Homework 1

(75 points)

Name: Aima Salman

1. *Order the following functions by growth rate* ***(12 points)*** *Indicate which of the functions grow at the same rate.*
   LOWEST GROWTH
   2/N
   $37 = 2^{50}$
   Sqrt(N)
   N
   NloglogN
   $N\log N = N\log(N^2)$
   $N\log^2 N$
   $N^{1.5}$
   N^2
   $N^2\log N$
   N^3
   2^N
   HIGHEST GROWTH

2. *Give the Big-O notation for the following expressions:* ***(10pts, 2pt each)***

   a. $2n^4 + 3n^3 - 5 = O(n^4)$

   b. $4^n - n^2 + 19 = O(4^n)$

   c. $\frac{5}{3}n = O(n)$

   d. $5n * \log(n) + 8 = O(n\log(n))$

   e. $[n(n+1)/2 + 2n] / 3 = O(n^2)$

3. *For each of the following code fragments give running time analysis (Big Oh). Explain your answer* ***(25pt, 5pts each)***

```
a. sum = 0;
   for ( i=0; i < n ; i++)
     sum++;
```

The runtime of this code fragment would be O(n) because, in the worst-case scenario, the loop can run n times. This is a linearly increasing growth rate. Because there is only one loop, we use that runtime to represent the entire code segment.

```
b. sum = 0;
   for( i = 0; i < n; i++)
      for(j = 0; j < i ; j++)
            sum++;
```

The runtime of this code fragment would be $O(n^2)$ because, in the worst-case scenario, the loop can run n^2 times. This is a quadratically increasing growth rate. Both the outer and inner loops can run n times each, therefore, each having run times of O(n). Because the loops are within each other, we combine the run times by multiplying them to represent the entire code segment.

```
c. sum = 0;
   for( i = 0; i < n; i++)
      for( j = 0; j < i*i ;  j++)
            for( k = 0; k<j; k++)
                  sum++;
```

The runtime of this code fragment would be $O(n^5)$ because, in the worst-case scenario, the loop can run n^5 times. This is a polynomial-ly increasing growth rate. The outermost loop runs n times, the middle one runs n^2 times because it squares the number from the previous loop, and the innermost loop runs n^2 times as well because it runs the same amount as the previous loop (j times). From the outermost to the innermost loop, the runtimes are O(n), $O(n^2)$, and $O(n^2)$. Because the loops are within each other, we combine the run times by multiplying them to represent the entire code segment.

```
d. if(value < n)
      for( i = 0; i < n; i++)
      {
            System.out.println(i);
      }
   else
      System.out.println(value);
```

The runtime of this code fragment would be O(n) because, in the worst-case scenario, the loop can run n times. This is a linearly increasing growth rate. Because there is only one loop, we use that runtime to represent the entire code segment. The if-else statements do not affect the runtime as they are O(1).

```
e. sum2 = 0;
   sum5 = 0;

   for(i=1; i<=n/2; i++)
```

```
{
   sum2 = sum + 2;
}
for(j=1; j<=n*n; j++)
{
   sum5 = sum + 5;
}
```

The runtime of this code fragment would be $O(n^2)$ because, in the worst-case scenario, the loop can run n^2 times. This is a quadratically increasing growth rate. The first loop can run n/2 times and the second one n^2 times, therefore making the runtimes O(n) and $O(n^2)$ respectively. Because the loops are not within each other, we pick the loop with the worst run time to represent the entire code segment.

4. *What is the time complexity of the below function?* ***(10 points)***

```
void fun(int n, int arr[])
{
    int i = 0, j = 0;
    for(; i < n; ++i)
        while(j < n && arr[i] < arr[j])
            j++;
}
```

The runtime of this code would be O(n) because, in the worst-case scenario, the loop can run n times. This is a linearly increasing growth rate. Although the loops are within each other, j does not reset with each iteration of i. Therefore, it just continues to increase. So, the most that the loops can run is 2n.

5. *What is the Big-O running time for this code? Explain your answer.* ***(10 points)***

```
int i = numItems;
while (i > 0)
{
     i = i / 2;  // integer division will eventually reach
zero

}
```

The runtime of this code would be O(log(n)) because, in the worst-case scenario, the loop can run log(n) times. This is a logarithmically increasing growth rate. Each iteration of this loop looks like $n/2^k$, so it will be a logarithmic function when solved.

6. *An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 if the running time is the following (assume lower order terms are negligible)* ***(8 Points)***
    a. *Linear*

- $\frac{T(500)}{T(100)} = \frac{500}{100}$
- $\frac{T(500)}{0.5} = \frac{500}{100}$
- $T(500) = 2.50ms$

b. O(N log N)

- $\frac{T(500)}{T(100)} = \frac{500\log(500)}{100\log(100)}$
- $\frac{T(500)}{0.5} = \frac{5\log(500)}{\log(100)}$
- $T(500) = 3.37ms$

c. Quadratic

- $\frac{T(500)}{T(100)} = \frac{500^2}{100^2}$
- $\frac{T(500)}{0.5} = \frac{250000}{10000}$
- $T(500) = 12.5ms$

d. Cubic

- $\frac{T(500)}{T(100)} = \frac{500^3}{100^3}$
- $\frac{T(500)}{0.5} = \frac{125000000}{1000000}$
- $T(500) = 62.5ms$