CPW 245 Homework # 1 – Big-O Notation

Here is a list of classes of functions that are commonly encountered when analyzing algorithms. The slower growing functions are listed first. c is some arbitrary constant.

Notation	Name
$O(1)$	constant
$O(\log(n))$	logarithmic
$O(n)$	linear
$O(n \log(n))$	linearithmic
$O(n^2)$	quadratic
$O(n^3)$	qubic
$O(n^c)$	polynomial
$O(c^n)$	exponential

1. (13 points) What is the Big-O time complexity of this algorithm as a function of **n**, the size of the array? Explain your answer.

```
public static boolean contains(int[] array, int value) {
    int n = array.length;
    for (int i = 0; i < n; i++) {
        if (array[i] == value) {
            return true;
        }
    }
    return false;
}
```

2. (13 points) Assuming the above definition of **contains**, what is the Big-O time complexity of this algorithm as a function of **n**, the size of the array? Explain your answer.

```
public static void printContains (int [] array, int value) {
    int n = array.length;
    if (contains(array, value)) {
        for (int i = 0; i < 10; i++) {
            System.out.println (array[0]);
        }
    }
}
```

3. (13 points) Assuming the above definition of **contains**, what is the Big-O time complexity of this algorithm as a function of **n**, the size of the array? Explain your answer.

```
public static void containsPrint (int [] array, int value) {
    int n = array.length;
    for (int i = 0; i < n - 1; i += ( n – i ) / 2 ) {
        if (contains (array, value)) {
            System.out.println (array[i]);
        }
    }
}
```

4. (13 points) What is the Big-O time complexity of this algorithm as a function of **n**, the size of the array? Explain your answer.

```
public void printAllOrderedPairs(int[] arrayOfItems) {
  for (int firstItem : arrayOfItems) {
    for (int secondItem : arrayOfItems) {
      int[] orderedPair = new int[]{firstItem, secondItem};
      System.out.println(Arrays.toString(orderedPair));
    }
  }
}
```

5. (13 points) What is the Big-O time complexity of this algorithm as a function of the parameter **n**? Explain your answer.

```
public static void countDown(int n) {
    if (n < 0) {
        n *= -1;
    }
    while (n > 0) {
        n--;
    }
}
```

6. (13 points) What is the Big-O time complexity of this algorithm as a function of the parameter **n**? Explain your answer.

```
public static void countDown2(int n) {
    if (n < 0) {
        n *= -1;
    }
    while (n > 0)
        n /= 2;
    }
}
```

7. (13 points) What is the Big-O time complexity of this algorithm as a function of the parameter **n**? Explain your answer.

```
public static void countDown3(int n) {
    int x = n;
    while (x > 0) {
        int y = n;
        while (y > 0) {
            int z = n;
            while (z > 0) {
                z--;
            }
            y--;
        }
        x--;
    }
}
```

8. (13 points) What is the Big-O time complexity of this algorithm as a function of the parameter **n**? Explain your answer.

```
public static void addData(int n) {
    ArrayList<Integer> list = new ArrayList<>();
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            list.add(0, i);
        }
        list.clear();
    }
    while (!list.isEmpty()) {
        list.remove(0);
    }
}
```