1. **Universal and Existential Qualifiers** (8 marks)
   Recall that a problem is **computable** if and only if there is an algorithm that halts and returns the correct solution on every valid input. Let $\text{Works}(P, A, I)$ be true if and only if algorithm $A$ halts and correctly solves problem $P$ on input instance $I$. Let $P = \text{Halting}$ be the Halting problem which takes a Java program $I$ as input and tells you whether or not it halts on the empty string. Let $P = \text{Sorting}$ be the sorting problem which takes a list of numbers $I$ as input and sorts them.

   (a) (4 marks) Express using universal and existential qualifiers ($\forall, \exists$) that $\text{Sorting}$ is computable. No justification is required.
   - **Answer:** $\exists A, \forall I, \text{Works}($Sorting, $A, I$). There is at least one algorithm, eg. $A = \text{mergesort}$, that works for every input instance $I$.

   (b) (4 marks) Express using universal and existential qualifiers ($\forall, \exists$) that $\text{Halting}$ is not computable. No justification is required.
   - **Answer:** $\forall A, \exists I, \neg \text{Works}($Halting, $A, I$) Every algorithm fails to work for at least one input instance $I$.

2. **Bounds** (15 marks)
   We use the logical connective $\rightarrow$ to represent a logical implication, consequence or entailment. For example, the proposition Statement 1 $\rightarrow$ Statement 2 means that if Statement 1 is true, Statement 2 must also be true.
   
   If $f(n)$ and $g(n)$ are two functions of the positive integer $n$, which of the following propositions are true? Please prove the ones you believe to be true, and provide a counterexample for those you think false.

   (a) $f(n) \in \Theta(g(n)) \rightarrow g(n) \in \omega(f(n))$
   - **Answer:** False - in fact, the negation is true: $f(n) \in \Theta(g(n)) \rightarrow g(n) \notin \omega(f(n))$. If, for example, $g(n) = f(n)$, then $f(n) \in \Theta(g(n))$, but $g(n) \notin \omega(f(n))$. To see this, recall that $g(n) \in \omega(f(n)) \rightarrow \forall c > 0, \exists n_0 > 0 : \forall n \geq n_0, g(n) > cf(n)$.
     
     but clearly the latter is false for $c \geq 1$.

   (b) $f(n) \in \Omega(g(n)) \rightarrow g(n) \in \Theta(f(n))$
   - **Answer:** True. To see this recall that $f(n) \in \Omega(g(n)) \rightarrow \exists c_1, n_1 > 0 : \forall n \geq n_1, f(n) \geq c_1 g(n)$
     
     We must show that this implies that $g(n) \in \Theta(f(n))$, i.e., that
     
     $\exists c_2, n_2 > 0 : \forall n \geq n_2, g(n) \leq c_2 f(n)$,
     
     which follows by setting $c_2 = 1/c_1$ and $n_2 = n_1$.

   (c) $f(n) \in \Omega(g(n)) \rightarrow g(n) \in \omega(f(n))$
   - **Answer:** False, and, in fact, impossible. For example, $f(n) = g(n) \rightarrow f(n) \in \Omega(g(n))$, but $g(n) \notin \omega(f(n))$. To see this, recall that $g(n) \in \omega(f(n)) \rightarrow \forall c > 0, \exists n_0 > 0 : \forall n \geq n_0, g(n) < cf(n)$,
     
     which is false for $c \leq 1$.  

(d) $f(n) \in \Theta(g(n)) \Rightarrow g(n) \in \omega(f(n))$

- Answer: False. For example, suppose that $f(n) = g(n)$. Then clearly $f(n) \in \Theta(g(n))$, but $g(n) \notin \omega(f(n))$. To see this, recall that

$$g(n) \in \omega(f(n)) \Rightarrow \forall c > 0, \exists n_0 > 0 : \forall n \geq n_0, g(n) > cf(n),$$

but clearly the latter is false for $c \geq 1$.

(e) $f(n) \in \Theta(g(n)) \Rightarrow f(n) \in \Omega(g(n))$

- Answer: True. This follows from the fact that

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in \Omega(g(n)) \& f(n) \in \Omega(g(n)).$$

3. **Sums** (16 marks) Derive a $\Theta$-approximation for the following sums. Please show your work, but be concise.

(a) $\sum_{i=1}^{n} i \cdot 2^i$

- Answer: Let $S = \sum_{i=1}^{n} f(i)$, where $f(i) = i \cdot 2^i$. Since $f(n) \geq 2^n, \forall n \geq 1$, clearly $f(n) \in 2\Omega(n)$. Thus the sum is geometric-like, and $\sum_{i=1}^{n} i \cdot 2^i = \Theta(f(n)) = \Theta(n \cdot 2^n)$.

To solve this without relying upon the ‘geometric-like’ category, first note that the sum is clearly in $\Omega(n \cdot 2^n)$, since $f(n) = n \cdot 2^n$. We must also show that the sum is in $\Theta(n \cdot 2^n)$. Specifically, we show that $S \leq 2n \cdot 2^n$. This is so because $\sum_{i=1}^{n-1} f(i) < f(n) = n \cdot 2^n$. To see this, observe that $\sum_{i=1}^{n-1} f(i) < n \sum_{i=1}^{n-1} 2^i = n(2^n - 2)$.

(b) $\sum_{k=5}^{4n} \frac{1}{3k+7}$

- Answer: This sum looks harmonic. Let $f(k) = \frac{1}{3k+7}$. We claim that $f(k) \in \Theta(1/k)$. To see this, observe that $f(k) \geq \frac{1}{4k}$ as long as $4k \geq 3k + 7 \Leftrightarrow k \geq 7$. Thus $f(k) \in \Omega(1/k)$. Next, observe that $f(k) \leq \frac{1}{3k}$, thus $f(k) \in \Theta(1/k)$. Since $f(k) \in \Theta(1/k)$,

$$\sum_{k=5}^{4n} \frac{1}{3k+7} \in \Theta(\log(4n)) = \Theta(2 + \log n) = \Theta(\log n).$$

To solve this without relying upon the ‘harmonic-like’ category, observe that

$$\sum_{k=5}^{4n} \frac{1}{3k+7} \geq c + \frac{1}{4} \sum_{k=1}^{4n} \frac{1}{k} \in \Theta(\log(4n)) = \Theta(\log n),$$

where $c$ is a constant. Thus the sum is in $\Omega(\log n)$. Next observe that,

$$\sum_{k=5}^{4n} \frac{1}{3k+7} \leq \frac{1}{3} \sum_{k=5}^{4n} \frac{1}{k} \in \Theta(\log(4n)) = \Theta(\log n).$$

Thus the sum is in $\Theta(\log n)$ as well.

Note that it is also possible to obtain the same result using integration.

(c) $\sum_{i=2}^{n} \sum_{j=1}^{n} \frac{\log i}{j}$

- Answer: We have

$$\sum_{i=2}^{n} \sum_{j=1}^{n} \frac{\log i}{j} = \left( \sum_{i=2}^{n} \log i \right) \cdot \left( \sum_{j=1}^{n} \frac{1}{j} \right).$$
4. The Carnival Coin Game (25 marks)

You are running a game booth at your local village carnival. In your game you lay out an array of $1 coins on a table. For a $5 charge, anyone can challenge you to a game. In this game, you and your customer alternately pick coins from the table, either 1 or 2 at a time. If your customer can make you pick up the last coin, he walks away with all the coins. You graciously allow your customer to go first.

Write an iterative algorithm CarniCoin(n) that tells you how many coins to pick up on every turn, pick up the last coin, he walks away with all the coins. You graciously allow your customer to go first. customer alternately pick coins from the table, either 1 or 2 at a time. If your customer can make you

(a) (5 marks) What is the postcondition for your algorithm?

• Answer: One coin remains on the table, and it is the customer’s turn.

(b) (5 marks) What precondition must you place on $n$ in order to guarantee that you will be able to meet the postcondition?

\[ \sum_{i=2}^{n} \log i \text{ is arithmetic-like, and } \sum_{j=1}^{n} \frac{1}{j} \text{ is harmonic. Hence we expect that} \]

\[ \sum_{i=2}^{n} \frac{\sum_{j=1}^{n} \log i}{j} = \left( \sum_{i=2}^{n} \log i \right) \cdot \left( \sum_{j=1}^{n} \frac{1}{j} \right) = \Theta(n \log n) \cdot \Theta(\log n) = \Theta(n \log^2 n). \]

How do we know that the first sum is arithmetic-like? We claim that \( \log n \in \Theta(1) \). To see this, observe first that \( \log n \geq 1 \forall n \geq 2 \), and \( 1 = n^{1-1} \). Thus \( \log n \in \Theta(1) \). Second, observe that \( \log n \leq n \forall n \geq 1 \), and \( n = n^{2-1} \). Thus \( \log n \in \Theta(1) \).

However, our result also relies upon the assumption that \( \Theta(f_1(n)f_2(n)) = \Theta(f_1(n))\Theta(f_2(n)), \) i.e., that \( \Theta \) is distributive over multiplication. Is this true?

Suppose that \( f_1(n) \in \Theta(g_1(n)) \) and \( f_2(n) \in \Theta(g_2(n)) \). In other words,

\[ \exists c_{11}, c_{12}, n_{10} > 0 : \forall n \geq n_{10}, c_{11}g_1(n) \leq f_1(n) \leq c_{12}g_1(n) \]

and

\[ \exists c_{21}, c_{22}, n_{20} > 0 : \forall n \geq n_{20}, c_{21}g_2(n) \leq f_2(n) \leq c_{22}g_2(n) \]

This means that

\[ \exists c_{31}, c_{32}, n_{30} > 0 : \forall n \geq n_{30}, c_{31}g_1(n)g_2(n) \leq f_1(n)f_2(n) \leq c_{32}g_1(n)g_2(n) \]

where \( c_{31} = c_{11}c_{21}, c_{32} = c_{12}c_{22} \), and \( n_{30} = \max(n_{10}, n_{20}) \).

(d) \( \sum_{k=1}^{n} k^k \)

- Answer: Clearly \( f(k) = k^k \geq 2^k \forall k \geq 2 \), thus \( f(k) \in \Omega(n) \), the sum is geometric-like, and \( \sum_{k=1}^{n} k^k = \Theta(f(n)) = \Theta(n^n) \).

To solve this without relying upon the ‘geometric-like’ category, first note that the sum is clearly in \( \Omega(n^n) \), since \( f(n) = n^n \). We must also show that the sum is in \( O(n^n) \). Specifically, we show that \( S \leq 2n^n \). This is so because \( \sum_{k=1}^{n-1} k^k \leq f(n) = n^n \). To see this, observe that \( \sum_{k=1}^{n-1} k^k \leq n \cdot n^{n-1} = n^n \)
Answer: \( n > 0 \) and \( n \mod 3 = 1 \).

(c) (5 marks) What is the loop invariant for your algorithm?

- Answer: The number \( i \) of coins on the table satisfies \( i \mod 3 = 1 \).

(d) (5 marks) Concisely state your algorithm in pseudocode.

- Answer:

```pseudocode
i = n
while i > 1
    i = i - 3
end while
```

The loop invariant is maintained because 3 coins are picked up on each iteration, so that the number of coins on the table, mod 3, is maintained.

The loop invariant is true on entry by virtue of the precondition. As a measure of progress, we can use the coins remaining on the table, which, as shown above, decreases by 3 on every iteration, and will therefore meet the exit condition in finite time.

On exit, \( i \leq 1 \) and \( i \mod 3 = 1 \). Since \( i \) decreases by 3 every iteration, \( i > -2 \). Thus \( i = 1 \) and the postcondition is satisfied.

5. Tri-Sort (20 marks)

**Input:** An array \( A[1..n] \) of employee records. \( A[i].id \) and \( A[i].rank \), respectively indicate the id and the rank of the corresponding employee. The id field contains the employee name and additional relevant information about that employee. The rank field is 0, 1, or 2.

**Output:** Array \( A \) sorted by rank.

Design and analyze an iterative incremental and “in-place” algorithm to solve this problem in \( O(n) \) worst-case time. “In-place” means that in addition to array \( A \), the algorithm uses no more than \( O(1) \) additional variables, each of which has size no more than a single array element. (So you are not allowed to use additional arrays, lists, etc., with more than a constant number of cells.) Describe your algorithm using loop invariants. Prove that it is \( O(n) \).

- Answer:

**Loop Invariant:** At the beginning of iteration \( k \) we have the Loop Invariant:
  - \( A[1..n] \) is a permutation of the input array
  - Variables \( i, j, k \) satisfy \( 1 \leq i \leq j \leq k \leq n + 1 \)
  - \( A[1..i-1] \) all have rank 0, \( A[i..j-1] \) have rank 1, and \( A[j..k-1] \) have rank 2.

**Establishing:** The LI is trivially true by setting \( i = j = k = 1 \).

**Maintaining:** We examine the rank \( A[k].rank \).
  - If \( A[k].rank = 2 \): \( A[k] \) is already in place. We only increment \( k \).
  - If \( A[k].rank = 1 \): We swap \( A[k] \) and \( A[j] \) and increment both \( j \) and \( k \).

**Progress:** The measure of progress is that \( k \) always increases by 1. We start with \( k = 1 \) and exit when \( k > n \).

**Ending:** When \( k = n + 1 \), the loop invariant implies the elements are sorted.
6. **Longest Interval in Union** (16 marks) Given a list of intervals \([a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\), where \(a_1, \ldots, a_n, b_1, \ldots, b_n\) are integers, we are interested in the length of the longest interval in the union of the given intervals. For example, if the given intervals are \([0, 3], [2, 6], [7, 14]\) and \([8, 10]\), then their union is \([0, 6] \cup [7, 14]\) and the length of the longest interval in the union is \(14 - 7 = 7\).

   (a) (12 marks) Design an algorithm to solve this problem.
   (b) (4 marks) Analyze its running time.

Your algorithm must be as efficient as possible. Algorithms with running time \(\Theta(n^2)\) and above will receive only partial marks.

- **Answer:** The idea is to sort the intervals in the increasing order of their left endpoints, and then to try to group them together to form the longest possible interval. Denote the sorted intervals by \([l_1, r_1], [l_2, r_2], \ldots, [l_n, r_n]\). Then we have \(l_1 \leq l_2 \leq l_3 \ldots \leq l_n\).

Consider the example \([0, 4], [2, 7], [6, 10], [13, 15], [17, 28], [19, 23]\). We will use the example for illustration purposes only, and it is not part of the construction or the proof. The union in the example is \([0, 10] \cup [13, 15] \cup [17, 28]\). Note that the interval \([0, 10]\) in the union is the union of the first three intervals in the list, the interval \([13, 15]\) is the fourth interval in the list, and \([17, 28]\) is the union of the last two intervals. This is not a coincidence: an interval in the union is always a union of several consecutive intervals from the list. To show this we prove the following lemma.

**Lemma 1:** Suppose that \(i < k < j\) and the intervals \([l_i, r_i]\) and \([l_j, r_j]\) are in the same interval \(I\) of the union, then \([l_k, r_k]\) is also in \(I\).

**Proof:** \(i < k < j\) and the list is sorted, hence \(l_i \leq l_k \leq l_j\). We know that \(l_i\) and \(l_j\) are in \(I\), and hence \(l_k\) must also be in \(I\). \(I\) is one of the intervals in the union, so the fact that \([l_k, r_k]\) intersects \(I\) implies that \([l_k, r_k]\) is contained in \(I\), which completes the proof.

Suppose that we want to know the first interval \(I_1\) in the union. We know that \(I_1\) contains \([l_1, r_1]\). If \([l_1, r_1]\) intersects \([l_2, r_2]\), then it must also contain \([l_2, r_2]\). Otherwise, \(r_1 < l_2\) and no interval covers \((r_1, l_2)\), hence \([l_2, r_2]\) cannot be in \(I_1\), and by lemma 1, no other interval can be in \(I_1\). Continuing using the same argument gives us a procedure of how to obtain the list of intervals in the union from the original list.

We scan the list from left to right and add the new interval we see to the rightmost interval of the union if they intersect, or create a new interval if they don’t. On the previous example, it works as follows:

<table>
<thead>
<tr>
<th>Current Interval</th>
<th>Current Union</th>
<th>Add/Create New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>[0, 4]</td>
<td>[0, 4]</td>
<td>Create New</td>
</tr>
<tr>
<td>[2, 7]</td>
<td>[0, 7]</td>
<td>Add</td>
</tr>
<tr>
<td>[6, 10]</td>
<td>[0, 10]</td>
<td>Add</td>
</tr>
<tr>
<td>[13, 15]</td>
<td>[0, 10] ∪ [13, 15]</td>
<td>Create New</td>
</tr>
<tr>
<td>[17, 28]</td>
<td>[0, 10] ∪ [13, 15] ∪ [17, 28]</td>
<td>Create New</td>
</tr>
<tr>
<td>[19, 23]</td>
<td>[0, 10] ∪ [13, 15] ∪ [17, 28]</td>
<td>Add</td>
</tr>
</tbody>
</table>

Formally, we give the following algorithm for making a list of intervals in the union from a sorted list of intervals.

**Algorithm** MakeUnion\(([l_1, r_1], \ldots, [l_n, r_n])\)

< **pre-cond**: \(l_1 \leq l_2 \leq \ldots \leq l_n\)

< **post-cond**: The disjoint intervals of the union

- Time complexity: We spend \(O(1)\) time per iteration for \(n\) iterations for a total of \(O(n)\) time.
begin
  unionlist = [l1, r1]
  for i=2 to n do
    [a, b] = I = last(unionlist)
    if b ≥ li then
      if ri > b
        I = [a, ri]
        Update I in unionlist
      endif
    else
      add(unionlist, [li, ri])
    endif
  endfor
  output unionlist
end algorithm

The algorithm performs Θ(1) operations on each iteration, hence it runs in time Θ(n). To prove correctness we establish the following loop invariant.

LI: After the i-th of the algorithm, UnionList contains disjoint intervals from left to right whose union is ∪j=1i [li, ri], (here i = 1, 2, 3, ..., n)

Observe that the LI after the last n-th iteration of the algorithm implies that the algorithm gives the correct output.

We use induction to prove LI. For the base case i = 1, UnionList contains [l1, r1], and the statement of LI is trivially true.

For the step, assume that LI holds after i iterations, we want to show that it holds after i + 1.

Denote by Ii = [ai, bi] the rightmost interval in UnionList after i iterations. Note that ai is the left endpoint of one of the intervals [l1, r1], [l2, r2], ..., [li, ri], hence ai ≤ li+1. So one of two possibilities holds:

(a) [li+1, ri+1] intersects Ii and no other intervals in UnionList.
(b) [li+1, ri+1] is disjoint from the union ∪j=1i [li, ri].

In the first case li+1 ≤ bi, and the first if statement holds. If ri+1 > bi, adding the i + 1-st interval increases Ii to [ai, ri+1], otherwise [li+1, ri+1] ⊂ Ii, and no action is needed. Hence in this case the LI is preserved by the algorithm.

In the second case the first if statement fails, and the algorithm adds [li+1, ri+1] to the UnionList list. Note that the new interval [li+1, ri+1] is located to the right from any other interval in the union so far, and is disjoint from them. Hence the LI is preserved in this case, completing the correctness proof of MakeUnion.

Now it is very easy to solve the problem using MakeUnion. All we have to do is to sort the intervals, let MakeUnion to construct the union intervals (which will be disjoint), and find the longest one among these intervals.

algorithm Longest ([a1, b1], ..., [an, bn])
<pre-cond>: n intervals [a1, b1], ..., [an, bn]
<post-cond>: The length of the longest interval in the union.

begin
  1) Sort [a1, b1], ..., [an, bn] in the increasing order of the left ends to obtain [l1, r1], ..., [ln, rn] with
     l1 <= l2 <= ... <= ln. Use MergeSort to do the sorting.
  2) Run MakeUnion ([r1, r1], ..., [rn, rn]) to obtain a list UnionList.

3) Output the length of the longest interval in UnionList.
end algorithm

The correctness of the algorithm follows immediately from the correctness of **MakeUnion**. Step 1 takes $\Theta(n \log n)$ time as we know from the analysis of **MergeSort**. Step 2 takes $\Theta(n)$ from the analysis of **MakeUnion**. Step 3 obviously takes $\Theta(n)$ time. Hence the running time of the algorithm is $\Theta(n \log n) + \Theta(n) + \Theta(n) = \Theta(n \log n)$.