1 Under what circumstances might it be more appropriate to select an asymptotically "slower" sorting algorithm? Be as specific as possible. (1)

2 Which is "asymptotically greater", $n!$ or n^n? Explain your reasoning. (1)

3 Explain why the following program segment runs in $O(\lg n)$ time. (2)

```
while (n > 0) {
  System.out.print(n % 2);
  n /= 2;
}
```

4 Explain why it is important to understand when a particular problem is algorithmically "unsolvable." What can be done to tackle these types of problems? (2)

5 Consider *Euclid's Algorithm* given below. (4)

Algorithm:
Input: a, b: any two, positive integers.

```
while a ≠ b
  if a > b
    a ← a − b
  otherwise
    b ← b − a
return a
```

Example:
Input: $12, 18$

a	b
12	18
12	6
6	6

Output: 6

(a) Follow *Euclid's Algorithm* for each of the following pairs of numbers.

i. $40, 16$
ii. $75, 105$
iii. $156, 286$

(b) How is the result of *Euclid's Algorithm* "mathematically significant"? That is, what does this algorithm actually produce as a result?

(c) Implement *Euclid's Algorithm* in Java.

6 The *Sieve of Eratosthenes* is an ancient algorithm designed for finding prime numbers below a given maximum value. It accomplishes this by marking as non-prime all multiples of each prime, starting with 2. The following steps summarize how the sieve works. (6)

1. Create a list of consecutive integers from 2 through n: $[2, 3, 4, 5, ..., n]$.
2. Initialize p to 2, the smallest prime number.
3. Starting with $2p$, mark each multiple of p less than or equal to n as non-prime.
4. Set p to the next number greater than p that is not marked as non-prime. If no such number exists, stop.
5. When the algorithm stops, all numbers not marked non-prime are all the prime numbers below n.

Create the method, `Eratosthenes()`, that will take n as a parameter and use the *Sieve of Eratosthenes* to find all prime numbers less than or equal to n. Verify your implementation by having your method print every found prime number.