1. What is the time complexity of the following code fragments, with respect to n? Give your answer in Θ-notation.

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

(b) 
  \[ \begin{align*}
  k &= 1; \\
  \text{for } (i = 0; i < n; i++) \\
  \text{temp} &= k; \\
  \text{for } (j = 0; j < k; j++) \\
  \text{temp} &= \text{temp} + 1; \\
  k &= \text{temp};
  \end{align*} \]

(c) 
  \[ \begin{align*}
  k &= 1, x = 1; \\
  \text{for } (i = 0; i < n; i++) \\
  \text{for } (j = 0; j < x; j++) \\
  k &= k + 1; \\
  x &= x \times 2;
  \end{align*} \]

2. Order the following functions from smallest to largest asymptotic complexity. Identity any pairs of functions that have the same complexity (i.e. are Θ of each other).

(a) \(2^x\)
(b) \(\log x^2\)
(c) \(x!\)
(d) \(\sqrt{x!}\)
(e) \(\log(\sqrt{x \log x})\)
(f) \(\log^x x\)
(g) \(\sum_{i=1}^{x} i\)
(h) \(\sum_{i=1}^{x} x\)

3. Consider the following two algorithms for sorting an array \(A\) of \(n\) comparable values \(A[1], A[2], \ldots A[n]\).

Selection-Sort

\[ \begin{align*}
  \text{for } i = 1 \rightarrow n - 1 & \text{ do} \\
  jMin &= i; \\
  \text{for } j = i \rightarrow n & \text{ do} \\
  \text{if } A[j] < A[jMin] & \text{ then} \\
  jMin &= j; \\
  \text{swap } A[i] & \text{ and } A[jMin]
  \end{align*} \]

Insertion-Sort

\[ \begin{align*}
  \text{for } i = 2 \rightarrow n & \text{ do} \\
  \text{for } j = i \rightarrow 2 & \text{ do} \\
  \text{if } A[j] < A[jMin] & \text{ then} \\
  jMin &= j; \\
  \text{swap } A[j] & \text{ and } A[jMin] \\
  \text{else} & \text{ break}
  \end{align*} \]

(a) Find the best-case and worst-case runtimes of Selection-Sort and Insertion Sort. Your answers should be in Θ notation.

(b) True or False: In practice, Insertion-Sort will always run at least as fast as Selection-Sort.
(c) True or False: The worst-case runtime of Selection-Sort is $\Omega(n)$.

(d) True or False: The worst-case runtime of Insertion-Sort is $\Omega(n^2)$.

4. Given two sets $A$ and $B$ and their sizes $m$ and $n$, the following algorithm calculates the sets’ intersection (i.e. the elements they have in common). The sets are represented as sorted arrays.

```python
intersect(A, m, B, n):
    C = new empty set
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            if (A[i] == B[j])
                C.append(A[i])
                break
    return C
```

(a) What is the runtime complexity of this algorithm? Give your answer in $\Theta$-notation in terms of $m$ and $n$. (The `append` operation is constant-time.)

(b) Write an algorithm for the same task with a better runtime complexity.