1 Problems

1. Write a recursive algorithm to check whether an integer \( x \), exists in an array \( A \) of \( n \) integers.

Solution: Algorithm 1.1 searches for an integer \( x \) in array \( A[n] \). It returns \( \text{true} \), if \( x \in A \) and \( \text{false} \) otherwise.

\[
\begin{align*}
\text{Function} & \quad \text{ARRAY-SEARCH}(A, x, n) \\
1: & \quad \text{if } (n = 1) \text{ then} \\
2: & \quad \text{if } (A[1] = x) \text{ then} \\
3: & \quad \hspace{0.5cm} \text{return}(\text{true}) \\
4: & \quad \text{else} \\
5: & \quad \hspace{0.5cm} \text{return}(\text{false}) \\
6: & \quad \text{end if} \\
7: & \quad \text{else} \\
8: & \quad \text{if } (A[n] = x) \text{ then} \\
9: & \quad \hspace{0.5cm} \text{return}(\text{true}) \\
10: & \quad \text{else} \\
11: & \quad \hspace{0.5cm} \text{ARRAY-SEARCH}(A, x, n - 1) \\
12: & \quad \text{end if} \\
13: & \quad \text{end if}
\end{align*}
\]

Algorithm 1.1: Array Search

2. Argue the correctness of your algorithm using induction.

Solution: Let \( P(n) \) denote the proposition that Algorithm 1.1 correctly searches for \( x \) in an array of size \( n \).

Base case \((n = 1)\): Observe that when \( n = 1 \), only lines 1 through 5 of Algorithm 1.1 are executed. If \( x \in A \), then the conditional in the \text{if} statement of line 2 is satisfied and hence \text{true} is returned. Likewise, if \( x \not\in A \), the conditional is falsified and hence line 5 of Algorithm 1.1 is executed, i.e., \text{false} is returned by Algorithm 1.1. We have thus established that when there is only one element in array \( A \), Algorithm 1.1 functions correctly.

Inductive Hypothesis: Assume that \( P(k) \) is \text{true}, i.e., assume that when Algorithm 1.1 is presented with an integer \( x \) and an array \( A \) of exactly \( k \) elements, then it returns \text{true} when \( x \in A \) and \text{false} otherwise.

Now consider the case in which Algorithm 1.1 is presented with an array of size \( k + 1 \) and asked to search for the presence of an integer \( x \). We consider the following two cases:

(i) \( x = A[k + 1] \): In this case, line 9 of Algorithm 1.1 is executed. Since \( x \in A \), the algorithm functions correctly.
(ii) \( x \neq A[k+1] \) - In this case, Algorithm 1.1 recurses over the first \( k \) elements of \( A \). From the inductive hypothesis, we know that Algorithm 1.1 functions correctly, when \( A \) has exactly \( k \) elements, i.e., if \( x \in A \), then \text{true} is returned and if \( x \notin A \), then \text{false} is returned.

We thus see that if Algorithm 1.1 functions correctly on arrays of size \( k \), then it also functions correctly on arrays of size \( k + 1 \); using the first principle of mathematical induction, we conclude that Algorithm 1.1 functions correctly for all \( n \geq 1 \).

3. Provide upper and lower bounds on \( S = \sum_{i=1}^{n} i \cdot \log i \).

\textbf{Solution:} We use the integration bounds discussed in class. Observe that \( i \cdot \log i \) is an increasing function of \( i \). Hence, we must have,

\[
S \leq \int_{1}^{n+1} x \log x \, dx
\]

Observe that

\[
\int x \log x \, dx = \log x \int x - \int \left( \frac{d}{dx} (\log x) \cdot \int x \, dx \right) \, dx = \frac{x^2 \log x}{2} - \int \frac{x}{2} \, dx = \frac{x^2 \log x}{2} - \frac{x^2}{4}.
\]

Thus, \( S \leq \left[ \frac{(n+1)^2 \log(n+1)}{2} - \frac{(n+1)^2}{4} - \frac{1}{4} \right] \).

For the lower bound, observe that \( S = 1 \log 1 + \sum_{i=2}^{n} i \cdot \log i \). Applying the lower bound from calculus, we conclude that

\[
S \geq \int_{1}^{n} x \cdot \log x \, dx = \frac{n^2 \log n}{2} - \frac{n^2}{4} - \frac{1}{4}.
\]

4. Let \( T \) denote a proper binary tree with \( n \) nodes having height \( h \). Formally establish that \( h \leq \frac{n-1}{2} \).

We use induction on the number of nodes \( n \) in the tree \( T \).

\textbf{Base case} \( n = 1 \): In this case, the only node in \( T \) and therefore the height \( h \) of \( T \) is zero. Since \( h \leq \frac{1-1}{2} \), the base case is proven.

\textbf{Inductive hypothesis:} Assume that whenever the number of nodes in a proper binary tree is at most \( k \), the height of the tree is at most \( \frac{k-1}{2} \). Now consider a proper binary tree having \( (k+1) \) nodes; since \( T \) is proper, we know that:

(i) There are at least two leaves at level \( h \).

(ii) We can group the leaves at level \( h \) into pairs, such that both leaves of a pair are children of the same node at level \( h - 1 \).

Remove one such leaf pair, say \((l_1, l_2)\), which are children of node \( l \) at level \((h - 1)\). The resultant binary tree \( T' \) is still proper and has \( k - 1 \) nodes; let \( h' \) denote the height of \( T' \). As per the inductive hypothesis, we know that \( h' \leq \frac{k-2}{2} \). There are precisely two possibilities to consider:

(i) \( h' = h \) - In this case, \( h = h' \leq \frac{k-2}{2} \leq \frac{k}{2} \).

(ii) \( h' < h \) - In this case \( l_1 \) and \( l_2 \) were the only nodes at level \( h \) and hence \( h' = h - 1 \). As per the inductive hypothesis, \( h' \leq \frac{k-2}{2} \) and hence, \( h = h' + 1 \leq \frac{k-2}{2} + 1 = \frac{k}{2} \).
In either case, we have established that \( h \leq \frac{k}{2} \) and using mathematical induction, we can conclude that the height \( h \) of a proper binary tree \( T \) having \( n \) nodes is at most \( \frac{n-1}{2} \).

5. Consider the following recursive definition of \( T(n) \).

\[
T(1) = 1 \\
T(n) = n \cdot T(n-1), \quad n \geq 2.
\]

Show that \( \log(T(n)) \in \Omega(n \cdot \log n) \).

**Solution:** Observe that \( T(n) \) is in fact \( n! \) and hence you are asked to show that \( \log n! \in \Omega(n \cdot \log n) \).

Note that

\[
\begin{align*}
\log n! &= \sum_{i=1}^{n} \log i \\
&= \sum_{i=2}^{n} \log i \\
&\geq \int_{1}^{n} \log x \, dx \\
&= [x \cdot \log x]_{1}^{n} - [x]_{1}^{n} \\
&= (n \cdot \log n - (n - 1)) \\
&\geq n \cdot \log n - n \\
&\geq n \cdot \log n - \frac{n}{2} \log n \\
&= \frac{1}{2} n \cdot \log n
\end{align*}
\]

We can then conclude that \( \log T(n) \in \Omega(n \cdot \log n) \). \( \square \)