CP312: Algorithm Design & Analysis

Assignment #1

CP312, WLU, 2022

Instructor: Masoomeh Rudafshan

Due: Friday January 14th, 2022

Declan Hollingworth

190765210

holl5210@mylaurier.ca

Nishant Tewari

190684430

tewa4430@mylaurier.ca

# Q1. Prove by induction that for all n>1 the following inequality holds:

Base Case: n = 2

∴ Base case holds true

Asume the inequality holds for n = k

Now we prove the case: n = k + 1

\*we can assume will be in the summation\*

\*add and sub \*

+\*this allows us to isolate S(k)\*

> S(k) >

∴ > and the inequality holds

# Q2. Suppose we have n >= 3 lines so that no two lines are parallel and no three lines intersect at a common point. Prove that at least one of the regions they form is a triangle. (hint: use induction)

Base Case: n = 3

Let L be the # of lines

The # of regions in the plane formed by n lines in general position states that no 2 lines are parallel and no 3 lines intersect at a common point then;

**L = n(n+1)/2 + 1**

L(3) = [ 3\*(3+1) / 2 ] + 1

L(3) = 6 + 1

L(3) = 7

**Note:**

Red Dots -> Regions

Orange Triangle -> Triangle

We can see that a triangle will form at n=3, this triangle will still exist for any n >= 3, therefore for any n >= 3 a triangle will form in at least one of the regions.

# 

# Q3.For each of the following pairs of functions f(n) and g(n), either f(n) = O(g(n)) or g(n) = O(f(n)), but not both. Determine which is the case.

1. f(n) = (n2 - n) / 2 , g(n) = 6n

]

[ -]

= ( - 1) =

∴ Since the limits is , then the following pair of functions is **g(n) = O(f(n))**

1. f(n) = n + 2⎷n, g(n) = n2

+

= 0

= 0

∴ Since both limits are 0, then the following pair of functions is **f(n) = O(g(n))**

1. f(n) = n + log n, g(n) = n ⎷n

+

= 0

= 0

∴ Since both limits are 0, then the following pair of functions is **f(n) = O(g(n))**

1. f(n) = n2 + 3n + 4, g(n) = n3

+ +

= 0

= 0

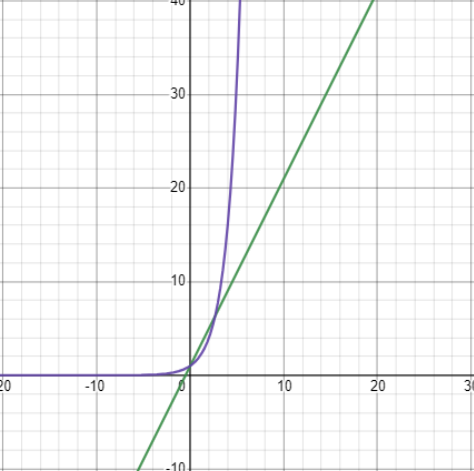
= 0

∴ Since all three limits are 0, then the following pair of functions is **f(n) = O(g(n))**

# 

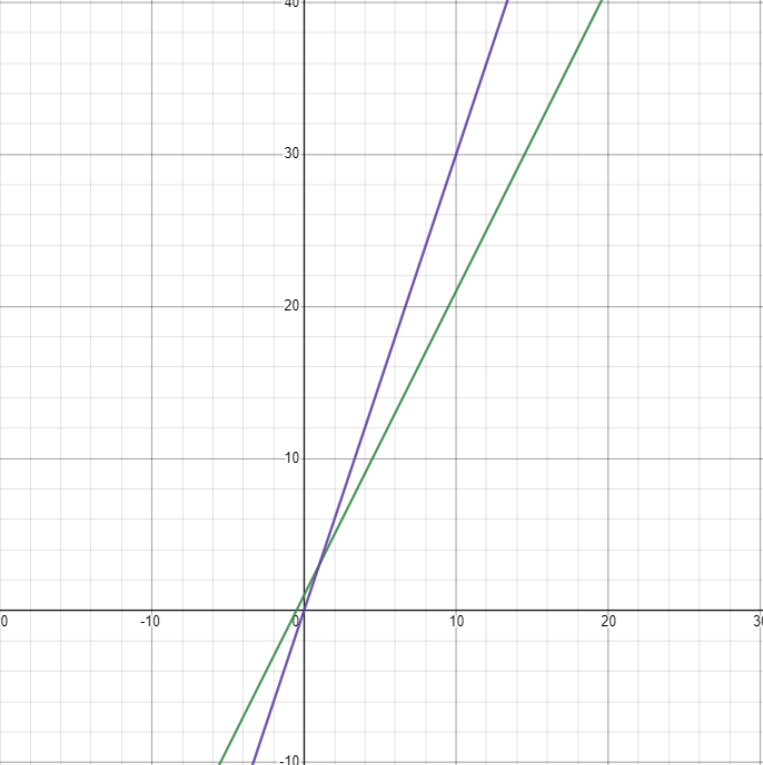
# Q4. Prove the following:

**2n + 1 = O(2n)**

**Is this the best upper-bound for the function on the left? In other words, is it a tight upper-bound? If not, specify the tight upper-bound?**

|  |  |  |
| --- | --- | --- |
| **Size of n** | **2n + 1** | **2n** |
| 1 | 3 | 2 |
| 2 | 5 | 4 |
| 3 | 7 | 8 |
| 4 | 9 | 16 |
| 5 | 11 | 32 |

Looking at the outcomes for the increasing values of n, we can see that 2n + 1 grows linearly, whereas 2n grows exponentially, meaning it is not a valid tight upper-bound. In order to find a tight upper bound, we must find a function f(n) that grows linearly and satisfies the following..

f(n) = 3n would be the obvious choice as it runs linearly and will be greater than or equal to 2n +1 for all instances of n.

|  |  |  |
| --- | --- | --- |
| **Size of n** | **2n + 1** | **3n** |
| 1 | 3 | 3 |
| 2 | 5 | 6 |
| 3 | 7 | 9 |
| 4 | 9 | 12 |
| 5 | 11 | 15 |

∴ O(3n) would be a valid tight upper bound for the function 2n +1

# Q5. For each of the following six code segments:

# 

# Give Big-O analysis of the running time

# Run the code and give the running time for several values of n

# Compare your analysis with the actual runtimes obtained

***Segment 1***

sum = 0

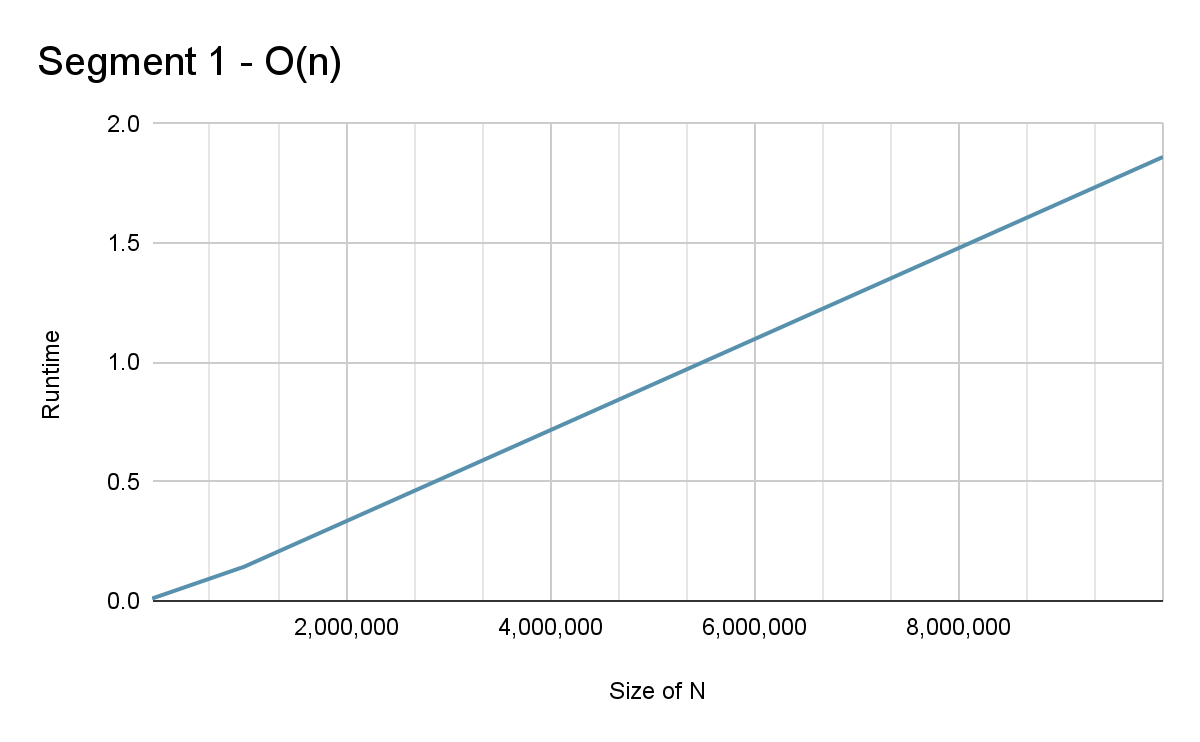
for i in range(n):

sum += 1

**Big-O:** O(n)

**Values:**

|  |  |  |
| --- | --- | --- |
| **Trial #** | **Size of N** | **Runtime (in seconds)** |
| 1 | 100,000 | 0.009996414184570312 |
| 2 | 1,000,000 | 0.14299726486206055 |
| 3 | 10,000,000 | 1.8609983921051025 |



***Segment 2***

sum = 0

for i in range(n):

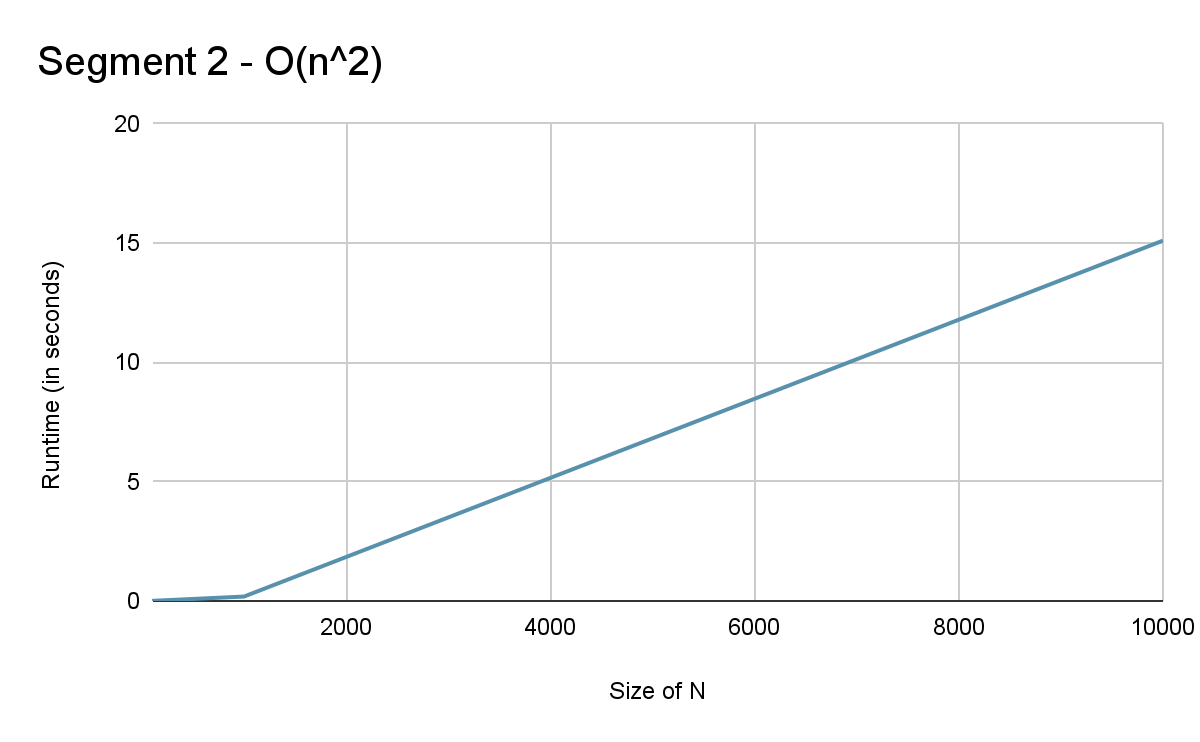
for j in range(n):

sum += 1

**Big-O:** O(n2)

**Values:**

|  |  |  |
| --- | --- | --- |
| **Trial #** | **Size of N** | **Runtime (in seconds)** |
| 1 | 100 | 0.0019991397857666016 |
| 2 | 1,000 | 0.1819934844970703 |
| 3 | 10,000 | 15.102467060089111 |



***Segment 3***

sum = 0

for i in range(n):

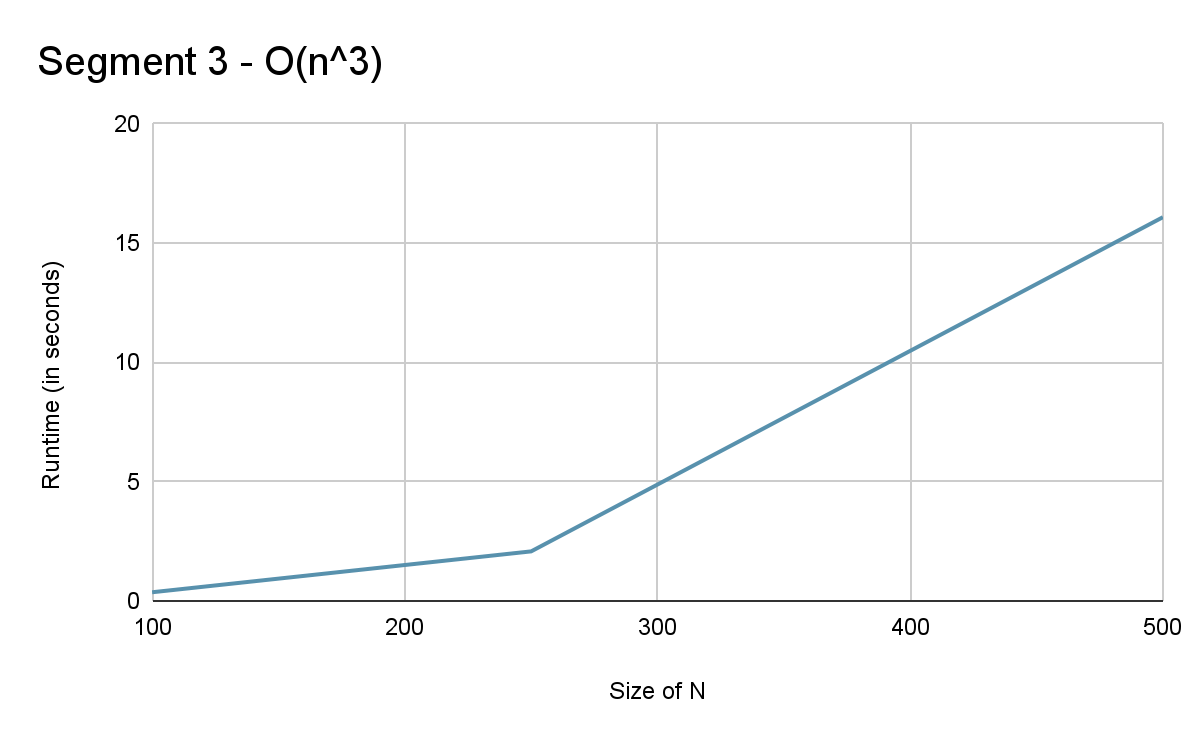
for j in range(n\*n):

sum += 1

**Big-O:** O(n3)

**Values:**

|  |  |  |
| --- | --- | --- |
| **Trial #** | **Size of N** | **Runtime (in seconds)** |
| 1 | 100 | 0.36200380325317383 |
| 2 | 250 | 2.0759949684143066 |
| 3 | 500 | 16.08599352836609 |

******

***Segment 4***

sum = 0

for i in range(n): #O(n)

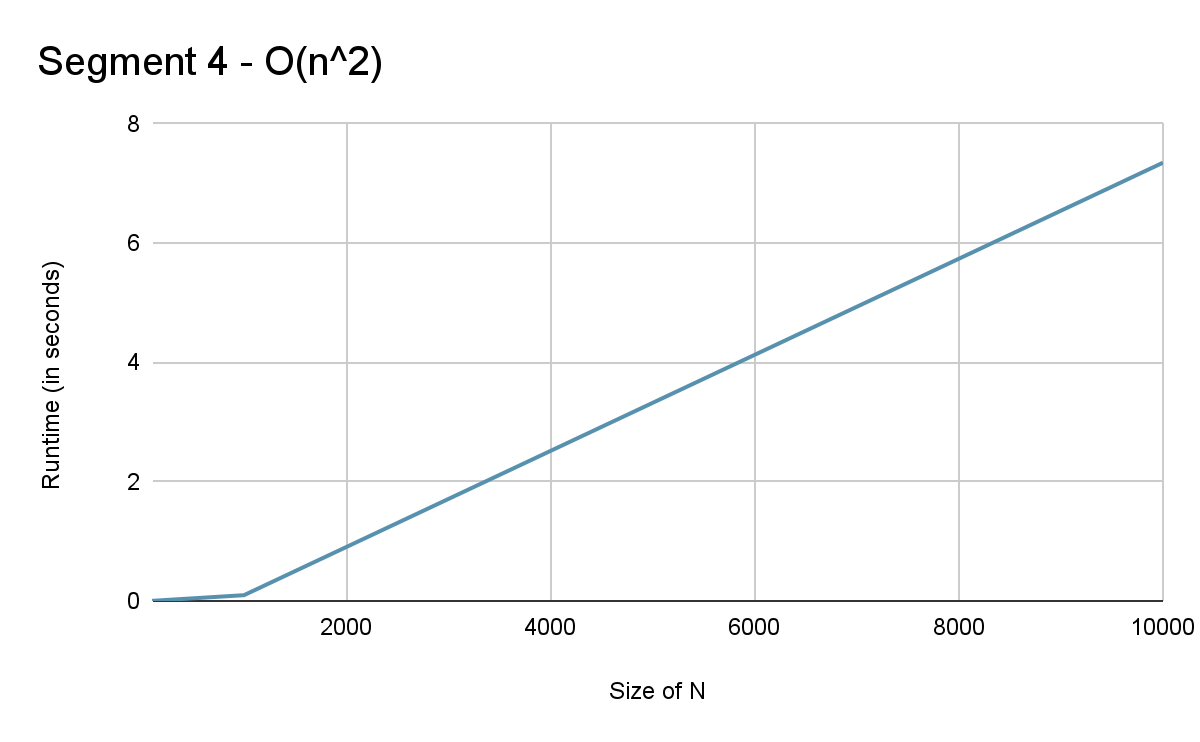
for j in range(i): #(n^2)

sum += 1

**Big-O:** O(n2)

**Values:**

|  |  |  |
| --- | --- | --- |
| **Trial #** | **Size of N** | **Runtime (in seconds)** |
| 1 | 100 | 0.0009996891021728516 |
| 2 | 1,000 | 0.09599757194519043 |
| 3 | 10,000 | 7.347998142242432 |



***Segment 5***

sum = 0

for i in range(n):

for j in range(i\*i):

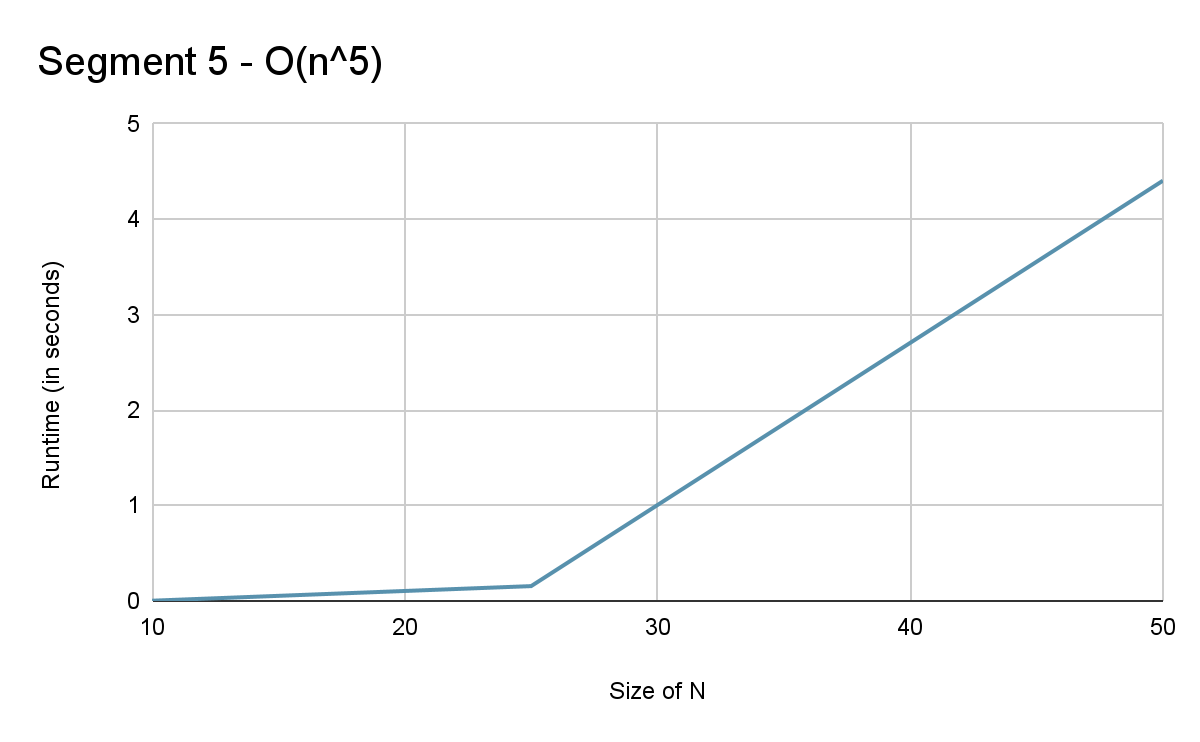
for k in range(j):

sum += 1

**Big-O:** O(n5)

**Values:**

|  |  |  |
| --- | --- | --- |
| **Trial #** | **Size of N** | **Runtime (in seconds)** |
| 1 | 10 | 0.002000093460083008 |
| 2 | 25 | 0.1549985408782959 |
| 3 | 50 | 4.404094696044922 |

******

***Segment 6***

sum = 0

for i in range(n):

for j in range(i\*i):

if (j%i == 0):

for k in range(j):

sum += 1

**Big-O:** O(n5)

**Values:**

|  |  |  |
| --- | --- | --- |
| **Trial #** | **Size of N** | **Runtime (in seconds)** |
| 1 | 50 | 0.1810002326965332 |
| 2 | 100 | 1.6929974555969238 |
| 3 | 150 | 8.793999910354614 |

