Homework #1

Name: María Ángel Palacios Sarmiento

Date: 17-09-2025

Class: CS435-003

1. **Question 1:**

   A) Rewrite the polynomial using Horner's Rule.

   $P(x) = 2 + x\ (4 + x\ (\ 3 + x\ (\ 6 + x\ (1 + x\ (5\ )\ )\ )\ )\ )$

   B) Pseudocode naïve polynomial

```
Algorithm: polynomial_evaluation(A, power x )
Input: An array A, a power p
Output: Computation of each term of the array A
total ← 0
for i in range(input) do
     p ← 1
     for j in range (1, i+1) do
     p ← p * x
     total ← total + p * A[i]
return total
```

   C) Running time alg of b: $T(n) = n * O(n)$
   $= O\ (n^2)$

   D) Pseudocode horners Rule:

```
Algorithm: horners_rule(A, x )
Input: An array A, an x value
Output: Computation of each term of the array A
start ← 1
total ← A[0]
for i in range(1, length(A)) do
     total = total * x + A[i]
return total
```

E) Running time alg of c: O(n)

2. Running time: O(n log n)

```
j = i → j = 2i → j = 3i → … → j = ni
j = I → j = ki → j = (k+1)i → … → (k+n)i
j <= n (Replace j with ki)
ki <= n
k <= n/i

Ex:
i = x |  j=i | j = i to n   | n/i | n/i
i = 1 |  j=1 | j = 1,2,3,4  | 4/1 | n/1
i = 2 |  j=2 | j = 2-4      | 4/2 | n/2
i = 3 |  j=3 | j = 3        | 4/3 | n/3
i = 4 |  j=4 | j = 4-4      | 4/4 | n/4

Therefore:

T(n) = n + n/2 + n/3 + … + 1
     = n [ 1 + 1/2 + 1/3 + … + 1/n ]
```

$= n\ [\ \sum_1^n \frac{1}{i}\ ] =$ O(n log n)

3. Evaluate the following:

a) Theoretically Algorithm B grows faster than Algorithm A because it is linear, which means that as n becomes larger, it will grow slower, which leads to define that b has better asymptotic theoretical performance.

b) $n = 5 * 10^6$

$$Ta (n) = Tb (n)$$

$$0.0001\ n^2 /n = 500\ n /n$$

$$0.0001\ n = 500$$

$$n = 500 / 0.0001$$

$$n = 5000000 \rightarrow 5 * 10^6$$

c) Algorithm A, because when we replace n as 10^6 for algorithm A, we end up having a time of 27.8 hours while if we replace with b, we end up having 138.8 hours. This means that definitely the value for A is smaller. However, on the other hands, if we decide to increase the value to 10^7 we can see that the value of A increases while B decreases. In other words, if $n > 10^6$ algorithm B begins to be faster as explained in literal a, while if $n < 10^6$ or small datasets, A is faster, this due to the threshold we got from the previous question.

$$Ta\ (10^6) = 0.0001\ (10^6) = 27.8 \text{ hours}$$

$$Tb\ (10^6) = 500\ (10^6) = 138.8 \text{ hours}$$

4. **Write the code:**

n comparison

```
def find_index(arr, target):
    n = len(arr)
    i =0

    for i in range(n):
        if arr[i] == target:
           return i
        else
             return -1
```

Since the alogithm is divided into blocks, the worst case scenario happens when the target is inside the last block, because the algorithm has to go through all the previous blocks before getting to the last. In other words, it makes $\sqrt{n}$ comparisons when jumping and inside each block to find the target, which in total they sum up at most $3\sqrt{n}$. Therefore the number of comparisons doesn't exceed this bound.

< 3 $\sqrt{n}$ comparison

```
def find_index(arr, target):
    n = len(arr)
    jump  = int(math.ceil(math.sqrt(n)))
    start_element=0

    for start_element in range(0, n, jump):
        if start_element + jump < n:
            last_element = start_element + jump
        else:
            last_element = n


        if arr[last_element - 1] >= target:

            for j in range(start_element,
last_element):
                if arr[j] == target:
                    return j
            return -1
```

5. Review the code:

Worst case scenario is when the target is inside the last and first block. Inside this snippet of the code we encounter ourselves with binary search and exponential search, both having complexity time of O(log n) which means that the algorithm keeps dividing itself until there is only one element of the array, therefore we consider each comparison reduces the space to half, therefore we assume O(log n).