1. Comparing functions
   (a) (10 points) Order the following functions according to their order of growth from the lowest to the highest. If you think that two functions are of the same order (i.e. $f(n) \in \Theta(g(n))$), put them in the same group.
   
   $5n + \log n, n^2, n + n^2 \log n, 2^n, n^2 \log n, (n + 1)!, 2^n, n^3, 2n + 5$. 
   
   $5n + \log n << n + n^2 \log n \sim n^2 \log n << n^{\frac{5}{2}} << n^3 + 2^n + 5 << 2^n << \log n$ 
   
   (b) (10 points) For each pair of functions in the table below, determine whether $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$, $f(n) \in \Theta(g(n))$, or all of them. It is NOT necessary to justify your answer.

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
<th>$g(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(n + 5) \log n$</td>
<td></td>
<td></td>
<td></td>
<td>$n^2 + \log n$</td>
</tr>
<tr>
<td>$n^2 \log n$</td>
<td></td>
<td></td>
<td></td>
<td>$5n^2 + n \log n$</td>
</tr>
<tr>
<td>$n \log n$</td>
<td></td>
<td></td>
<td></td>
<td>$n \sqrt{n}$</td>
</tr>
<tr>
<td>$n^2 + 10 \log n$</td>
<td></td>
<td></td>
<td></td>
<td>$n^2 - \sqrt{n}$</td>
</tr>
<tr>
<td>$\log_3 (n^3)$</td>
<td></td>
<td></td>
<td></td>
<td>$2 \log_2 n$</td>
</tr>
</tbody>
</table>

2. (10 points) Using the definition of $\Theta$, prove that $\frac{n^2}{2} + 5n + 1 \in \Theta(n^2)$.

   Let $c_1 \leq \frac{1}{2}, c_2 \geq 7$, and $n > n_0$. Note that $\forall n > n_0$

   $$c_1 n^2 \leq \frac{n^2}{2} \leq n^2 + 5n + 1 \leq n^2 + 5n^2 + n^2 \leq 7n^2 = c_2 n^2.$$ 

   Therefore $\frac{n^2}{2} + 5n + 1 \in \Theta(n^2)$. 

   2
3. (10 points) Find the order of growth of the following sums. (i.e., is it in \( \Theta(n^2) \), \( \Theta(n \log n) \), or \( \ldots \)?)

a. \[
\sum_{i=0}^{n} (3^{i-1} + 2^i) = \sum_{i=0}^{n} 3^{i-1} + \sum_{i=0}^{n} 2^i = \frac{1}{3} \sum_{i=0}^{n} 3^i + \sum_{i=0}^{n} 2^i
\]

= \[
\frac{3^n - 1}{3 - 1} + 2^{n+1} - 2 \in \Theta(2^n)
\]

b. \[
\sum_{i=1}^{n} (\log(i) + 2i)
\]

\[
\sum_{i=1}^{n} \log(i) + 2 \sum_{i=1}^{n} i \approx n \log(n) + n(n+1) \in \Theta(n^2)
\]
4a. (10 points) Analysis of iterative algorithm. You are given the following code snippet:

```java
int sum = 0;
for (int i = 1; i <= n; i = i + 1) {
    sum = sum + i;
}
print (sum);
```

a. How many times will line 3 be executed, as a function of n?

\[ n \text{ times} \]

b. What is the output of the program if \( n = 32 \)?

\[
1 + 2 + 3 + \cdots + 32 = \frac{32(33)}{2} = 16(33) = 528
\]

c. In general, what is the output of the program for any given positive integer \( n \)?

\[
\frac{n(n+1)}{2}
\]
4b. (Extra credit. 10 points) Analysis of iterative algorithm. You are given the following code snippet:

```java
int sum = 0;
for (int i = 1; i <= n; i = i * 2) {
    sum = sum + i;
}
print (sum);
```

a. How many times will line 3 be executed, as a function of n?

\[
\left\lfloor \log_2 n \right\rfloor + 1
\]

b. What is the output of the program if \( n = 32 \)?

\[
1 + 2 + 4 + 8 + 16 + 32 = 63
\]

\[
= 63
\]

c. In general, what is the output of the program if \( n = 2^k \), where \( k \) is some positive integer?

\[
\left\lfloor \log_2 (2^k) \right\rfloor + 1 = k + 1
\]

\[
= 2n - 1
\]
5. (20 points) Assume that $T(1) \in \Theta(1)$. Solve the following recurrence functions using the master method. If the master method cannot be applied, state the reason. Justify your answer.

a. $T(n) = 9T(n/9) + n$;

Note that

$n \in \Theta(n^{\log_3 9})$, so we can apply the master method to get

$T(n) \in \Theta(n \log n)$.

b. $T(n) = 3T(n/4) + n^2$;

Let $a = 3$, $b = 4$, then $a < b$. Note that

$n^2 \in \Theta(n^{\alpha+\epsilon})$, $0 < \epsilon \leq 2 - \alpha$.

Since $3 \left(\frac{n}{4}\right)^2 = \frac{3}{16} n^2$, we can apply the master method to get

$T(n) \in \Theta(n^2)$.

$$3 \left(\frac{n}{4}\right)^2 \leq cn^2$$

for $c > \frac{3}{16}$.
c. \( T(n) = 9T(n/3) + n; \)

Note that
\[ n \leq O(n^{\log_3 9} - e) \quad \text{if } e \leq 1, \]
So we can apply the master method to get
\[ T(n) = \Theta(n^2). \]

\[ \square \]

d. \( T(n) = nT(n/2) + n^3; \)

We cannot apply the master method because \( T(n) \) is not of the form \( aT(n/b) + f(n) \), where \( a \) and \( b \) are constants.

\[ \square \]

e. \( T(n) = 4T(n/2) + n / \log n; \)

Note that
\[ \frac{n}{\log n} \leq O\left(n^{\log_2 4} - e\right) \quad \text{if } e \leq 1, \]
so we can apply the master method to get
\[ T(n) = \Theta(n^2). \]
6. (10 points) Recursion tree method.

Assume that $T(1) \in \Theta(1)$. Solve the following recurrence function using the recursion tree method to get an asymptotically tight bound.

$$T(n) = 2T(n - 2) + 1$$

\[\sum_{i=0}^{h} 2^i = 2^{h+1} - 1 = \Theta(2^h) = \Theta(2^{n/2}) = \Theta(\sqrt{2^n})\]
7. (10 points) Substitution method.

Assume that $T(1) \in \Theta(1)$ and $T(n) = T(n/2) + 4T(n/4) + n^2$. Prove $T(n) \in O(n^2)$ using the substitution method.

Set $c = 2$ and $n_0 \geq 0$. Note that $T(n) \leq c$.

Assume that $n \geq n_0$.

$0 \leq T(n) = \frac{c}{4} n^2$ and

$0 \leq T(n) = \frac{c}{16} n^2$.

Then (since $\frac{1}{4} < \frac{1}{16}$)

$0 \leq T(n) = \frac{c}{4} n^2 - 4T(n/4) + n^2$

$\leq \frac{c}{4} n^2 + \frac{c}{16} n^2 + n^2$

$= \frac{2c+4}{16} n^2$

$\leq c n^2$.

Therefore $T(n) \in O(n^2)$. 


8a. (10 points) Analysis of recursive algorithms.
Consider the pseudocode of the following two algorithms. The input A is an array of size n. In AlgX, A is divided into 2 subarrays, and the algorithm is recursively applied to one of the subarrays. In AlgY, A is divided into 4 subarrays, and the algorithm is recursively applied to two of them. Assume that it takes constant time to divide A and pass the subarrays into the recursive function.

```
AlgX (A[1..n])
    if (n <= 1) return;
    p = floor(n/2);
    r = rand();
    if (r <= 0.5)
        AlgX (A[1..p]);
    else
        AlgX (A[p+1..n]);
end
```

```
AlgY (A[1..n])
    if (n <= 1) return;
    q = floor(n/4);
    r = rand();
    if (r <= 0.5)
        AlgY (A[1..q]);
        AlgY (A[2q+1..3q]);
    else
        AlgY (A[q+1..2q]);
        AlgY (A[3q+1..n]);
end
```

a. Write down the recurrence function for the running time of AlgX and solve it. (Hint: you may use the master method).

\[ T(n) = T(n/2) + f(n) \quad \text{where} \quad f(n) = \Theta(1). \]

\[ T(n) = \Theta(\log n). \quad \checkmark \]

b. Write down the recurrence function for the running time of AlgY and solve it. (Hint: you may use the master method).

\[ T(n) = 2T(n/4) + g(n) \quad \text{where} \quad g(n) = \Theta(n^c). \]

\[ T(n) = \Theta(n^c). \]

\[ \checkmark \]

c. Which algorithm is more efficient?

AlgX is more efficient. \quad \checkmark
8b. **(Extra credits: 20 points)** In AlgY on the previous page, if instead of dividing $A$ into 4 subarrays, let's say we divide it into $k$ subarrays, where $k$ is an even integer and is greater than 2. We then recursively apply the algorithm to half of the subarrays.

   a. What will be the running time of the algorithm in terms of $n$ and $k$? (Hint: write down the recurrence and solve it using the master method).

   $T(n) = \frac{k}{2} T\left( \frac{n}{k} \right) + \Theta(1)$

   **Master method case 1:**

   $T(n) = \Theta(n^{\log_{k} \frac{k/2}{2}}) = \Theta(n^{\frac{\log k}{1 - \log_{k} 2}})$

   $= \Theta(n^{1 - \log_{k} 2})$

   b. Does the efficiency of the algorithm improve or become worse when $k$ increases? Justify your answer.

   $k \uparrow \rightarrow \log_{k} \frac{k}{2} \downarrow \rightarrow n^{1 - \log_{k} 2} \uparrow$

   So it becomes worse.

   c. When answering the questions in 7 and 8(a)-(b), we have assumed that parameter passing during procedure calls takes constant time, regardless of the size of the array. This is valid in most systems because a pointer or reference to the array is passed, not the array itself. Now let's say that we have to pass the parameter by actually creating a copy of the subarray during procedure call and the time is $f(n) = \Theta(n)$ for passing an array of size $n$. Answer questions 7(a)-(c) and 8(a)-(b) again.

   8a. (a) $T(n) = T(n/2) + \Theta(n)$ $\implies T(n) = O(n)$

   (b) $T(n) = 2T(n/4) + \Theta(n)$ $\implies T(n) = O(n)$

   8b. (a) $T(n) = \frac{k}{2} T\left( \frac{n}{k} \right) + \Theta(n) \implies T(n) = O(n)$