**Assignment 3**

1. Graph the functions 8n, 4nlogn, 2n2, n3, and 2n using a logarithmic scale for the x- and y-axes; that is, if the function value f(n) is y, plot this as a point with x-coordinate at logn and y-coordinate at logy.
2. The number of operations executed by algorithms A and B is 8nlog n and 2n2, respectively. Determine n0 such that A is better than B for n ≥ n0.
3. The number of operations executed by algorithms A and B is 40n2 and 2n3, respectively. Determine n0 such that A is better than B for n ≥ n0.
4. Order the following functions by asymptotic growth rate.
   1. 4nlog n+2n
   2. 210
   3. 2log n
   4. 3n+100log n
   5. 4n
   6. 2n
   7. n2 +10n
   8. n3
   9. nlogn

N.B: put in the table or list in ascending order.

1. For each function f(n) and time t in the following table, determine the largest size n of a problem P that can be solved in time t if the algorithm for solving P takes f(n) microseconds (one entry is already completed).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Microsecond | Second | Month | year |
| Log*n* | 1x10200 |  |  |  |
| n |  |  |  |  |
| n2 |  |  |  |  |
| 2n |  |  |  |  |

1. Calculate the time complexity of the following algorithms.
   1. **def example1(s ):**

*n = len(s)*

*total = 0*

**for j in range(n):**

*total + = s[j]*

**return** *total*

* 1. **def example2(s ):**

*n = len(s)*

*total = 0*

**for j in range(0,n,2):**

*total + = s[j]*

**return** *total*

* 1. **def example3(s ):**

*n = len(s)*

*total = 0*

**for j in range(n):**

**for k in range(1+j):**

*total + = s[k]*

**return** *total*

* 1. **def example4(s ):**

*n = len(s)*

*prefix = 0*

*total = 0*

**for j in range(n):**

*total + = s[j]*

*total+=prefix*

**return** *total*