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).

$\lg(n)$	$\sqrt{n}$	n	$nlg(n)$	n^2	n^3	2^n	$n!$
0	1	1	0	1	1	2	1
1	1.4	2	2	4	8	4	2
1.585	1.73	3	5	9	27	8	6
2	2	4	8	16	64	16	24
2.322	2.24	5	11.61	25	125	32	120
3	2.449	6	15.51	36	216	64	720
2.81	2.646	7	19.67	49	343	128	5040
3	2.828	8	24	64	512	256	40320
3.17	3	9	28.53	81	729	512	362880
3.322	3.162	10	33.22	100	1000	1024	3628800

2) 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.

$$\sum_{i=1}^{15} i^2$$

3) 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.

$$\sum_{i=0}^{10} 2^i$$

4) 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).

$$\sum_{i=1}^{8} 2^{-i}$$

5) 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.

$$\sum_{i=1}^{n-1} i$$

CSC6013 - Worksheet for Week 2

6) 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.

```
1  # Input: an array A of real numbers
2  # Output: the arithmetic mean of the entries in the array
3  def arithmeticMean(A):
4      sum, count = 0, 0
5      for x in A:
6          sum += x
7          count += 1
8      return sum/count
```

7) 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.

```
1  # Input: an upper triangular square matrices A where all
2  #     entries below the diagonal = 0, and an integer n
3  #     giving the dimension of this nxn matrix
4  # Output: a real number giving the sum of the entries
5  def UpperTriangularMatrixSum (A, n):
6      sum = 0
7      for i in range(n):
8          for j in range(i, n):
9              sum += A[i][j]
10     return sum
```