1 Problems

1. Given an array $A$ of $n$ integer elements, how would you find the second smallest element in $n + \log_2 n$ comparisons.

**Solution:** Consider the following algorithm:

```plaintext
Function FIND-2MIN(A, low, high)
1: $n = high - low + 1$
2: $S_{2min} = \emptyset$
3: If ($n = 1$) then
4: $min_w = A[1]$
5: $S_{2min} = \emptyset$.
6: return($min_w, S_{2min}$)
7: end if
8: if ($n = 2$) then
10: $min_w = A[1]$
11: Add $A[2]$ to $S_{2min}$
12: return($min_w, S_{2min}$)
13: else
15: Add $A[1]$ to $S_{2min}$
16: return($min_w, S_{2min}$)
17: end if
18: end if
19: {We know that $n \geq 3$}
20: mid = $\lfloor \frac{high+low}{2} \rfloor$
21: ($lmin_w, lS_{2min}) = \text{FIND-2MIN}(A, low, mid)$
22: ($rmin_w, rS_{2min}) = \text{FIND-2MIN}(A, mid + 1, high)$
23: if ($lmin_w \leq rmin_w$) then
24: $min_w = lmin_w$
25: $S_{2min} = lS_{2min} \cup rmin_2$
26: else
27: $min_w = rmin_w$
28: $S_{2min} = rS_{2min} \cup lmin_2$
29: end if
30: return($min_w, S_{2min}$)

Algorithm 1.1: Finding the two smallest elements in an array
```
The above algorithm returns the smallest element in the whole array $\min_w$ and a set $S_{2\min}$ of candidate elements for the second minimum element.

The number of comparisons is characterized by the following recurrence relation:

$$
T(1) = 0 \\
T(2) = 1 \\
T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1
$$

This recurrence is easily solved to get $T(n) = n - 1$.

The size of the candidate set $S_{2\min}$ can be characterized by the following recurrence:

$$
G(1) = 0 \\
G(2) = 1 \\
G(n) = 1 + G\left(\frac{n}{2}\right)
$$

$G(n)$ is easily seen to be $\log_2 n$. We can find the smallest element in $S_{2\min}$ using at most $\log_2 n$ comparisons; it thus follows that the second smallest element can be found in $n + \log_2 n$ comparisons. □

2. Indicate whether each of the following identities is true or false, giving a proof if true and a counterexample otherwise.

(a) $f(n) + o(f(n)) \in \Theta(f(n))$.

(b) $(f(n) \in O(g(n))) \land (g(n) \in O(h(n))) \Rightarrow (f(n) \in O(h(n)))$.

(c) $\log^{1/\epsilon} n \in O(n^\epsilon)$, $(\forall \epsilon) 0 < \epsilon < 1$.

(d) $2^n \in \Omega(5^{\log_2 n})$.

Solution:

(a) The key observation is that $o(f(n)) \in O(f(n))$. Also, $f(n) \leq f(n) + o(f(n))$; it follows that $f(n) + o(f(n)) \in \Theta(f(n))$.

(b) The premises state that $f(n) \leq c_1 g(n)$ and $g(n) \leq c_2 h(n)$. It follows that $f(n) \leq c_1 \cdot c_2 h(n)$ and hence $f(n) \in O(h(n))$.

(c) Observe that

$$
\lim_{n \to \infty} \frac{\log^{1/\epsilon} n}{n^\epsilon} \\
= \lim_{n \to \infty} \frac{1}{\epsilon} \log \log n \\
= 0 \text{ by applying L'Hospital's rule}
$$

The identity is therefore true.

(d) Observe that

$$
\lim_{n \to \infty} \frac{2^n}{5^{\log_2 n}} \\
= \lim_{n \to \infty} \frac{n^{\log 2}}{\log n 5} \\
\to \infty \text{ by applying L'Hospital's rule}
$$

It therefore follows that the identity is true. □
Function FIND-KLARGEST(A, k, n)
1: We assume that the array elements are stored in A[1] through A[n] and that k is an integer ∈ [1, n]. We also assume without loss of generality, we assume that the numbers are distinct.
2: if (n = 1) then
3: {k has to be 1 as well}
4: return(A[n])
5: end if
6: We consider a variation of the PARTITION() procedure in which elements larger than the pivot are thrown in the left subarray and elements smaller than the pivot are thrown in the right subarray.
8: Copy the elements larger than A[j] into a new array C and the elements smaller than A[j] into a new array D.
9: if ((k = j) then
10: return(A[j])
11: else
12: if (k < j) then
13: return(FIND-KLARGEST(C, k, (j − 1)))
14: else
15: return(FIND-KLARGEST(D, k − j, (n − j)))
16: end if
17: end if

Algorithm 1.2: Selection through Partition

3. Devise a Divide-and-Conquer procedure for computing the kth largest element in an array of integers. Analyze the asymptotic time complexity of your algorithm. (Hint: Use the Partition procedure discussed in class.)

Solution:

Algorithm (1.2) represents a Divide-and-Conquer strategy for our problem.

The worst case running time of the algorithm is captured by the recurrence:

\[
\begin{align*}
T(1) &= O(1) \\
T(n) &= T(n-1) + O(n)
\end{align*}
\]

This implies that algorithm runs in time \(O(n^2)\) in the worst case. ∎
4. Argue the correctness of the `MERGE()` procedure discussed in class. (Hint: Write a recursive version of `MERGE()` and then use induction.)

Solution:

```plaintext
Function MERGE(A, l1, h1, B, l2, h2, C, l3, h3)
1: {We assume that the arrays A[l1 : h1] and B[l2 : h2] are being merged into the array C[l3 : h3].}
2: if (A is empty) then
3:    Copy B into C
4:    return
5: end if
6: if (B is empty) then
7:    Copy A into C
8:    return
9: end if
10: if (A[l1] ≤ B[l2]) then
12:    MERGE(A, l1 + 1, h1, B, l2, h2, C, l3 + 1, h3)
13: else
14:    C[l3] = B[l2]
15:    MERGE(A, l1, h1, B, l2 + 1, h2, C, l3 + 1, h3)
16: end if

Algorithm 1.3: Recursive Merge

To prove the correctness of the algorithm, we use induction on the sum s of the elements in A and B.

Clearly if s = 1, the algorithm functions correctly, since it copies the non-empty array into C.

Assume that the algorithm works correctly when 1 ≤ s ≤ k, for some k > 1. Now consider the case in which s = k + 1. In this case, Step (10:) of the algorithm moves the smallest element in both A and B into the first position in C. This is followed by a recursive call on arrays whose total cardinality is at most k. By the inductive hypothesis, the recursive calls work correctly and since C[l3] is already in its correct place, we can conclude that the arrays A and B have been correctly merged into array C.

□

5. What is the value returned by Algorithm (1.4) when called with n = 10?

```plaintext
Function LOOP-COUNTER(n)
1: count = 0
2: for (i = 1 to n) do
3:    for (j = 1 to i) do
4:      for (k = 1 to j) do
5:        count++
6:      end for
7:    end for
8: end for
9: return(count)

Algorithm 1.4: Loop Counter

Solution:

For arbitrary n, it is clear that the value of `count` is given by:

\[
\text{count}(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} 1
\]
\begin{align*}
\quad = \sum_{i=1}^{n} \sum_{j=1}^{i} j \\
\quad = \sum_{i=1}^{n} \frac{i \cdot (i + 1)}{2} \\
\quad = \frac{1}{2} \left[ \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i \right] \\
\quad = \frac{1}{2} \cdot \left[ \frac{n \cdot (n + 1) \cdot (2n + 1)}{6} + \frac{n \cdot (n + 1)}{2} \right] \\
\quad = \frac{n \cdot (n + 1)}{4} \cdot \left[ \frac{2n + 1}{3} + 1 \right]
\end{align*}

It follows that \( \text{count}(10) = \frac{55}{2} \times 8 = 220. \) \( \square \)