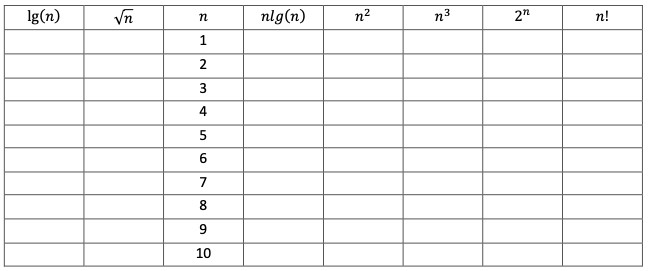
CSC6013 - Worksheet for Week 2

1. Complete this table rounding each decimal to the nearest integer. This should give you a sense of the comparative growth rates of the functions. Note that *lg(n)* means python’s binary log - log2(n).



**1**

**2**

**6**

**24**

**120**

**720**

**5040**

**40,320**

**362,880**

**3,628,800**

**1024**

**512**

**256**

**128**

**64**

**32**

**16**

**8**

**4**

**2**

**1**

**8**

**27**

64

**125**

**216**

**343**

**512**

**729**

**1000**

**1**

**4**

**9**

**16**

**25**

**36**

**49**

**64**

**81**

**100**

**0**

**2**

**5**

**8**

**12**

**16**

**20**

**24**

**29**

**33**

**3**

**3**

**3**

**3**

**2**

**2**

**2**

**2**

**1**

**1**

**3**

**3**

**3**

**3**

**3**

**2**

**2**

**2**

**1**

**0**

1. Solve the summation in two ways: write out all terms of the sum and perform the arithmetic; then use one of the formulas in the class notes to confirm your answer.

  **= 1240**

**12 +22 + 32 + 42 + 52 + 62 + 72 + 82 + 92 + 102 + 112 + 122 + 132 + 142 + 152 =1240**

1. Solve the summation in two ways: write out all terms of the sum and perform the arithmetic; then use one of the formulas in the class notes to confirm your answer.



**20 +21 + 22 + 23 + 24 + 25 + 26 + 27 + 28 + 29 + 210 =2047**

1. Solve the summation in two ways: write out all terms of the sum and perform the arithmetic; then use one of the formulas in the class notes to confirm your answer (round off this answer to three decimal places).

**2-1 + 2-2 + 2-3 + 2-4 + 2-5 + 2-6 + 2-7 + 2-8 =0.996**



1. Solve this summation in two ways: write out all terms of the sum, but there will be no arithmetic to perform; then use one of the formulas in the class notes to express the sum as a polynomial function of n.

  **=**

**1 + 2+ 3 + … + n-1 =**

1. Determine the Big-Oh class of each algorithm. That is, formally compute the worst-case running time as we did in class using a table to produce a function that tracks the work required by all lines of code. Include all steps of the algebraic simplification, but you do not need to provide comments to justify each step. Arithmetic mean = “add them all up, and divide by how many”. Let the size of the problem = n = the number of entries in the array.

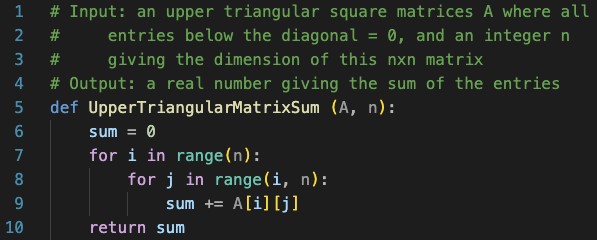
A screenshot of a computer program

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Line | Cost | Count |
| 4 | C1 | 1 |
| 5 | C2 | n |
| 6 | C3 | n |
| 7 | C4 | n |
| 8 | C5 | 1 |

* **T(n)** = **C1** + n**C2** + n**C3** + n**C4** + **C5**
* **T(n)** = **C1** + **C5** + n(**C2** + **C3** + **C4**)
* **T(n)** = **C6** + n**C7**
* **T(n)** <= n**C6** + n**C7**
* **T(n)** <= n(**C6** + n**C7)**
* **T(n)** <= n**C8**

1. Determine the Big-Oh class of each algorithm. That is, formally compute the worst-case running time as we did in class using a table to produce a function that tracks the work required by all lines of code. Include all steps of the algebraic simplification, but you do not need to provide comments to justify each step. Sum of entries in an upper triangular nxn array. Let the size of the problem = n = the dimension of the nxn matrix.



|  |  |  |
| --- | --- | --- |
| Line | Cost | Count |
| 6 | C1 | 1 |
| 7 | C2 | n |
| 8 | C3 | n + n-1 + n-2 + …+ 1 🡪 |
| 9 | C4 | n + n-1 + n-2 + …+ 1 🡪 |
| 10 | C5 | 1 |

* **T(n)** = **C1** + n**C2** + ((n+1)n)/2**C3** + ((n+1)n)/2**C4** + **C5**
* **T(n)** = **C1** + **C5** + n**C2** + n2(**C3** + **C4**)
* **T(n)** = **C6** + n**C2** + n2 **C7**
* **T(n)** <= n2**C6** + n2**C2** + n2 **C7**
* **T(n)** <= n2 (**C6** + **C2** + **C7)**
* **T(n)** <= n2**C8**