yes, $t_b(n) = O(T_a(n))$ because both $t_a(n)$ and $t_b(n)$ are $\Theta(n)$ $\therefore T_b(n)$ is in the set $O(n)$ since $T_b(n)$ "$\leq$" $O(n)$

3. Is $T_c(n) = \Theta(T_a(n))$? Answer YES or NO and justify your answer.

   No, $T_c(n) \neq \Theta(T_a(n))$ because $T_c(n)$ is $\Theta(\log n)$ and $T_a(n)$ is $\Theta(n)$ and intuitively, $T_c(n)$ and $T_b(n)$ do not scale at the same rate.

```java
    static int longest_inc_sub(int[] seq) {
        int N = seq.length;
        int d[] = new int[N];
        Arrays.fill(d, 1);
        for (int i = 1; i < N; i++)
            for (int j = 0; j < i; j++) {
                if (seq[j] < seq[i] && d[i] < (d[j] + 1))
                    d[i] = d[j] + 1;
            }

        // Finding the maximum element from d[] as our solution
        int max = 0;
        for(int x = 0; x < d.length; x++) {
            if (max < d[x])
                max = d[x];
        }
        return max;
    }
```

The following are sequences given for our solution to be tested against:

- 10, 9, 2, 5, 3, 101, 7, 18
- 186, 359, 274, 927, 890, 520, 571, 310, 916, 798, 732, 23, 196, 579, 426,188, 524, 991, 91, 150, 117, 565, 993, 615, 48, 811, 594, 303, 191, 505, 724, 818, 536, 416, 179, 485 , 334 , 74, 998, 100, 197, 768, 421, 114, 739, 636, 356, 908 , 477, 656
- 318 , 536 , 390 , 598 , 602 , 408 , 254 , 868 , 379 , 565 , 206 , 619 , 936 , 195 , 123, 314 , 729 , 608 , 148 , 540, 256 , 768 , 404 , 190 , 559 , 1000 , 482 , 141 , 26, 230 , 550 , 881 , 759 , 122 , 878, 350, 756, 82, 562, 897, 508, 853, 317 , 380 , 807 , 23 , 506 , 98 , 757 , 247

For the sake of brevity, each sequence is labeled as as Seq_n in the table below sequentially, which shows the result of our solution for each sequence.

Sequence	Result
Seq_1	4
Seq_2	10
Seq_3	9

Algorithm description

We first create an integer array of equal size as the input and initialize all elements as 1, which we declared as d. The elements are set to 1 because the minimum increasing sequence has to be 1. d stores the solution for the subproblems that we encounter as i is iterated.

The condition checks if the previous element, j is less than the current iterated element, i and whether $d[i]$ is less than $d[j] + 1$ to determine if there are elements greater than any of the previously iterated elements. If the condition passes, we add one to the solution for that index. Since we iterate through the elements $n \cdot n$ times, the solution has a runtime of $O(n^2)$ using the dynamic programming approach.