Chapter 5

DAST. Exercise 4. Solution.

5.1 Stability.

For each of the following sorts, prove or disprove whether they are stable.

1. Heap sort.
2. Bubble sort.
3. Counting sort.

Solution.

1. Heap sort. No, for a counterexample, consider an array consisting of n identical values. The array is already a heap, and therefore the build subroutine wouldn't swap elements. Yet, on the first iteration of heapsort, the root of the heap, which is also the first element of the array, will be moved to the end of the output array.

2. Bubble sort. Yes, let's suppose that x and y are two elements in the input array, such that $x = y$ and x appears before y in the input. By the definition of the bubble sort algorithm, which only swaps adjacent elements, x can only pass y if there is an iteration i such that:

   (a) $A_i = x$ and $A_{i+1} = y$
   (b) $A_i > A_{i+1}$

   And it's clear that both conditions can't satisfied simultaneously

3. Counting sort. Yes, let's consider again x and y are two elements in the input array, such that $x = y$ and x appears before y in the input. Denote by v, u the positions of x, y in A. By definition of the algorithm, we set the output in the fourth for loop. Since the loop iterates in decreasing order, we have that y is placed before x at position C_y. However, the value of C_y decreases immediately afterwards and cannot increase again (since no line in the fourth for loop is defined to increase any of the C values). Therefore, it must hold that when the loop reaches the position of x on the $i = v$ iteration, $C_x = C_y$ is strictly lower than $C_x = C_y$ in the $i = u$ iteration. This means that x will be placed before y in the output.

5.2 Sorts That Assume Input Properties.

In any of the following sections, we make another assumption about the input. For each section, write an algorithm that uses the assumption to sort the inputs and compute its running time. You don't have to provide a correctness proof.

1. $x_1, x_2, ..., x_n \in \mathbb{R}$ such that $\max_i\{x_i\} - \min_i\{x_i\} \leq 5n$ and $|x_i - x_j| > 1$ for any $i \neq j$.

2. $x_1, x_2, ..., x_n \in \mathbb{R}$ such that $\max_i\{x_i\} - \min_i\{x_i\} \leq 5n$ and $|x_i - x_j| > \Delta$ for any $i \neq j$, but it is unknown what Δ is. The running time should depend on Δ.

Solution.

1. We are going to use a variation of counting sort. First, we observe that if it is given that $x_i < 5n$ for any i, then combining the fact that $|x_i - x_j| > 1$ for any i, j, we can conclude that there is a one-to-one mapping between the xs and integers in the range $[5n]$, given by $\lfloor x_i \rfloor$. Otherwise, if $\lfloor x_i \rfloor = \lfloor x_j \rfloor$, it implies that $|x_i - x_j| \leq 1$. Therefore, in this case, we can count which elements appear in the mapping image, 'sort them', and associate them with their source in the range. Algorithm 1 does exactly that, but it also substitutes the minimal element for concentrating the image in $[5n]$. The running time of the algorithm is $\Theta(n)$ since we iterate over the array with a length of at most $5n$ a constant number of times.

```
1  Let $B$ be a $5n$-length array
2  Let $C$ be a $n$-length array
3  Let $\xi \leftarrow \min x$ for $i \in [5n]$ do
4      $B_i \leftarrow$ Null
5  end
6  for $i \in [n]$ do
7      $B_{\lfloor x_i - \xi \rfloor} \leftarrow x_i$
8  end
9  Let $j \leftarrow 0$
10 for $i \in [5n]$ do
11     if $B_i$ is not Null then
12         $C_j \leftarrow B_i$
13         $j \leftarrow j + 1$
14     end
15 end
16 return $C$
```

Algorithm 1: Sorts assuming $|x_i - x_j| > 1$ and $\max_i\{x_i\} - \min_i\{x_i\} \leq 5n$

2. We observe that if $\Delta > 1$, then Algorithm 1 sorts the numbers. However, if $\Delta < 1$, then one of the values of B might get overridden. As a result, the number of non-null elements in B after line (8) will be lower than n. Thus, the returned array will have a length less than n. To address this issue, we will use binary search to find the value of Δ. Then, we will transform the

array $x_1, x_2, ..., x_n$ to $\Delta^{-1} \cdot x_1, \Delta^{-1} \cdot x_2, ..., \Delta^{-1} \cdot x_n$, which satisfies the assumption of the above section, except that $\max x_i - \min x_i < 5n/\Delta$. This means that the running time of Algorithm 1 after modifications will be $\Theta(n/\Delta)$. For each guess of Δ, we will check if Algorithm 1 successfully sorts x into C. If it does, then we return. Otherwise, we check for $\Delta \leftarrow \frac{1}{2}\Delta$. In total, the running time will be the sum up to:

$$\begin{aligned} T(n,\Delta) &= 5n \cdot c + 5n \cdot 2 \cdot c + 5n \cdot 2^2 \cdot c + .. + 5n \cdot \Delta^{-1} \cdot c \\ &\leq c \cdot \Delta^{-1} 5n \left(\sum^{\infty} \frac{1}{2^i} \right) = c \cdot 10n/\Delta = O(n/\Delta) \end{aligned}$$

```
Let $\Delta \leftarrow 1$
$C \leftarrow$ junk at length $< n$
while $C$ length $< n$ do
    Let $x'_1, x'_2..x'_n \leftarrow \Delta^{-1} \cdot x_1, \Delta^{-1} x_2..\Delta^{-1} x_n$
    $C \leftarrow$ call to Algorithm 1 over $x'$s
      where $\max x'_i - \min x'_i < 5\Delta^{-1} n$
    $\Delta \leftarrow \Delta/2$
end
return $C$
```

Algorithm 2: Sorts assuming $|x_i - x_j| > \Delta$ and $\max_i\{x_i\} - \min_i\{x_i\} \leq 5n$

5.3 Camels.

Bob is willing to send Alice camels, $c_1, c_2, ..., c_n$ in a specific order. Unfortunately for him, Eve doesn't like him. Hence, she decides to spoil their order by stacking camels as follows: Eve chooses camel c_i and a position j, and then moves the ith camel c_i between c_j and c_{j+1}, She can move as many camels as she wishes. Alice, known for her strictness, wouldn't accept camels standing out of order. Luckily, Bob is allowed to send the camels again in their original order for five times, and Eve cannot choose the same camel twice. In other words, if Eve chooses to move c_i in the first attempt, then she cannot move it in any of the other four attempts.

Alice records the positions of the camels every time they come. Write an algorithm that will help Alice restore the order. (Prove correctness and compute the running time.)

Solution. Guideline: Let $A \in \mathbb{R}^{5 \times n}$ be the recordings listed by Alice, where $A_{i,j} = k$ if the jth camel came at position k on the ith attempt. We can think of the problem as a sorting problem. Denote by $A^\star$ an arbitrary arrangement of the camels. Been able to answer whether $A^\star_j \leq A^\star_{j'}$ regarding the original order, meaning that the inequality $A^\star_j \leq A^\star_{j'}$ is satisfied if the camel on the j position should be put before the camel on the j' position will allow us to sort $A^\star$ into the original order. We claim that Algorithm 3 is a true comparator for camels ordering.

Claim 5.3.1. *If the jth camel appears before the j'th camel in the original order, then Algorithm 3 returns $\leq$, otherwise it returns $>$.*

Proof. Suppose that the jth camel appears before the j'th camel in the original order. Then the only cases where the jth camel might appear after the j'th camel in one of the attempts are the ones where Eve moved at least one of them (That's a simple claim, complete solution should contains it's proof). Since Eve is restricted to touching each camel at most once, in at least 3 of the 5 attempts, both camels were not touched and their relative order did not change. Technically, $A_{i,j} \leq A_{i,j'}$ for at least 3 values of i, so in the for loop, the value of Δ will increase at least 3 times. Repeating the above, but changing the direction of the inequality, proves that if in the original order, j appears after j', then Δ cannot exceed 2. □

Now we have an algorithm. First, take an initial guess for $A^\star$ (For example, take $A^\star$ to be the setting of camels on the first attempt), Then call to your favorite sorting algorithm (in the comparison model), and use the comparator in Algorithm 3 for executing it. Incorrectness of the suggested algorithm implies the incorrectness of either our comparator or the incorrectness of the sorting algorithm which in both cases is contradiction. Since the running time of Algorithm 3 is $O(1)$, any choice of $\Theta(n \log n)$-time sorting algorithms results a $\Theta(n \log n)$-time algorithm for sorting the camels. That's because:

$$\begin{aligned} T(n) &\leq \text{Time(The sorting alg calls on each iteration to Algorithm 3)} \\ &\leq \Theta(n \log n) \cdot O(1) = \Theta(n \log n) \end{aligned}$$

```
Result: Compare Camels Pair $j, j'$.
Let $\Delta \leftarrow 0$
for $i \in [5]$ do
    if $A_{i,j} \leq A_{i,j'}$ then
        $\Delta \leftarrow \Delta + 1$
    end
end
if $\Delta \geq 3$ then
    return '$\leq$'
end
else
    return '$>$'
end
```

Algorithm 3: Camels Comparator.

5.4 Predecessor in Binary Search Tree

1. Write pseudocode for the predecessor subroutine.
2. Prove its correctness.

Solution. Correctness. Let z be a vertex in the tree such that z.key is lower (or equals) than x.key. Observe that by the BST property, either z is a left-descendant of x or z and x have a common ancestor w (which might be z) such that z.key $\leq w$.key $\leq x$.key. In the second case, it is clear that z cannot be the predecessor unless w is z, or in other words, z is a right-ancestor of x. By concatenating inequalities, we have that the lowest right-ancestor of x is the one closest from left to x among all its right-ancestors. Let us denote it by y. Now, observe that if z is a left-descendant of x, then it is greater than y and lower than x. Thus, if x.left is not empty, then the closest element to x from the left is the maximum of its left descendants. In the other case, where x.left is indeed empty, we have that y remains the closest element to x from the left in the whole tree.

Now, Algorithm 4 first checks if x.left is not empty. If so, it looks for x's maximal left-descendant. Otherwise, it climbs up until it reaches its first (lowest) right-ancestor.

```
Result: Return the predecessor of $x$.
if $x$.left is not Null then
    return Tree-Max($x$.left)
end
Let $y \leftarrow x$.parent
while $y$ is not Null and $x = y$.left do
    $x \leftarrow y$
    $y \leftarrow x$.parent
end
return $y$
```

Algorithm 4: Predecessor.