Assignment 3

Sarah Sindeband

Z23498939

COP 3014 001

3.8 Order the following functions by asymptotic growth rate.

Ranging from most efficient to least:

2^{10}, 2^{logn}, 4n, 3n + 100logn, 4n, nlogn, 4nlogn + 2n, $n^2 + 10n, n^3, 2^n$

3.19 Show that n is O(nlogn).

f(n) $\leq$ cg(n)	
n $\leq$ cnlogn	
n $\leq$ c nlogn	let c = 3 & n = 2
$\Rightarrow$ 2 $\leq$ 3(2log2)	$2log_2 2 = 2 \: x \: 1$
$\Rightarrow$ 2 $\leq$ 3(2)	

Therefore, n is O(nlogn).

3.20 Show that n^2 is Ω(nlogn).

f(n) $\geq$ cg(n), for n $\geq$ n_0	let c = 1 & n = 2
n^2 $\geq$ c(nlogn)	$2log_2 2 = 2 \: x \: 1$
$\Rightarrow$ 2^2 $\geq$ 1(2 x log2)	$2log_2 2 = 2 \: x \: 1$
$\Rightarrow$ 4 $\geq$ 1(2 x 1)	
$\Rightarrow$ 4 $\geq$ 2	

Therefore, n^2 is Ω(nlogn).

3.24 Give a big-Oh characterization, in terms of n, of the running time of the example2 function shown in Code Fragment 3.10.

```
9  def example2(S):
10   """Return the sum of the elements with even index in sequence S."""
11   n = len(S)
12   total = 0
13   for j in range(0, n, 2):          # note the increment of 2
14     total += S[j]
15   return total
```

line 12	n
line 13	n
line 14	1
line 15	1

n + n + 1 + 1 is O(n) characterization.

3.25 Give a big-Oh characterization, in terms of n, of the running time of the example3 function shown in Code Fragment 10.

```
17 def example3(S):
18   """Return the sum of the prefix sums of sequence S."""
19   n = len(S)
20   total = 0
21   for j in range(n):              # loop from 0 to n-1
22     for k in range(1+j):          # loop from 0 to j
23       total += S[k]
24   return total
```

line 20	n
line 21	n
line 22	n^2
line 23	1
line 24	1

n + n + n^2 + 1 + 1 is O(n^2) characterization.

3.27 Give a big-Oh characterization, in terms of n, of the running time of the example5 function shown in Code Fragment 10.

```
def example5(A, B):               # assume that A and B have equal length
  """Return the number of elements in B equal to the sum of prefix sums in A."""
  n = len(A)
  count = 0
  for i in range(n):              # loop from 0 to n-1
    total = 0
    for j in range(n):            # loop from 0 to n-1
      for k in range(1+j):        # loop from 0 to j
        total += A[k]
    if B[i] == total:
      count += 1
  return count
```

The three nested loops would be O(n^3) characterization.

3.33

Al: O(nlogn) faster for n $\geq$ 100

Bob: O(n^2) faster for n < 100

This is possible because Al's nlogn function will begin at a higher y intercept when the y axis is featuring the constant (c) value. As n, the number of inputs, increases (on the x axis) the Bob's exponential function will grow at a much larger rate than Al's. Therefore, until the threshold of n = 100 is passed, Al's algorithm will be slower.