Programing= 100

Tamir Krief, Iaian Milton, Blessing Abumere
COSC 336
9/26/2024

Assignment 3

Exercise 1.

(a) $T(n) = 3T(\frac{n}{4}) + 3$. a = 3, b = 4, $f(n) = 3$. $n^{\log_4^3}$ vs. 3. $n^{\log_4^3}$ is the winner because $f(n) = 3$ is a constant function and doesn't grow. Therefore the Theta evaluation for the recurrence is $T(n) = \Theta(n^{\log_4^3})$.

(b) $T(n) = 2T(\frac{n}{2}) + 3n$. a = 2, b = 2, $f(n) = 3$n. $n^{\log_2^2}$ or n vs. $3n$. This is a tie because $f(n) = 3$n has proportional growth to n. Therefore the Theta evaluation for the recurrence is $T(n) = \Theta(n \log n)$.

(c) $T(n) = 9T(\frac{n}{3}) + n^2$. a = 9, b = 3, $f(n) = n^2$. $n^{\log_3^9}$ or n^2 vs. n^2. This is a tie because both functions are equivalent. Therefore the Theta evaluation for the recurrence is $T(n) = \Theta(n^2 \log n)$.

Exercise 2:

(a) $T(n) = \Theta(2^n)$
(b) $T(n) = \Theta(n)$
(c) $\Theta(n \log n)$
(d) $\Theta(n)$

Programming Task.

Table 1: Programming Task Results

7,3,8,1,5	4
input-3.4.txt	248339
input-3.5.txt	24787869

Results are correcct

```java
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;

public class Assignment3 {
    public static void main(String[] args) {
        System.err.println(UP_Pairs(new int[]{7, 3, 8, 1, 5}) + " UP pairs in [7, 3, 8, 1, 5]");
        System.out.println("input-3.4.txt UP_Pairs: " + UP_Pairs(inputFile("input-3.4.txt")));
        System.out.println("input-3.5.txt UP_Pairs: " + UP_Pairs(inputFile("input-3.5.txt")));
    }

    public static int UP_Pairs(int[] arr) {
        if (arr == null) return 0;
        return merge(arr, 0, arr.length);
    }

    /**
     * Modified merge method to return UP pairs
     * @param A the main array
     * @param p left index
     * @param r right index
     * @return UP pair count
    */
    public static int merge(int[] A, int p, int r) {
        if (r - p <= 1) return 0; // No pairs if there's one or zero elements

        int mid = p + (r - p) / 2;

        int pairs = merge(A, p, mid) + merge(A, mid, r);

        // Create left and right subarrays
        int n1 = mid - p;
        int n2 = r - mid;

        int[] L = new int[n1];
        int[] R = new int[n2];

        // Fill left and right arrays
        for (int i = 0; i < n1; i++) {
            L[i] = A[p + i];
        }
        for (int j = 0; j < n2; j++) {
            R[j] = A[mid + j];
        }

        // Merge process
        int i = 0, j = 0, k = p;

        while (i < n1 && j < n2) {
            if (L[i] < R[j]) {
                pairs += n2 - j; // Count remaining elements in R
                A[k++] = L[i++];
            } else {
                A[k++] = R[j++];
            }
        }

        // Copy remaining elements
        while (i < n1) {
            A[k++] = L[i++];
        }
        while (j < n2) {
            A[k++] = R[j++];
        }
```

```
        return pairs;
    }

    /**
     * Reads the file and converts it to an int array
     */
    public static int[] inputFile(String filename) {
        try (Scanner input = new Scanner(new FileReader(filename))) {
            // Size is the first number
            final int n = input.nextInt();
            int[] A = new int[n];

            // Reads ints in the file
            for (int i = 0; i < n; i++) {
                A[i] = input.nextInt();
            }

            return A;
        } catch (FileNotFoundException e) {
            System.err.println("File not found: '" + filename + "'");
        }
        return null;
    }
}
```