1. (2 points each) Determine whether the following statements are True or False.

(a) False There exists a sufficiently large \( k \) such that \( x \) is \( O((\log x)^k) \).

(b) True \( \log(x^5) \) is \( O(\log(x)) \).

(c) True If \( f_1 \) is \( O(g_1) \) and \( f_2 \) is \( O(g_2) \), then \( f_1f_2 \) is \( O(g_1g_2) \).

(d) False If \( f_1 \) is \( O(g_1) \) and \( f_2 \) is \( O(g_2) \), then \( f_1 + f_2 \) is \( O(g_1 + g_2) \).

(e) True Let \( f(n) \) be the \( n \)-th Fibonacci number. Then \( f(n) \) is \( O(3^n) \).

2. (2 points each) Fill in the blanks.

(a) The binary search algorithm has complexity \( \Theta(\log n) \).

(b) The merge sort algorithm has complexity \( \Theta(n \log n) \).

(c) The number of multiplications used in the usual matrix multiplication of two \( n \times n \) matrices is \( \Theta(n^3) \).

(d) Let \( f(n) = 2^n n! \). A recursive definition of \( f(n) \) is

\[
\begin{align*}
  f(1) & = 2 \\
  f(n + 1) & = 2(n + 1)f(n)
\end{align*}
\]

for \( n \geq 1 \).

3. (4 points) Find two functions \( f(x) \) and \( g(x) \) such that \( f(x) \) is \( O(g(x)) \), but \( 2^{f(x)} \) is not \( O(2^{g(x)}) \).

(Answers may vary.)

Let \( f(x) = \log(x^2) \) and \( g(x) = \log(x) \). Then \( 2^{f(x)} = x^2 \) and \( 2^{g(x)} = x \).

4. (4 points) Find two functions \( f \) and \( g \) such that \( f \) is not \( O(g) \) and \( g \) is not \( O(f) \).

(Answers may vary.)

Let \( f(x) = \sin(x) \) and \( g(x) = \cos(x) \).
5. (12 points) Prove using the definition of big-$O$ notation that $3^n$ is not $O(2^n)$. (Do not use calculus.)

We will prove by contradiction. Suppose that $3^n$ is $O(2^n)$. Then there are constants $c$ and $k$ such that $3^n \leq c \cdot 2^n$ for all $n > k$. It follows that $(3/2)^n \leq c$, so $n \leq \log_{1.5}(c)$. This contradicts the statement that the inequality $3^n \leq c \cdot 2^n$ is true for all $n > k$. We then conclude that $3^n$ cannot be $O(2^n)$.

-OR-

We will prove by contradiction. Suppose that $3^n$ is $O(2^n)$. Then there are constants $c$ and $k$ such that $3^n \leq c \cdot 2^n$ for all $n > k$. However, if $n = \max(\log_{1.5}(c), k) + 1$, then $n > k$ and $n > \log_{1.5}(c)$, so $(3/2)^n > c$, contradicting the statement that $3^n \leq c \cdot 2^n$ for all $n > k$. We then conclude that $3^n$ cannot be $O(2^n)$.

6. (6 points each) Arrange the functions in each list in so that each function is big-$O$ of the next function. (All logarithms are base 2.)

(a) $n \log n$, $1.1^n$, $0.9^n$, $n^4$, $n^3 + n^2 + n$, $\log(n!)$

$0.9^n$, $\log(n!)$, $n \log n$, $n^3 + n^2 + n$, $n^4$, $1.1^n$

(b) $(\log n)^n$, $10^n$, $\log(n^7)$, $(\log(n))^7$, $2^n \log n$, $n$

$\log(n^7)$, $(\log(n))^7$, $n$, $10^n$, $(\log n)^n$, $2^n \log n$

(Note: $\log n > 10$ for large $n$, and $2^n \log n = n^n$.)
7. (13 points) Prove using the definition of big-$O$ notation that $\sqrt{2x^2 + 15}$ is $O(x)$.

(Answers may vary.)

Let $c = 2$ and $k = 4$. Then for any $x > k$, we have $15 \leq x^2$, so $2x^2 + 15 \leq 3x^2 \leq 4x^2$. Taking squareroot of both sides gives us $|\sqrt{2x + 15}| \leq c|x|$ for all $x > k$. Hence $\sqrt{2x + 15}$ is $O(x)$, by definition.

8. (3 points each) Give big-$O$ estimates for the number of times “max” is computed in the following segments of algorithms. Only the tightest estimate will get credit. Choose your answer from the following list:

\[ n, n^{1.5}, n^2, n^{2.5}, n^3, \log n, n \log n, n^2 \log n, 2^n \]

(a)

\begin{verbatim}
 x := 1
 for i := 1 to n
   for j := 1 to i
     x := max(x, i * j)
 return x
\end{verbatim}

(b)

\begin{verbatim}
 x := 1
 for i := 1 to n
   x := max(x, i)
 for j := 1 to n^2
   x := max(x, j)
 return x
\end{verbatim}

(c)

\begin{verbatim}
 x := 1
 for i := 1 to n
   j := 1
   while j < n
     x := max(x, i * j)
     j := 2 * j
 return x
\end{verbatim}

(d)

\begin{verbatim}
 x := 1
 for i := 1 to n
   j := 1
   while j < 1000
     x := max(x, i * j)
     j := j + 1
 return x
\end{verbatim}

Solution: (a) $O(n^2)$, (b) $O(n^2)$, (c) $O(n \log n)$, (d) $O(n)$
9. Let \( f(n) \) be a function on non-negative integers defined recursively as

\[
f(0) = 0, f(1) = 1, \text{ and } f(n) = 3f(n-1) - 2f(n-2) \text{ for } n \geq 2.
\]

(a) (3 points) Find a non-recursive formula for \( f(n) \).

\[
f(n) = 2^n - 1
\]

(b) (11 points) Prove using strong induction that your answer from part (a) is correct.

Basis Step: When \( n = 0 \), \( 2^n - 1 = 0 = f(n) \).

Inductive Step: Let \( k \geq 0 \), and suppose \( f(\ell) = 2^\ell - 1 \) for \( 0 \leq \ell \leq k \). In particular, we have \( f(k) = 2^k - 1 \) and \( f(k-1) = 2^{k-1} - 1 \). Then

\[
\begin{align*}
f(k + 1) &= 3f(k) - 2f(k - 1) \\
&= 3(2^k - 1) - 2(2^{k-1} - 1) \\
&= 3 \cdot 2^k - 3 - 2^k + 2 \\
&= 2 \cdot 2^k - 1 \\
&= 2^{k+1} - 1.
\end{align*}
\]

We conclude by strong induction that \( f(n) = 2^n - 1 \) for all \( n \geq 0 \).

10. Following is a recursive algorithm for computing \( (n!)^2 \) for any positive integer \( n \geq 1 \).

(a) (5 points) Fill in the blanks.

\[
\text{procedure} \ factorialSquare(n: \text{ positive integer})
\text{ if } n = 1 \text{ then return } 1
\text{ else}
\text{ return } n \ast n \ast \text{factorialSquare}(n-1)
\]

(b) (3 points) Let \( g(n) \) be the number of multiplications used for input \( n \). Give a recursive definition of \( g(n) \).

\[
g(1) = 0, \text{ and } g(n) = g(n-1) + 2.
\]

(c) (3 points) Give an exact formula for \( g(n) \).

\[
g(n) = 2n - 2 \text{ for all } n \geq 0
\]