1. [5] Order the following set of functions by their growth rate:

Unordered Complexities	Ordered Complexities
N	2/N
√N	37
N^1.5	$\sqrt{N}$
N^2	N
N log N	N log(log(N))
N log(log(N))	N log N
N log^2 N	$N \log^2 N$
2/N	$N^{1.5}$
2^N	N^2
2^(N/2)	$N^2 \log(N)$
37	N^4
N^2 log(N)	$2^{N/2}$
N^4	2^N

2. [5] A program takes 35 seconds for input size 20 (i.e., n=20). Ignoring the effect of constants, approximately how much time can the same program be expected to take if the input size is increased to 100 given the following run-time complexities? Chapter 2.1 notes that: T(N) ≤ cf(N). For this you'll need to find the c (constant scaling factor) for a given Big-O growth rate.

1. O(N) $O(N) = \frac{35}{20} \times 100 = 175$ seconds
2. O(N + log N) O(N) is the dominating part, $O(N + \log N) = O(N) = 175$ s
3. O(N^3) $O(N^3) = \frac{35}{20^3} \cdot 100^3 = 4375$ s
4. O(2^N)[1] $O(2^n) = \frac{35}{2^{20}} \cdot 2^{100} = 4.231240369 \times 10^{25}$ s

[1] You might need an online calculator with arbitrarily large numbers for this one. Scientific notation and 8 significant figures is just fine.

3. [8] Given the following two functions:

```
int g(int n)
{
   if(n <= 0)
   {
      return 0;
   }
   return 1 + g(n - 1);
}
```

```
int f(int n)
{
   int sum = 0;
   for(int i = 0; i < n; i++)
   {
      sum += 1;
   }
   return sum;
}
```

A. [2] State the runtime complexity of both f() and g()

g( ): O(N), f( ): O(N)

B. [2] State the memory (space) complexity for both f() and g()

g( ): O(N), recursive N times.

f( ): O(1), the sum result space.

C. [4] Write another function called "int h(int n)" that does the same thing, but is significantly faster.

```
int h (int n) {
      return n;
      }
```

4. [5] State g(n)'s runtime complexity:

```
int f(int n){
   if(n <= 1){
      return 1;
   }
   return 1 + f(n/2);
}

int g(int n){
   for(int i = 1; i < n; i *= 2){
      f(i);
   }
}
```

$\Rightarrow$ f( ) calls itself recursively with $\frac{n}{2}$ times,
time complexity: O(log N)

$\Rightarrow$ g( ) has a for loop and in each iteration f( ) is getting called.
time complexity: $O(\log N \cdot \log N)$
$= O(\log^2 N)$

5. [5] What is the runtime complexity of Adam's famous string splitter code? Hint: Make sure to look into the source code for string.find() in the C++ std library. I've included that code (downloaded from GNU).

```
static vector<String> split(String text, String delimiter)
{
        vector<String> pieces;
        int location = text.find(delimiter);
        int start = 0;

      //while we find something interesting
        while ( location != String.Length() ){

            //build substring
                string piece = text.substring(start, location - start);
                pieces.push_back(piece);
                start = location + 1;

            //find again
                location = text.indexOf(delimiter, start);
        }
        string piece = text.substr(start, location - start);
        pieces.push_back(piece);
        return pieces;
}
```

```
// Excerpt from OpenJDK's implementation of string searching in
//  src/java.base/share/classes/java/util/regex/Pattern.java
//  The key component is how long it takes to run to find the next match
boolean match(Matcher matcher, int i, CharSequence seq) {
            if (i > matcher.to - minLength) {
                matcher.hitEnd = true;
                return false;
            }
            int guard = matcher.to - minLength;
            for (; i <= guard; i++) {
                if (next.match(matcher, i, seq)) {
                    matcher.first = i;
                    matcher.groups[0] = matcher.first;
                    matcher.groups[1] = matcher.last;
                    return true;
                }
            }
            matcher.hitEnd = true;
            return false;
```

The while loop runs $O(N)$. text.substring(...) runs $O(N - \text{constant})$.

The overall runtime complexity is $O(N^2)$.

6.

```
public static int SmallestValueK(int n) {

        int [] decimal = new int [10];

        for (int i =0; i< decimal.length; i++) {
           decimal[i] = 0;
        }


        int k = 0;
        while(sum(decimal) < 10)

        {
        k++;
        int product = n*k;
        while(product != 0) {
           decimal[product % 10] = 1;
                    product = product / 10;         }

   }
   return k; }

   public static int sum (int [] array){
            int sum = 0;
            for(int i = 0; i<array.length; i++) {
                     sum = sum + array[i];
            }
            return sum;
   }
```

Time Complexity: O(log N)

7.

A. Determine whether a provided number is odd or even
Time complexity: O(1)
Space complexity: O(1)

example:

```
public static String OddOrEven(int n) {
if (n%2 == 0) {
         return "even";
} else {
         return "odd";
}
}
```

B. Determine whether or not a number exists in a list

Time complexity: O(n)
Space complexity: O(1)

example:

```
    public static boolean NumberExists(List <Integer> l, int n) {
    for (int i = 0; i<l.size(); i++) {
        if(l.get(i) == n)
            return true;
    }
    return false;
}
```

C. Find the smallest number in a list

Time complexity: O(n)
Space complexity: O(1)

example:

```
    public static int SmallestNumber(List <Integer> l1) {
    int smallest = l1.get(0);
    for (int x : l1) {
        if (x<smallest) {
            smallest = x;
        }
    }
    return smallest;
}
```

D. Determine whether or not two unsorted lists contain all of the same values(assume no duplicate values)

Time Complexity: O(n^2)
Space Complexity: O(1)

piece of pseudocode:

```
for i in range(0,n);
for j in range(0,n);
if a[I] != b[j] ——— return false
——— return true
```

E. Determine whether or not two sited lists contain all of the same values (assume no duplicate values)

Time Complexity: O(n)
Space Complexity: O(1)

piece of pseudocode:

```
for int in in range(0,n)
if a[I] != b[I] ——— return false;
——— return true;
```

F. Determine whether a number is in a BST

Time Complexity: O(log N)
Space Complexity: O(1)

piece of pseudocode:

```
Compare value n with the middle element
If x matches with the middle element, return the mid index
Else if x > mid element, then x lie in right half subarray of the mid
element, recur for right half.
Else if x < mid element, recur for the left half.
```

8. What is Git and what is it for?

Git is the most commonly version control system. It tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions. Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.

9. Get a copy of a Git repository off of the GitLab server?

git Clone + HTTPS or SSH

10. Once I've created/edited/removed a file in my Git repository, what common do I use to stage it for committing?

git add

11. Once I've staged all of my changes, which command do I use to create the next version of the repository?

git commit -m " "

12. Now that I've created at least one new update to my repository, which command do I use to send those changes to the GitLab server?

git push

13. If the server has had updates from another computer, which command do I use to get these changes on my local computer without starting from a whole new copy of the Git repository?

git pull

14. How does this variable get set and what is it get set with?

```
    public static int main(String args[]) {
        return(0);
}
```

args[] is an array in type String and command line argument. The command line argument can be passed in main function by the command java filename “command line argument” in terminal. Java VM will first execute the main function as well as bring in the command line argument.