CST 370
Spring 2018
Midterm (Written)

Name: _Aaron Gordon______________________

Email: _aagordon@csumb.edu_______________

- Do not start until told to do so.
- Use your time wisely—make sure to answer the questions you know first.
- **Total points = 50**
- **Read the questions carefully.**

1. (**8 points**) What is output of the following code? A brief explanation of how you arrived at the answer is necessary.

```
int values = [17, 3, 5, 8, 9, 11, 13, 15, 1];

Stack s;
for (int i = 0; i < 9; i++)
     s.push( values[ i ] );

int n = 25;
for (int i = 0; i < 4; i++)
     {
          n += s.pop( );
     }

for (int i = 0; i < 2; i++)
     {
          n -= s.pop( );
     }
cout << n;
```

All nine elements of the integer array **values** are pushed onto the stack **s** via the first *for* loop:

1
15
13
11
9
8
5
3
17

The second *for* loop adds the top four elements of **s** to the integer **n**, which is initialized to the value 25:

n = 25 + 1 + 15 + 13 + 11 = 65

9
8
5
3
17

The final *for* loop subtracts the next two elements from the top of **s** from **n**:

n = 65 – 9 – 8 = 48

The last line of the program prints **n** to the screen. This means the final output of the program is:

48

2. (**8 points**) Write the output value (displayed by "cout" steatement) of the following code and outline how you get the output value

```
myQueue.enqueue(200);

myQueue.enqueue(100);

myQueue.enqueue(500);

cout << myQueue.front() << endl;

// the function front() prints the element at the front of the
queue.

myQueue.dequeue();

cout << myQueue.front() << endl;
```

The values 200, 100 and 500 are enqueued into **myQueue** in that order:

500	100	200

The first instance of *cout* prints the front of **myQueue**, which is 200.

Next, the front of **myQueue** is dequeued:

	500	100

The second instance of *cout* prints the new front of **myQueue**, which is 100.

This means the final output of the program is:

200
100

3. (**8 points**) What is the complexity of the following code (in big O notation), assuming that the cout statement has O(1) complexity and n > 2)? Explain your answer.

```
for (int i = 1; i < n ; i++)

        for (int j = i; j < n ; j++)

                cout << i + j;
```

The outer *for* loop is executed **n-1** times, while the inner *for* loop is executed **n-i-1** times, where **i** is, as a result of the outer loop, the integer set [1, n-1]. Thus, the *cout* statement is executed:

$(n-2) + (n-3) + \ldots + 2 + 1 + 0 - 1$

This is the sum from -1 to n-2, which reduces to the sum of 1 to n-2 minus 1:

$[[(n - 2)(n - 2 + 1)] / 2] - 1 = [[(n - 2)(n - 1)] / 2] - 1 = [(n^2 - 3n + 2) / 2] - 1$

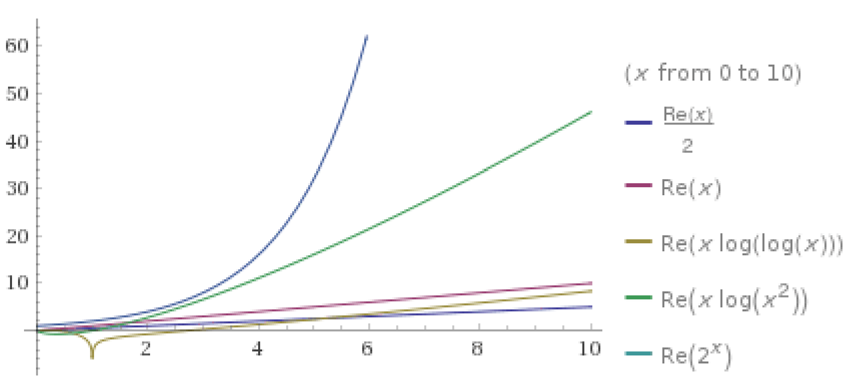
In Big O notation we drop constant factors and consider only the largest term. Thus the complexity of the above code is:

$O(n^2)$

4. (**8 points**) Rank the following in order of increasing complexity O(N), O(N loglogN), O(2/N), O(N log(N^2)), O(2^N).

O(2/n), O(N), O(N loglog N), O(N log(N^2)), O(2^N)

Plot:



5. (**10 points**) Consider two stacks each of size 6. When you pop an element from the first stack you multiply it by 3 and add it to the second stack (if the second stack is not full). When you pop an element from the second stack you multiply it by 4 and add it to the first stack (if the second stack is not full).

   Push numbers 1 to 5 to the first stack in order (i.e., push 1 first, then 2 and so on). Push numbers 6 to 10 to the second stack in order. First pop two numbers from the first stack (remember when you pop a number it is going to be added to the second stack). Then pop three numbers from the second stack (remember when you pop a number it is going to be added to the first stack).

   a) What is the value in the top of the first stack?

   b) What is the value at the top of the second stack?

   c) What is the current size of the first stack?

   d) What is the current size of the second stack?

Push 1 to 5 and 6 to 10:

5	10
4	9
3	8
2	7
1	6

Pop two numbers from the first stack:

	15
	10
	9
3	8
2	7
1	6

Pop three numbers from the second stack:

36	
40	
60	
3	8
2	7
1	6

a) 36
b) 8
c) 6
d) 3

6. (**8 points**) Solve the following recurrence relation. Explain how you derive the solution.

$x(n) = 4x(n-1)$ for $n > 1$, $x(1) = 1$

$x(n-1) = 4x(n-2) \rightarrow x(n) = 4(4x(n-2)) = 16x(n-2)$

$x(n-2) = 4x(n-3) \rightarrow x(n) = 16(4x(n-3)) = 64x(n-3)$

$x(n-3) = 4x(n-4) \rightarrow x(n) = 64(4x(n-4)) = 256x(n4)$

…

The above pattern suggests that..

$$x(n) = 4^k * x(n-k)$$

Let k = n-1, then..

$$x(n) = 4^{n-1} * x(n – (n-1)) = 4^{n-1} * x(1) = 4^{n-1} * 1$$

Thus:

$$\mathbf{x(n) = 4^{n-1}}$$