Assignment 1

Key Points	Big O Notation
Source	Assignment

Question 1

Question 1: (1 marks)

For algorithm A, If the exact number of steps is $T(n)=2n+3n^2-1$ what is the Big O? Explain.

this would be `O(n^2)`, since the n^2 term will dominate

Question 2

Question 2: (2 marks)

Consider the below functions we discussed in our lecture:

Linear, logarithmic, exponential, quadratic, constant, cubic,

Write the above function from top to bottom order from most to least efficient.

- Constant `O(1)` - most efficient
- Logarithmic `O(log(n))`
- Linear `O(n)`
- quadratic `O(n^2)`
- cubic `O(n^3)`
- exponential `O(2^n)` - least efficient

Question 3

Question 3: (3 marks)

Consider the below code fragment:

```
int test = 0;
   for (int i = 0; i < n; i++){
      for (int j = 0; j < n; j++){
         test = test + i * j;
      }
   }
```

What is its Big-O running time? Explain your answer.

In this code fragment, we see a nested for loop within a for loop, meaning the big O of this code will be `O(n^2)`

Question 4

Question 4: (3 marks)

Consider the below code fragment:

```
int func(){
   int test = 0;
   for (int i = 0; i < n; i++){
      test = test + 1;
   }
   for (int j = 0; j < n; j++){
      test = test - 1;
   }
   return 0;
}
```

What is its Big-O running time? Explain your answer.

In this case, we see two individual for loops. Therefore the big O of this code fragment is `O(n)` with linear time.

Question 5

Question 5: (4 marks)

Consider the below code fragment:

```
int func(){
   int i = n;
   int count = 0;
   while (i > 0){
      count = count + 1;
      i = i // 2;
   }
   return 0;
}
```

What is its Big-O running time? Explain your answer.

Since the number of steps halves each time, we would say this has a big O of `O(log(n))`

Question 6

Question 6: Write a scenario (or a code fragment), whose complexity is $O(n^3)$ **(3 marks)**

```
public class q6 {
    public static void main(String[] args) {
        nCubed(5);

    }

    public static void nCubed(int n){
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    System.out.println("" + i + j + k);
                }
            }
        }
    }

}
```

Question 7

Question 7: If an algorithm performing at $O(n^2)$ has the integer 7 as input, what is the worst case scenario for the algorithm? **(1 marks)**

In a worst case scenario, we could undergo 49, since 7^2 is 49.

Question 8

Question 8: Use Big O Notation to describe the time complexity of the following function that determines whether a given year is a leap year: **(1 marks)**

```
bool isLeapYear(year) {
        return (year % 100 === 0) ? (year % 400 === 0) : (year % 4 === 0);
      }
```

```
if year/100 has remainder 0:
    return if year/400 has a remainder of 0
else:
  return if year/4 has a remainder of 0
```

This has the same number of iterations no matter what the input is, therefor it is constant time, and has a big O of `O(1)` with constant time.

Question 9

Question 9: Use Big O Notation to describe the time complexity of this function, which is below: **(3 marks)**

```
int chessboardSpace(numberOfGrains)
{  chessboardSpaces = 1;
    placedGrains = 1;
    while (placedGrains < numberOfGrains) {
        placedGrains *= 2;
        chessboardSpaces += 1;
}
return chessboardSpaces; }
```

Explain your answer.

In this case, the number os steps is halved on each iteration of the loop, so the big O is `O(log(n))`

Question 10

Question 10: Consider the code below: **(4 marks)**

```
i = 1;
sum = 0;
while (i <= n) {
    i = i + 1;
    sum = sum + i;
}
```

In our lecture, we have done an example about calculating the primitive operations and then determines the complexity. First identify the primitive operation of every line, and then calculate the Big-O of the above code? Also mention the class of growth rate function.

Rewriting the code, and then commenting the primitive operations of each step

```
i = 1;  // 1 op
sum = 0;  // 1 op

while (i<=n){  // 1 op x (n+1)
  i = i +1;  // 2 ops x n
  sum = sum + i;  // 2 ops x n
}
```

For this one, it should be noted that the while loop first line is executed one more time than it's inner contents always (because of the last time when i≤n evaluates to false)

The big O of this function is: 1 + 1 + (n+1) + 2*n + 2*n = `O(5n + 3)`

When we get rid of all the constants, this is `O(n)`

That means the class of the growth rate function is linear.

Question 11

Question 11: In our lecture, we have discussed the Big Omega represents the lower bond. What is the lower bound of the below function:

$3nlogn - 2n$

- we are talking about big Omega, which represents the lower bound
- the lower bound is `n log(n)` in regards to big Omega.