1. For each of the following pairs of functions \((f(n), g(n))\) determine if \(f(n) = O(g(n))\), \(f(n) = o(g(n))\), \(f(n) = \Theta(g(n))\), \(f(n) = \Omega(g(n))\), or \(f(n) = \omega(g(n))\); assume \(1 < a < b\), and that \(a, b \in \mathbb{R}\) are constants. You should determine the strongest relationship between each pair of functions. For example, if you are given the pair \((n, n^2)\), then you should say \(n = o(n^2)\) and not \(n = O(n^2)\). You must justify your answers.

\((2^n, 2^{n^2/2}), (e^n, n2^n), (n^{\sqrt{n}}, 2^{an}), (a^n, 2^{bn}), \text{ and } (a^{n^2}, b^n)\).

2. Assume that you have functions \(f\) and \(g\) such that \(f(n) = O(g(n))\). For each of the following statements, decide whether you think it is true or false and give a brief supporting argument or a counterexample.

i. \(\log f(n) = O(\log g(n))\).

ii. \(2^f(n) = O(2^g(n))\).

iii. \(f(n)^2 = O(g(n)^2)\).
3. Prove or disprove the following. In case of a proof, use the definitions of \(O, \Omega, \Theta\) and give values of the constants in the definitions for which the conditions in the definition hold.

(i) \(5n\sqrt{n} = O(\frac{1}{2}n^2 - 10)\)
(ii) \(2^{5\lg n + \lg\lg n} \lg(n^5) = O(4^{\lg n})\)

4. For each of the code fragments given below, give a bound of the form \(\Theta(f(n))\) on its running time on an input of size \(n\). Justify your answer.

(a) 

\[
\begin{align*}
\text{sum} & \leftarrow 0 \\
\text{for} \ (i = 1; \ i < n; \ i = i \times 2) \\
& \quad \text{for} \ (j = i; \ j < n; \ j = j + 1) \\
& \quad \text{sum} \leftarrow \text{sum} + 1 \\
\end{align*}
\]

(b) 

\[
\begin{align*}
\text{for} \ (i = 1; \ i \leq n + 100; \ i = i + 1) \\
& \quad \text{for} \ (j = 1; \ j \leq i \times n; \ j = j + 1) \\
& \quad \quad \text{sum} \leftarrow \text{sum} + j \\
& \quad \text{for} \ (k = 1; \ k \leq 3n; \ k = k + 1) \\
& \quad \quad c[k] \leftarrow c[k] + \text{sum}
\end{align*}
\]

(c) 

\[
\begin{align*}
\text{sum} & \leftarrow 0 \\
\text{for} \ (i = 1; \ i \leq n; \ i = i + 1) \\
& \quad \text{for} \ (j = 1; \ j \leq i; \ j = j \times 2) \\
& \quad \quad \text{for} \ (k = 1; \ k \leq j; \ k = k \times 3) \\
& \quad \quad \quad \text{sum} \leftarrow \text{sum} + 1 \\
\end{align*}
\]

5. Solve the following recurrences using the method of expansion (iteration), giving your answer in \(\Theta\) notation. For all recurrences, assume that \(T(n) = 1\) for all \(n \leq 2\) and \(n\) is an exact power of 2.

a. \(T(n) = T(n/2) + n\)

b. \(T(n) = T(n - 1) + \lg n\)

6. Solve the following recurrences using the method of expansion (iteration), giving your answer in \(\Theta\) notation. For all recurrences, assume that \(T(n) = 1\) for \(n \leq 10\) and \(n\) is an exact power of 2.

a. \(T(n) = T(n - 2) + n^3\)

b. \(T(n) = 2T(n/2) + \lg n\)
7. Daniel is shirking his TA duties by swiping on Tinder, where there is a list of $n$ candidates he may match with. He only has one SuperLike, so he is trying to find one candidate to SuperLike. When considering a person, Daniel can give them a score with the highest score being the best and no ties being possible. He considers the candidates one by one. Because of how Tinder works, after he considers the $k$th candidate, he can either SuperLike the candidate before considering the next candidate or he forever loses the chance to SuperLike that candidate. We suppose that the candidates are considered in a random order, chosen uniformly at random from all $n!$ possible orderings.

We consider the following strategy. First, consider $m$ candidates but reject them all: these candidates give Daniel a sample of the scores. After the $m$th candidate, SuperLike the first candidate he considers who receives a higher score than all of the previous candidates.

(a) Let $E$ be the event that Daniel SuperLikes his most preferred candidate, and let $E_i$ be the event that the $i$th candidate is the most preferred and he SuperLikes them. Determine $\Pr[E_i]$, and show that

$$\Pr[E] = \frac{m}{n} \sum_{j=m+1}^{n} \frac{1}{j-1}$$

(b) Bound $\sum_{j=m+1}^{n} \frac{1}{j-1}$ to obtain

$$\frac{m}{n} (\ln n - \ln m) \leq \Pr[E] \leq \frac{m}{n} (\ln(n-1) - \ln(m-1)).$$

You may find the analysis of the bound on the $n$th Harmonic number on pages 1153-1156 of the text useful.

(c) Show that $m(\ln n - \ln m)/n$ is maximized when $m = n/e$, and explain why this means $\Pr[E] \geq 1/e$ for this choice of $m$.

8. Jared is a Meme Lord and spends about 5 hours a day browsing, sharing, and tagging his friends in memes. Given a set of memes, a meme $m$ is “similar” to a meme $m'$ if and only if there exists a caption $c$ that is appropriate for memes $m$ and $m'$. For example, “Salt Bae” and “Vogue Boy” would be considered similar because the caption “When you use WLOG in your CIS 160 proof” is appropriate for both. However, the similarity relationship for memes is not transitive.

Jared is a popular figure on two separate meme pages. Because double posting would be the end of his meme career, Jared must split his memes into two groups. He wants to split the memes in such a way that he maximizes the number of pairs of similar memes that are separated.

Jared doesn’t believe there is a very efficient algorithm for grouping these memes, so he asked his good friend, real estate billionaire Sadat, for some help. Sadat came up with the following algorithm:

1. Randomly and independently color each meme red or blue with probability 1/2 each.
2. Output the grouping defined by the red/blue split of the memes.
Let the random variable $X$ denote the number of similarity relationships between the
groups output by the algorithm. In other words, $X$ refers to the number of similarity
relationships between red-colored memes and blue colored memes.

a. Compute $E[X]$ as a function of the number of pairs of similar memes, and deduce
that $E[X] \geq \frac{\text{OPT}}{2}$, where OPT is the maximum number of separated pairs possible.

b. Let $p$ be the probability that number of separated pairs output by the algorithm has
size at least $\frac{49}{100} \text{OPT}$. Using Markov’s inequality show that $p \geq \frac{1}{51}$.
(Markov’s inequality states that for any non-negative random variable $X$ and any for
any $a > 0$, $\Pr[X \geq a] \leq \frac{E[X]}{a}$)

c. Compute the variance $\text{Var}[X]$.

d. Let $p$ be the probability defined in part (b). Use Chebyshev’s inequality together with
part (c) to show that $p = 1 - O(|E|^{-1})$, where $E$ is the set consisting of all pairs of
similar memes.
(Chebyshev’s inequality states that for a random variable $X$ and for any $a > 0$,
$\Pr[|X - E[X]| \geq a] \leq \frac{\text{Var}[X]}{a^2}$

Note how Chebyshev’s inequality gives a much sharper bound here than does Markov’s
inequality.)

e. How would you modify the algorithm so that it always finds the number of separated
pairs to be at least $\frac{49}{100} \text{OPT}$ but has only expected linear running time?

9. Consider the following basic problem. You’re given an array $A$ consisting of $n$ integers
$A[1], A[2], \ldots, A[n]$. You’d like to output a two-dimensional $n$-by-$n$ array $B$ in which $B[i, j]$
(for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ – that is, the sum $A[i] +
A[i+1] + \cdots + A[j]$. (The value of the array entry $B[i, j]$ is left unspecified whenever $i \geq j$,
so it doesn’t matter what is the output of these values.)

Here is a simple algorithm to solve this problem.

```
for (i=1; i <= n; i++)
    for (j=i+1; j <= n; j++)
        Add up array entries $A[i]$ through $A[j]$
        Store the result in $B[i,j]$
```

a. For the above code fragment give a bound of the form $O(f(n))$ on its running time on
an input of size $n$. Justify your answer.

b. For this same function $f$, show that the running time of the algorithm on an input of size
$n$ is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running
time.)
c. Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem – after all, it just iterates through the relevant entries of the array $B$, filling in a value for each – it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n \to \infty} g(n)/f(n) = 0$.

10. Consider an algorithm $sort(A)$ that takes as input an array $A$ of integers. The algorithm works by calling $aux(0, n, A)$ where $n$ is the length of $A$ and where $aux(lo, hi, A)$ is a recursive algorithm. The arguments $lo$ and $hi$ of $aux$ delimit the portion of the array $A$ that $aux$ sorts, namely $A[lo], A[lo + 1], \ldots, A[hi - 1]$.

$aux(lo, hi, A)$ works as follows:
1. If $hi - lo$ is 0 or 1, return. Otherwise go to the next step.
2. If $hi - lo$ is 2, put $A[lo]$ and $A[hi - 1]$ in order (swap if needed) then return. Otherwise go to the next step.
3. Divide the array portion between $lo$ and $hi$ into three (approximately) equal parts. Call $insertion-sort$ to order the middle third, then recursively call $aux$ for the lower third, and finally recursively call $aux$ for the upper third.
4. Merge all three sorted parts.

a. Let $T(n)$ be the worst-case running time for $sort$ on an array of length $n$ (assume that $n$ is an exact power of 3). Write a recurrence relation for $T(n)$. On the right side of the recurrence relation do not include the terms of the form $cn^p$ where $p \geq 0$, except for the term of highest degree. Explain your answer briefly.

b. Analyze the running time of the algorithm. Prove your answer.