Exercise 1.

(a) \( T(n) = 5 \cdot T\left(\frac{n}{2}\right) + \mathcal{O}(n) \). The Master Theorem with \( d = 2, \alpha = \log_2 5, \beta = 1 \) implies, since \( \alpha > \beta \), that \( T(n) = \mathcal{O}(n^{\log_2 5}) = \mathcal{O}(n^{1.322}) \).

(b) \( T(n) = 2 \cdot T(n - 1) + \mathcal{O}(1) \). The theorem on linear reductions with \( c = 2, k = 0 \) implies, since \( c > 1 \), that \( T(n) = \mathcal{O}(2^n) \).

(c) \( T(n) = 9 \cdot T\left(\frac{n}{3}\right) + \mathcal{O}(n^2) \). The Master Theorem with \( d = 3, \alpha = 2, \beta = 2 \) implies, since \( \alpha = \beta \), that \( T(n) = \mathcal{O}(n^2 \log n) \).

It follows that algorithm C has the best asymptotic running time.

Exercise 2. Base case: \( T(2^1) = 2 \cdot 0 + (2^1 - 1) = 1 \), same as \( 2^1 \cdot (\log_2 2^1 - 1) + 1 = 2^1 \cdot 0 + 1 = 1 \)

Inductive step: \[
T(2^{k+1}) = 2 \cdot T\left(\frac{2^{k+1}}{2}\right) + (2^{k+1} - 1) \\
= 2 \cdot T(2^k) + (2^{k+1} - 1) \\
= 2 \cdot (2^k \cdot (\log_2 2^k - 1) + 1) + (2^{k+1} - 1) \\
= 2^{k+1} \cdot (\log_2 2^k - 1) + 2 + 2^{k+1} - 1 \\
= 2^{k+1} \cdot (\log_2 2^k - 1) + 1 \\
= 2^{k+1} \cdot \log_2 2^k + 1 \\
= 2^{k+1} \cdot \log_2 2^{k+1} + 1
\]

Exercise 3. The worst case is when the element occurs last in the list (or not at all). Let \( T(n) \) be the total cost of running \text{Search}(x, [x_1, \ldots, x_n]) \) in this case.

- if \( x_1 = x \) then return yes \hspace{0.5cm} \text{cost} = 1 \) (one list element comparison)
- else if \( n > 1 \) then return \text{Search}(x, [x_2, \ldots, x_n]) \hspace{0.5cm} \text{cost} = T(n-1) \) (recursive call with list size \( n - 1 \))
- else return no \hspace{0.5cm} \text{cost} = 0

This can be described by the recurrence \( T(1) = 1; \ T(n) = 1 + T(n-1) \) with the solution \( T(n) = \mathcal{O}(n) \).

Exercise 4. Again, the worst case is when the element occurs last in the list (or is larger than the last element). Let \( T(n) \) be the total cost of running \text{BinarySearch}(x, [x_1, \ldots, x_n]) \) in this case.

- if \( n = 0 \) then return no \hspace{0.5cm} \text{cost} = 0
- else if \( x_{[\frac{n}{2}]} > x \) then return \text{BinarySearch}(x, [x_1, \ldots, x_{[\frac{n}{2}]-1}]) \hspace{0.5cm} \text{cost} = 1 \) (one list element comparison; this condition is not satisfied when \( x \) occurs last in the list)
- else if \( x_{[\frac{n}{2}]} < x \) return \text{BinarySearch}(x, [x_{[\frac{n}{2}]+1}, \ldots, x_n]) \hspace{0.5cm} \text{cost} = 1 + T(\left\lfloor \frac{n}{2} \right\rfloor) \) (one comparison plus cost of recursive call with the second half of the list)
- else return yes \hspace{0.5cm} \text{cost} = 0

This can be described by the recurrence \( T(0) = 0; \ T(n) = 2 + T\left(\frac{n}{2}\right) \) with the solution \( T(n) = \mathcal{O}(\log n) \).