Problem 1.

1. Termination: does not terminate as the recursion has no base case.
   Correct output: not applicable, it does not halt.
   Algorithm correctness: not correct as it does not terminate.

2. Termination: terminates.
   Correct output: produces the correct output for all possible \( n \geq 0 \).
   Algorithm correctness: correct algorithm.

3. Termination: terminates.
   Correct output: it does not produce the correct output for all \( n > 1 \), the loop iteration logic mess up variable values.
   Algorithm correctness: not correct as it does not produce the correct output for all possible positive valued of \( n \).

Problem 2. False
Suppose that this statement is true, then there exists a constant \( c \) and a number \( n_0 \) such that \( 3^n \leq c2^n \forall n \geq n_0 \). The last element is equivalent to \( \left(\frac{3}{2}\right)^n \leq c \forall n \geq n_0 \). However, \( \left(\frac{3}{2}\right)^n \to \infty \) as \( n \to \infty \), so \( \left(\frac{3}{2}\right)^n \leq c \) cannot be true for all \( n \geq n_0 \) for any constant \( c \).

Problem 3.
i loops counter: \( n, n/2, n/4, \ldots, 1 \)
i loop total iterations: \( \log n + 1 \)
j loops counter: \( i \), where \( i = n, n/2, n/4, \ldots, 1 \)
j loop total iterations: \( n + n/2 + n/4 + \ldots + 1 = 1 \cdot \left(2^{\log n + 1} - 1\right)/(2 - 1) = 2n - 1 \) [geometric series sum]
k loops counter: \( 1, 2, 4, 8, \ldots, n/4, n/2, n \)
k loop total iterations: \( (\log n + 1) \cdot j \) loops total iterations
Overall complexity = \( k \) loop total iterations = \( (\log n + 1) \cdot (2n - 1) = O(n\log n) \)

Problem 4. This is ordered list from slowest to fastest functions. You will still get full grade if you order them in the order list from fastest to slowest. Functions in a same group are \( \Theta \) of each other.

1. \( 1 \)

2. \( \sqrt{\log n} \)

3. \( \log n \)
4. \( \log^2 n, (\log n)^2 \)
5. \( 2\sqrt{2\log n} \)
6. \( \sqrt{n} \)
7. \( 3n + 2, 2^{\log n} \)
8. \( n \log n \)
9. \( 4n^2 / \sqrt{n} \)
10. \( 4^{\log n} \)
11. \( n^3 + n^2, n^3 \)
12. \( n^{\log \log n} \)
13. \( 3^{n/3} \)
14. \( 2^n \)
15. \( e^n \)
16. \( n3^n \)
17. \( 4^{n-1} \)
18. \( n! \)
19. \( (n + 1)! \)
20. \( 2^2^n \)
21. \( 2^{2^n+1} \)

Problem 5.

1. In this case, we will simply consider the area for every possible pair of the lines and find out the maximum area out of those.

\[
\begin{align*}
\text{maxVol} &= 0 \\
a\text{Maxi} &= -1 \\
a\text{Maxj} &= -1 \\
\text{for } i &= 1 \text{ to } n-1 \\
\{ \\
&\quad \text{for } j = i+1 \text{ to } n
\end{align*}
\]
\[
\{ \\
\quad \text{vol} = (j - i) \times \min(a_i, a_j) \\
\quad \text{if } \text{vol} > \text{maxVol} \\
\quad \quad \text{maxVol} = \text{vol} \\
\quad \quad \text{aMaxi} = i \\
\quad \quad \text{aMaxj} = j \\
\}
\]

output aMaxi, aMaxj

Time complexity: \(O(n^2)\). Calculating area for all \(\frac{n(n-1)}{2}\) height pairs.

2.

low = 1
high = n
maxVol = 0
aMaxi = -1
aMaxj = -1
while (low < high)
{
\quad \text{vol} = (\text{high} - \text{low}) \times \min(a[\text{low}], a[\text{high}])
\quad \text{if } \text{vol} > \text{maxVol}
\quad \quad \text{maxVol} = \text{vol}
\quad \quad \text{aMaxi} = \text{low}
\quad \quad \text{aMaxj} = \text{high}
\quad \quad \text{if } a[\text{low}] < a[\text{high}]
\quad \quad \quad \text{low} = \text{low} + 1
\quad \quad \quad \text{else } \text{high} = \text{high} - 1
\}
output aMaxi, aMaxj

The intuition behind this approach is that the area formed between the lines will always be limited by the height of the shorter line. Further, the farther the lines, the more will be the area obtained.

We take two pointers, one at the beginning and one at the end of the array constituting the length of the lines. Further, we maintain a variable \(\text{maxVol}\) to store the maximum area obtained till now. At every step, we find out the area formed between them, update \(\text{maxVol}\) and move the pointer pointing to the shorter line towards the other end by one step.

**Proof:** Use \(v[\text{low}, \text{high}]\) indicates the volume of container with low and high. Suppose
$a[low] < a[high]$, then we move low to $low + 1$, that means we ignored $v[low, high - 1], v[low, high - 2], \ldots, \text{etc.}$ If this is safe, then the algorithm is right, and it’s obvious that $v[low, high - 1], v[low, high - 2], \ldots$ can’t be larger than $v[low, high]$ since its width can’t be larger than $high - low$, and its height is limited by $a[low]$.

Time complexity : $O(n)$. Single pass.