Luis Gascon, Ethan Webb, Femi Dosumu
COSC 336
March 5, 2023

Assignment 3

Exercise 1

Analyze the following recurrences using the method that is indicated. In case you use the Master Theorem, state what the corresponding values of a, b, and $f(n)$ are and how you determined which case of the theorem applies.

- $T(n) = 3T(\frac{n}{4}) + 3$. Use the Master Theorem to find a $\Theta(\cdot)$ evaluation, or say "Master Theorem cannot be used", if this is the case.

$$\begin{aligned} a = 3\, b = 4\, f(n) &= 3 \\ n^{\log_4 3} &\text{ vs. } 3 \\ n^{\log_4 3} &> 3 \qquad \text{case 1 holds} \\ \therefore \Theta(n^{\log_4 3}) & \end{aligned}$$

- $T(n) = 2T(\frac{n}{2}) + 3n$. Use the Master Theorem to find a $\Theta(\cdot)$ evaluation, or say "Master Theorem cannot be used", if this is the case.

$$\begin{aligned} a = 2\, b = 2\, f(n) &= 3n \\ n^{\log_2 2} &= n \\ n &\text{ vs. } 3n \qquad \text{case 2 holds} \\ \therefore \Theta(n \log n) & \end{aligned}$$

- $T(n) = 9T(\frac{n}{3}) + n^2 \log n$. Use the Master Theorem to find a $\Theta(\cdot)$ evaluation, or say "Master Theorem cannot be used", if this is the case.

"Master Theorem cannot be used"

Exercise 2

- $T(n) = 2T(n-1) + 1$, $T(0) = 1$. Use the iteration method to find a $\Theta(\cdot)$ evaluation for $T(n)$.

$$\begin{aligned} T(1) &= 2(1) + 1 \\ T(2) &= 2(2 \cdot 1 + 1) + 1 \\ T(3) &= 2(2(2 \cdot 1 + 1) + 1) + 1 \\ &\therefore \Theta(2^n) \end{aligned}$$

- $T(n) = T(n-1) + 1$, $T(0) = 1$. Use the iteration method to find a $\Theta(\cdot)$ evaluation for $T(n)$.

$$\begin{aligned} T(1) &= 1 + 1 \\ T(2) &= (1 + 1) + 1 \\ T(3) &= ((1 + 1) + 1) + 1 \\ &\therefore \Theta(n) \end{aligned}$$

- Give a $\Theta(\cdot)$ evaluation for the runtime of the following code:

```
i = n
while(i >= 1) {
  for (j=1; j <= n; j++)
      x = x+1
  i = i/2
}
```

$$\Theta(n \log n)$$

- Give a $\Theta(\cdot)$ evaluation for the runtime of the following code:

```
i = n
while(i >= 1) {
  for (j=1; j <= i; j++)
    x = x+1
  i = i/2
}
```

$$\Theta(n \log n)$$

Modifications to merge()

To obtain the correct amount of star pairs, we only had to modify the merge() function so that it increments the star-pair value if it meets the condition for a pair to be a star pair and at the end of the function, return that value. During the comparison stage of merge, we added an incremental counter whenever the element in the left is less than the one in the right.

Modifications to mergeSort()

The merge function is modified so it can keep track of the number of star pairs per recursive call. When mergeSort is called again, the return value from the previous recursive call is stored as the star pair variable. When it's time to merge, the star pair variable gets added by the return value of the merge function. After the base case has been satisfied, the number of star pairs is returned.

Array	Result
$[7, 3, 8, 1, 5]$	4
Input with 1000 elements	4,376
Input with 10,000 elements	61,321

The source code for the algorithm is located on the next page.

```
public static int merge(
List<Integer> a, int left, int mid, int right) {
    int star_pairs = 0;
    int l_length = mid - left + 1;
    int r_length = right - mid;
    int[] l_arr = new int[l_length];
    int[] r_arr = new int[r_length];

    for (int i = 0; i < l_length; i++)
      l_arr[i] = a.get(left + i);

    for (int i = 0; i < r_length; i++)
      r_arr[i] = a.get(mid + 1 + i);

    int i, j, k;
    i = j = 0;
    k = left;

    while (i < l_length && j < r_length) {
      if (l_arr[i] < r_arr[j]) {
        a.set(k, l_arr[i]);
        star_pairs++;
        i++;
      } else {
        a.set(k, r_arr[j]);
        j++;
      }
      k++;
    }

    while (i < l_length) {
      a.set(k, l_arr[i]);
      i++;
      k++;
    }

    while (j < r_length) {
      a.set(k, r_arr[j]);
      j++;
      k++;
    }
    return star_pairs;
  }
```

Figure 1: merge

```
static int mergeSort(List<Integer> a,
int left, int right, int star_pairs) {
    int mid = (left + right) / 2;
    if (left < right) {
      // Sort left half of the array
      star_pairs = mergeSort(a, left, mid, star_pairs);

      // Sort right half of the array
      star_pairs = mergeSort(a, mid + 1, right, star_pairs);

      // Merge the arrays
      star_pairs += merge(a, left, mid, right);
    }
    return star_pairs;
  }
```

Figure 2: mergeSort