*This homework covers material from Chapter 2 in the textbook. It is due in class on Wednesday, January 31. You can work with other students on this assignments, but you should write up your own answers in your own words. Remember to **justify your answers**. The last two problems should be easy, but they are meant to show interesting examples.*

1. Compute the exact value of each of the following summations, by applying known summation formulas—and some regular math:

a) $\sum_{k=1}^{20} (k+2)$ **b)** $\sum_{i=3}^{10} \left(\frac{1}{2}\right)^i$ **c)** $\sum_{n=0}^{8} \frac{2^n + 3^n}{5^n}$

2. For each of the following functions $f(n)$, find a function $g(n)$ such that $f(n) = \Theta(g(n))$. Choose $g(n)$ to be as simple as possible. (Justify your answer using properties of Θ, along with your knowledge of dominance relations among functions.)

a) $n\log(n) + n + n^2$ **b)** $(5n^3 + 3n)(3\log(n) + \sqrt{n})$ **c)** $2^n + 3^{n+1} + 4^{n+2}$

d) $(n^2 + n^3)(n^4 + n^5)$ **e)** $(1 + n + \sqrt{n})(n + \log(n))$

3. Suppose that A and B are integer arrays of size n. Give a run time analysis for each of the following Java code segments. That is, find a function $f(n)$ such that the run time is in $\Theta(f(n))$. Briefly justify your answers.

a)
```
for (int i = 0; i < n; i++) {
    B[i] = 0;
    for (int j = 0; j < i; j++)
        B[i] = B[i] + A[j]
}
```

b)
```
int[] C = new int[n];
for (int i = 0; i < n; i++) {
    C[i] = A[i]*B[i];
}
```

c)
```
int k = 0;
for (int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        if (A[i] == B[j])
            k = k + 1;
    }
}
```

4. Devise an algorithm that finds the second-largest element in an array A of length N. You can assume that $N \geq 2$ and that all the elements in the array are different. (Write the algorithm either in Java or in careful pseudocode.) What is the run-time analysis for your algorithm? Exactly how many comparisons of array elements does it do?

5. B is an infinite array of numbers, **sorted into increasing order**. You know that the array contains the number X, but you don't know where in the array it is located. Devise an algorithm that will find the index, N, of X in the array (that is, find N such that $A[N] = X$). **Your algorithm must have run time $\Theta(log(N))$, where N is the index of X in the array.**

6. The run-time analysis of an algorithm in terms of input size can depend on how you measure the size of the input. The usual analysis of the algorithm for multiplying two $n \times n$ matrices gives the run time as $\Theta(n^3)$. But the input to the algorithm actually consists of $2 \cdot n^2$ numbers. What is the run-time analysis for multiplication of $n \times n$ matrices, if you measure the size of the input as the actual number of numbers that are input to the calculation.

7. If $f(n)$ is a function that gives the run time for an algorithm on inputs of size n, it's natural to assume that $f(n)$ is increasing, that is, bigger inputs take more time to process. But not all functions are increasing. Suppose we define $f(n) = 1$ when n is even and $f(n) = n^2$ when n is odd. If you look at the sequence of values of f, they would be: $1^2, 1, 3^2, 1, 5^2, 1, 7^2, 1, 9^2, 1, \ldots$.

a) Explain why $f(n) = \mathcal{O}(n^2)$, using the definition of Big O.

b) Explain why $f(n) \neq \Omega(n^2)$, using the definition. (That is, it is not possible to find a positive constant c such that $f(n) \geq cn^2$ for all large enough values of n, no matter how small you make c.)

c) Write a function, `int g(int n)`, whose run time for $n > 0$ is $\Theta(f(n))$, where $f(n)$ is the function given above. Your function doesn't need to do anything interesting.

(Note that since $f(n) = \mathcal{O}(n^2)$ but $f(n) \neq \Omega(n^2)$, it is also true that $f(n) \neq \Theta(n^2)$. According to the definition on page 39 of the textbook, this means that $f(n) << n^2$. I don't think this definition make sense in this case. It does, however, make sense for increasing functions.)