Exercise Session 03

Solve the following exercises.

Exercise 1.
Run the merge sort algorithm on the following array of numbers: $[3, 41, 52, 26, 38, 57, 49, 9]$. Give the state of the array after five calls of the algorithm MERGE are performed during the execution of MERGE-SORT.

Solution 1.
The state of the array after five calls to MERGE are performed in MERGE-SORT is $[3, 26, 41, 52, 38, 57, 9, 49]$. One can verify this by performing a step-by-step unfolding of the recursion tree as done in class.

Exercise 2.
Solve exercise CLRS 2.3–3, 2.3–4, and 2.3–6.

Solution 2.
CLRS 2.3–3. Consider the following recurrence for $n = 2^k$ and $k \in \mathbb{N} \setminus \{0\}$.

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(n/2) + n & \text{if } n = 2^k \text{ for } k > 1 \end{cases}$$

We prove that $T(n) = n \lg n$ by induction on k.

Base Case ($k = 1$). For $k = 1$ we have that $n = 2^1 = 2$. By definition of T we have $T(2) = 2$ which is equals to $2 \lg 2 = n \lg n$.

Inductive Step ($k > 1$). Then since $n = 2^k$ we have that $n/2 = 2^{k-1}$. Hence we have

$$\begin{aligned} T(n) &= 2T(n/2) + n && \text{(by def. } T\text{)} \\ &= 2(n/2 \lg(n/2)) + n && \text{(by inductive hypothesis)} \\ &= n \lg(n/2) + n \\ &= n(\lg n - \lg 2) + n && \text{(recall that } \lg(a/b) = \log a - \lg b\text{)} \\ &= n(\lg n - 1) + n && \text{(recall that } \lg 2 = 1\text{)} \\ &= n \lg n - n + n \\ &= n \lg n \end{aligned}$$

This concludes the proof.

Another way to solve the same exercise is the following. Let $n = 2^k$ for some $k \in \mathbb{N} \setminus \{0\}$, we can rewrite the recurrence in terms of k as

$$T(2^k) = \begin{cases} 2 & \text{if } k = 1 \\ 2T(2^{k-1}) + 2^k & \text{if } k > 1 \end{cases}$$

Analogously, the equality $T(n) = n \lg n$, is equivalently restated as $T(2^k) = 2^k \lg 2^k = k \cdot 2^k$. We prove by induction on k that $T(2^k) = k \cdot 2^k$.

Base Case ($k = 1$). Then, $k \cdot 2^k = 2 = T(2^k)$.

Inductive Step ($k > 1$).

$$\begin{aligned} T(2^k) &= 2T(2^{k-1}) + 2^k && \text{(by def. } T\text{)} \\ &= 2((k-1) \cdot 2^{k-1}) + 2^k && \text{(by inductive hypothesis)} \\ &= 2(k \cdot 2^{k-1} - 2^{k-1}) + 2^k \\ &= k \cdot 2^k - 2^k + 2^k \\ &= k \cdot 2^k \end{aligned}$$

CLRS 2.3–4. The recursive variant of the insertion sort algorithm is described in the following pseudocode. The algorithm takes and array $A[1\,..\,n]$, an index $1 \leq p \leq n$, and sorts the subarray $A[1\,..\,p]$.

```
InsertionSort(A, p)
1  if p > 1
2      InsertionSort(A, p − 1)
3      // Insert A[p] into the sorted sequence A[1 .. p − 1]
4      key = A[p]
5      i = p − 1
6      while i > 0 and A[i] > key
7          A[i + 1] = A[i]
8          i = i − 1
9      A[i + 1] = key
```

In this case a meaningful input size n for the algorithm is p, which corresponds to the number of elements to be sorted. The worst-case running time for the above algorithm is described as the solution of the following recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1, \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$

The solution of the exercise ends here. For those who are interested, we show that the asymptotic worst-case running time of the recursive version of InsertionSort is still $\Theta(n^2)$. To this end we will prove that $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$.

To prove $T(n) = O(n^2)$ we show that for some $c > 0$, $T(n) \leq cn^2$ for all $n \geq 1$. We proceed by induction on n

Base Case ($n = 1$) In this case $T(1) = \Theta(1)$, therefore $T(1) \leq c_1$ for some positive constant c_1. By choosing $c \geq c_1$ we have $T(n) \leq c_1 \leq c = cn^2$.

Inductive Step ($n > 1$) In this case $T(n) = T(n-1) + \Theta(n)$. By inductive hypothesis and definition of Θ notation, there exist $c, c_2 > 0$ and $n_0 > 0$ such that $T(n) \leq c(n-1)^2 + c_2 n = cn^2 - 2cn + c + c_2 n$, for all $n \geq n_0$. Therefore we have

$$\begin{aligned} T(p) &\leq cn^2 - 2cn + c + c_2 n \\ &\leq cn^2 - 2cn + c + cn && \text{(choose } c \text{ such that } c \geq c_2) \\ &\leq cn^2 - 2cn + cn + cn && (n \geq 1) \\ &= cn^2 \end{aligned}$$

We prove now that $T(n) = \Omega(n^2)$ by showing that for some $d > 0$, $T(n) \geq dn^2$ for all $n \geq 1$. Again, we proceed by induction on n.

Base Case ($n = 1$) In this case $T(1) = \Theta(1)$, therefore $T(1) \geq d_1$ for some positive constant d_1. By choosing d satisfying $d \leq d_1$ we obtain $T(n) \geq d_1 \geq d = dn^2$.

Inductive Step ($n > 1$) In this case $T(n) = T(n-1) + \Theta(n)$. By inductive hypothesis and definition of Θ notation, there exist $d, d_2 > 0$ and $n_0 > 0$ such that $T(n) \geq d(n-1)^2 + d_2 n = dn^2 - 2dn + d + d_2 n$, for all $n \geq n_0$. Therefore we have

$$\begin{aligned} T(p) &\geq dn^2 - 2dn + d + d_2 n \\ &\geq dn^2 - 2dn + d_2 n && (d \geq 0) \\ &\geq dn^2 && (\text{if } d \leq d_2/2 \text{ then } -2dn + d_2 n \geq 0) \end{aligned}$$

CLRS 2.3–6. Using a binary search for determining the position where to insert the element at each iteration of the INSERTION-SORT does not improve the overall worst-case running time. This is because the insertion procedure still requires one to shift to the left the elements in the subarray to make room for the element prior to its insertion, and this requires a number of steps linear in the size of the currently sorted subarray $A[1\,..\,j-1]$.

Exercise 3.
Consider the problem of finding the smallest element in a nonempty array of numbers $A[1\,..\,n]$.

(a) Write an *incremental* algorithm that solves the above problem and determine its asymptotic worst-case running time.

(b) Write a *divide-and-conquer* algorithm that solves the above problem and determine its asymptotic worst-case running time.

(c) Assume that the length of A is a power of 2. Write a recurrence describing how many comparison operations (among elements of A) your divide-and-conquer algorithm performs, and solve the recurrence using the recursion-tree method.

Remark: count ONLY the comparisons performed among elements in A. E.g., a comparison like $i \leq A.length$ shall not be counted, whereas $A[i] \leq k$ where k is a variable storing some element of A shall be counted. Moreover, if you use expressions like $\min(A[i], A[j])$ for some indices i, j, that also counts as 1 comparison.

Hint: A full binary tree with n leaves has $n-1$ internal nodes (see CLRS B.5.3 pp.1177–1179).

Solution 3.
(a) An incremental algorithm to find the smallest element in an array $A[1\,..\,n]$ is the following.

```
SMALLEST(A)
1  min = A[1]
2  for i = 2 to A.length
3       if A[i] < min
4            min = A[i]
5  return min
```

The worst-case running time occurs when A is sorted in decreasing order, that is $A[1] > A[2] > \cdots > A[n]$, in which case the running time is

$$T(n) = c_1 + nc_2 + (n-1)c_3 + (n-1)c_4 + c_5 = \Theta(n)\,.$$

(b) A *divide-and-conquer* algorithm to find the smallest element in an array is presented in the pseudocode below. It takes an array $A[1\,..\,n]$, two indices p and q such that $1 \leq p \leq q \leq n$, and returns the value of the smallest element in the subarray $A[p\,..\,q]$.

```
SMALLEST(A, p, q)
1  if q = p
2       return A[p]
3  else
4       m = ⌊(p + q)/2⌋
5       min_L = SMALLEST(A, p, m)
6       min_R = SMALLEST(A, m + 1, q)
7       return min {min_L, min_R}
```

A meaningful size for the input is the number of elements of the subarray $A[p\,..\,q]$, i.e., $n = q - p + 1$. To simplify our analysis we will assume that $n = 2^k$ for some $k \in \mathbb{N}$. The asymptotic worst-case running time of SMALLEST is described by the following recurrence

$$T(n) = \begin{cases} a & \text{if } n \leq 1, \\ 2T(n/2) + b & \text{if } n > 1. \end{cases}$$

where a denotes the running time for lines 1–2, and b denotes the running time for lines 1, 4, and 7. Note that under the assumption that $n = 2^k$, the recursion tree describing the unraveling of the above recurrence is a full binary tree having internal nodes labelled with b and leaves labelled with a. Since the number of leaves corresponds to the number of elements in the array $A[1\,..\,n]$, we have that

$$T(n) = an + b(n-1) = (a+b)n - b = \Theta(n) \tag{1}$$

Equation (1) can be proved by induction on k where $n = 2^k$. Equivalently, we show that for $k \in \mathbb{N}$, $T(2^k) = a2^k + b(2^k - 1)$. Let us rewrite the recurrence in terms of k

$$T(2^k) = \begin{cases} a & \text{if } k = 0, \\ 2T(2^{k-1}) + b & \text{if } k > 0. \end{cases}$$

Base Case ($k = 0$). Then, $T(2^k) = a = a2^k + b(2^k - 1)$.

Inductive Step ($k > 0$). Then,

$$\begin{aligned} T(2^k) &= 2T(2^{k-1}) + b && \text{(by def. of } T\text{)} \\ &= 2(a2^{k-1} + b(2^{k-1} - 1)) + b && \text{(by inductive hypothsis)} \\ &= a2^k + b2^k - b \\ &= a2^k + b(2^k - 1) \end{aligned}$$

(c) The number of comparisons among elements of A for an input instance of size n is described by the following recurrence

$$C(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 2C(n/2) + 1 & \text{if } n > 1. \end{cases}$$

As discussed before, $C(n)$ corresponds to the number of internal nodes of a full binary tree having n leaves. Therefore, the number of comparisons is $n - 1$.

★ **Exercise 4.**

Solve exercise CLRS 2.3–7.

Solution 4.

We have to describe a $\Theta(n \log n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.

We start by formally defining the above computational problem

Input: A sequence of integer numbers $S = \langle x_1, x_2, \ldots, x_n \rangle$ and an integer x.

Output: TRUE if there exist $i, j \in \{1, \ldots, n\}$ such that $x = x_i + x_j$, FALSE otherwise.

The following algorithm solves the above computational problem. The function F takes as input an array $S[1\,..\,n]$ of integers, and an integer x.

```
F(S, x)
1  MERGE-SORT(S, 1, S.length)
2  for i = 1 to S.length
3       key = x − A[i]
4       j = BIN-SEARCH(S, 1, S.length, key)
5       if j ≠ 0
6            return TRUE
7  return FALSE
```

The worst-case running time of the algorithm occurs when it returns FALSE. The asymptotic worst-case running time is $\Theta(n \log n)$ because line 1 takes $\Theta(n \log n)$, and line 4, which takes $\Theta(\log n)$, is executed n times.