Goals for this assignment
• Practice using Big-Oh/Theta/Omega notation
• Analyze the running times of some algorithms

Submission Checklist
You should submit a PDF file titled hw4.pdf. Upload it on ICON under Assignments > Homework 4. Physical paper copies are not accepted. If you handwrite, then your work must be legible.

Part 0: Read about Asymptotic Analysis (http://homepage.cs.uiowa.edu/~bdmyers/cs2230_fa17/) and take Quiz 4

Part 1: Experiments
1. Ryan and Brandon are arguing about the solution to your upcoming homework assignment on sorting algorithms. Ryan claims that his \( O(n \log n) \)-time solution must always be faster than Brandon’s \( O(n^2) \) solution. However, Brandon claims that he ran several experiments on both algorithms on his laptop and sometimes his was faster. Explain what might have happened.

Part 2: Growth rate
2. Order the following functions by asymptotic growth rate:
   a. \( 5n \log n + 4n \) | \( 200n^3 \) | \( 10n + 2n \) | \( 4\log n \)
   b. \( 6n \) | \( 7n \log n \) | \( 8n + 9 \) | \( 60000 \times n^6 \)
   c. \( 2^{100} \) | \( 2n^2 + 200n^2 \log n \) | \( n^3 - 2000 \) | \( n^{90} \) | \( 3^{n-1} \)
   d. \( 63 \) | \( 4n \) | \( 3\log n \) | \( 2^{n+2} \) | \( 10^{\log n} \)

Part 3: Proof and Analysis
3. Give a good big-Oh characterization in terms of \( n \) of the running time of the following. Provide brief justification for your answer by finding a suitable \( k \) and \( n_0 \).
   a. \( n^4 + 3n \)
   b. \( 15n^{16} + 3n \log n + 2n \)
   c. \( 3n \log n + 2\log n \)
   d. \( 12n \times 3^n - 50n \)

4. Give a good big-Theta characterization in terms of \( n \) of the running time of the following. Provide brief justification for your answer (in terms of finding a \( k_0, k_1 \), and \( n_0 \)).
   a. \( 5 \log n + 12n^2 \)
   b. \( 6n \log n + 4n \)

5. Show that the following statements are true:
   a. \( 2^{n+5} \in O(2^n) \)
   b. \( n^2 \in \Omega(n \log n) \)
Part 4: Algorithm Analysis

6. Given the following algorithms below, give a big-Oh characterization of the running time in terms of the input size (or magnitude). For each problem, you must provide justification (description, equations, and/or diagrams) for your answer to earn any points.

a. Find the running time of two_sum in terms of N, the length of arr.

```java
public static boolean two_sum(int[] arr) {
    for (int i=0; i<arr.length; i++) {
        for (int j=i; j<arr.length; j++) {
            if (i==j && arr[i]+arr[j]==0) {
                return true;
            }
        }
    }
    return false;
}
```

b. Find the running time of something(n), in terms of n.

```java
public static int something(int n){
    for (int i=1; i<n; i++) {
        if(i%2 == 0)
            break;
    }
    return 1;
}
```

c. First, find the big-Oh running time of inside, in terms of Na and Nb, the lengths of the arrays a and b.

```java
private static double[] inside(double[] a, double[] b) {
    double[] c = new double[a.length];
    int i = 0, j = 0;
    for (int k = 0; k < c.length; k++) {
        if (i < a.length) {
            if (j < b.length) {
                if(a[i] <= b[j]) {
                    c[k] = a[i];
                } else {
                    c[k] = b[j];
                }
            } else {
                c[k] = a[i];
                i++;
            }
        } else {
            if (j < b.length) {
                c[k] = b[j];
                j++;
            }
        }
    }
    return c;
}
```
Now, find the running time of `outside`, in terms of N, the length of list. Use your answer from above for calculating the running time a call to `inside`.

```java
public static double[] outside(double[] list) {
    int x = list.length;
    if (x <= 1) return list;
    double[] a = new double[x/2];
    double[] b = new double[x - x/2];
    for (int i = 0; i < a.length; i++) {
        a[i] = list[i];
    }
    for (int i = 0; i < b.length; i++) {
        b[i] = list[i + x/2];
    }
    return outside(inside(a, b));
}
```

d. Find the running time of `printItA(n)`, in terms of n.

```java
public static void printItA(int n){
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j*=2)
        {
            System.out.println("Something");
        }
    }
}
```

e. Find the running time of `printItB(n)`, in terms of n.

```java
public static void printItB(int n){
    for(int i = 0; i < n; i++)
    {
        for(int j = n; j > 0; j/=2)
        {
            System.out.println("Something");
        }
    }
```
f. Find the running time of `strange_sumA` in terms of N, the length of `arr`.

```java
int strange_sumA(int[] arr) {
    if (arr.length == 1) {
        return arr[0];
    } else {
        int newlen = arr.length/2;
        int[] arrLeft = new int[newlen];
        int[] arrRight = new int[newlen];
        for (int i=0; i<newlen; i++) {
            arrLeft[i] = arr[i];
        }
        for (int i=newlen; i<arr.length-1; i++) {
            arrRight[i-newlen] = arr[i];
        }
        return strange_sumA(arrLeft) + strange_sumA(arrRight);
    }
}
```

g. Find the running time of `strange_sumB` in terms of N, the length of `arr`. (If it makes the analysis easier, you may assume that the length of `arr` is a power of 2).

```java
int strange_sumB(int[] arr, int left, int right) {
    if (right - left == 1) {
        return arr[left];
    } else {
        return strange_sumB(arr, left, right/2) + strange_sumB(arr, right/2, right);
    }
}
```

h. Briefly, why do `strangeSumA` and `strangeSumB` have different asymptotic running times?