Sheet 2

1. **Use the definition of Big O to prove that *f(n) = O(g(n))*:** (assume *log* to be a binary logarithm (base 2)).

a. $f(n) = 3n^2 + 2n + 1$ and $g(n) = n^2$

b. $f(n) = 5n + 10$ and $g(n) = n$

c. $f(n) = 2n^3 + 4n^2 + 5$ and $g(n) = n^3$

d. $f(n) = \log(n) + 3$ and $g(n) = \log(n)$

e. $f(n) = n + 5$ and $g(n) = n^2$

---

2. **Use the definition of Big *Ω* to prove that *f(n) = Ω(g(n))*:** (assume *log* to be a binary logarithm (base 2)).

a. $f(n) = 3n^2 + 2n + 1$ and $g(n) = n^2$

b. $f(n) = 5n + 10$ and $g(n) = n$

c. $f(n) = 2n^3 + 4n^2 + 5$ and $g(n) = n^3$

d. $f(n) = \log(n) + 3$ and $g(n) = \log(n)$

e. $f(n) = n^2 + 3n\log(n)$ and $g(n) = n^2$

---

3. **Use the definition of Big Θ to prove that *f(n) = O(g(n))*:** (assume *log* to be a binary logarithm (base 2)).

a. $f(n) = 3n^2 + 2n + 1$ and $g(n) = n^2$

b. $f(n) = 2n^3 + 4n^2 + 5$ and $g(n) = n^3$

c. $f(n) = \log(n) + 3$ and $g(n) = \log(n)$

4. **Analyze the following pieces of code and calculate the complexity of the algorithms used in each of them.**

a.

```c
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
}
```

b.

```c
void printPairs(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("(%d, %d) ", arr[i], arr[j]);
        }
    }
}
```

c.

```c
int binarySearch(int arr[], int left, int right, int x) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] < x)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}
```

d.

```c
int getFirstElement(int arr[], int n) {
    return arr[0];
}
```

e.

```c
void printTriplets(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                printf("(%d, %d, %d) ", arr[i], arr[j], arr[k]);
            }
        }
    }
}
```

f.

```c
int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```