Romerico David, Haley Elliott, Julian Halsey, Nkechiyem Molokwu
COSC 336
03/05/2024

Assignment 3

Exercise 1:

1. $a = 3, b = 4, f(n) = 3$
$n^{log_4 3}$ vs. 3
$=> n^{0.79}$ vs. 3
$=> n^{0.79}/3$ because $n^{0.79}$ is larger than 3
$\therefore$ T(n) = $\Theta(n^{log_4 3})$ since n is still a polynomial

2. $a = 2, b = 2, f(n) = 3n$
$n^{log_2 2}$ vs. $3n$
$=> n$ vs. $3n$
$\therefore$ T(n) = $\Theta(n * log_2 n)$ since there was a tie

3. $a = 9, b = 3, f(n) = n^2 \log n$
$n^{\log_3 9}$ vs $n^2 \log n$
$= n^2$ vs $n^2 \log n$
$=> \frac{n^2 \log n}{n^2} = \log n$ is not a polynomial
$\therefore$ Master Theorem cannot be used

Exercise 2:

1. $T(n) = 2T(n-1) + 1, T(0) = 1$
$= 2[2T(n-2) + 1] + 1$
$= 2^2 T(n-2) + 2$
$= 2^2[2T(n-3) + 1] + 2$
$= 2^3 T(n-3) + 3$
$= 2^n T(n-k) + k$
Assume $n - k = 0 \therefore k = n$
$=> 2^n T(0) + n$
$= 2^n(1) + n$
$T(n) = \Theta(2^n)$

2. $T(n) = T(n-1) + 1, T(0) = 1$
$= [T(n-2) + 1] + 1$
$= T(n-2) + 2$
$= [T(n-3) + 1] + 2$
$= T(n-3) + 3$
$= T(n-k) + k$
Assume $n - k = 0 \therefore k = n$
$=> T(0) + n$
$= 1 + n$
$T(n) = \Theta(n)$

3. $T(n) = \Theta(n \log(n))$

4. $T(n) = n + \frac{n}{2} + \frac{n}{2^2} + ... = \frac{n}{2^k}$
where $k = \log n$
$= n(1 + \frac{1}{2} + \frac{1}{2^2} + ... + \frac{1}{2^k})$
$= n\frac{1-\frac{1}{2}^{\log(n)+1}}{1-\frac{1}{2}}$
$= \frac{n}{\frac{1}{2}} = 2n$
$T(n) = \Theta(n)$

Programming Task:

Description of Algorithm:

Our algorithm implements Merge Sort in order to achieve $\Theta(n \log n)$ running time. We slightly modified the Merge Sort algorithm so that it recursively calculates and sums the number of pairs in each traversal of the left and right halves of the original list. If the current element of the left half is less than the current element of the right half, and because the right half is sorted, we can conclude that the current element of the left half is less than all of the following elements in the right half. Therefore, the number of pairs can be incremented by the number of remaining elements in the right half.

Code:

```
public static int mergeSort(int[] A, int l, int r, int pairs) {
    if (l < r) {
        int m = l + (r - l) / 2;
        pairs = mergeSort(A, l, m, pairs);
```

```java
            pairs = mergeSort(A, m + 1, r, pairs);
            pairs += merge(A, l, m, r);
        }
        return pairs;
    }

    public static int merge(int[] A, int l, int m, int r) {
        int pairs = 0;
        int nL = m - l + 1; // length of left half
        int nR = r - m; // length of right half

        int[] L = new int[nL]; // left half array
        int[] R = new int[nR]; // right half array

        for (int i = 0; i < nL; i++)
            L[i] = A[l + i]; // putting elements into left half

        for (int j = 0; j < nR; j++)
            R[j] = A[m + j + 1]; // putting elements into right half

        int i = 0;
        int j = 0;
        int k = l;

        /*
         * The loop merges the left and right halves together
         * and calculates the number of pairs as follows:
         * If the element at the left half is less
         * than the element at the right half,
         * then all of the following elements of the right half are
         * also greater than the current left element.
         * So, pairs is increased by
         * the number of remaining elements in the right array
         */

        while (i < nL && j < nR) {
            if (L[i] < R[j]) {
                pairs += R.length - j;
```

```
                A[k] = L[i++];
            } else
                A[k] = R[j++];
            k++;
        }

        /*
         * Leftover elements in either the left or right halves are
         * appended at the end of the A array
         */

        while (i < nL)
            A[k++] = L[i++];

        while (j < nR)
            A[k++] = R[j++];

        return pairs;

    }
```

Results Table:

Data Set #	*-pairs
1	4
2	248,339
3	24,787,869

Table 1: Programming Task