Fill In The Blank

1. break
2. while loop, for loop
3. 2^n
4. T(n) = 4T(n/2) + n

Answer The Questions

1.

```
if(food == 0) {
                return 1;
        }

if(food < 0) {
                return 0;
        }
```

2. A recursive function will call itself, while a loop will just repeat until a condition is met. By using a recursive function programmers have the option to recycle their code, therefore saving memory and run time.

3. If a base case is not found then the function will repeat forever until memory storage is eventually lost.

4. Linear search will start from the start and search one by one till the end of the list, while a binary search will split the list in half then search, repeating this until the target number is found. For a large data set the binary approach is superior, but for a smaller data set the linear approach is best.

Compute The Running Time

1.

```
public static int foo7( int n )
{
  // n is guaranteed to be >= 0
  if ( n == 0 )   = O(n)
      return 0;   = O(1)
  else
```

```
    return ( n + foo7( n - 1 ) );  = O(n)
}
```

T = O(n) + O(n) + O(1)
2 O(n) + O(1)
Find fastest growth and drop any coefficient
T = O(n)
Growth rate is Linear

2.

```
public static int foo8( int n )
{
  // n is guaranteed to be >= 1
  if ( n == 1 || n == 2 )  = O(n)
     return 1;  = O(1)
  else
     return ( foo8( n - 1 ) + foo8( n - 2 ) );  = O(n) * O(n)
}
```

Hint: Note that T(n –2) <= T(n – 1)

T = O(n) + O(1) + O(n^2)
O(n^2) + O(n) + O(1)
Find fastest growth and drop any coefficient
T = O(2^n)
Growth rate is Exponential

3.

```
public static void foo11( int n )
{
  // n is guaranteed to be >= 0
  if ( n == 0 )  = O(n)
     return 0;   = O(1)
  else
     return ( 5 + 2 * foo11( n - 1 ) );     = O(n) * O(1)
}
```

T = O(n) + O(n) + O(1)
2 O(n) + O(1)
Find fastest growth and drop any coefficient
T = O(n)
Growth rate is Linear

4.

```
public static int eating(int food) {

		if(food == 0) {    = O(n)
			return 1;   = O(1)
		}

		if(food < 0) {      = O(n)
			return 0;      = O(1)
		}

		return eating(food-1) + eating(food-2) + eating(food-3);   = O(2^n)

	}
```

T = O(1) + O(1) + O(2^n)
O(2^n) + 2 O(1)
Find fastest growth and drop any coefficient
T = O(2^n)
Growth rate is Exponential

Understand And Fill The Code

1.

```
 public class One {

	public static void main(String[] args) {

		Scanner input = new Scanner(System.in);
		System.out.println("What is your String? Please input all letters in either
UPPERCASE or lowercase! Thanks!");
		String s = input.next();

		char[] c = s.toCharArray();

		System.out.println(" repeats " + foo(s,c) + " times!");

		input.close();

	}

	public static int foo(String s, char[] c ){

		if ( s.length() == 0 ) {
```

```
                return 0;

            }

            else{
                int count = 1;
                for (int i = 0; i < s.length(); i++) {
              for (int j = i + 1; j < s.length(); j++) {
                if (c[i] == c[j]) {
                  System.out.print(c[j] + " ,");
                  count++;
                }
              }
                }
                return count;
            }
        }

}

2.
public class Two{

        public static void main(String[] args) {

            Scanner input = new Scanner(System.in);
            System.out.println("Enter the number you would like to square: ");
            int n = input.nextInt();

            System.out.println("The result of your number (" + n + "), squared is " + foo(n));

            input.close();
        }

        public static int foo( int n ){

         while(n < 1000) {

            if ( n >= 1000 )  {

             return n;

            }
```

```
                if(n < 1000) {
                        n = n * n;
                 }
        }

        return n;
        }


}
```

3.

public class Two **(Accidentally named it Two instead of Three)** {

```
        public static void main(String[] args) {

                Scanner input = new Scanner(System.in);
                System.out.println("Enter the number you would like to square: ");
                int n = input.nextInt();

                System.out.println("The result of your number (" + n + "), squared is " + foo(n));

                input.close();
        }

        public static int foo( int n ){

         while(n < 1000) {

                 if ( n >= 1000 )  {

                  return n;

                 }

                 if(n < 1000) {
                         n = n * n;
                  }
        }

        return n;
        }
```

```
}
```