CS 4349: Assignment 3

Written by Zach Leach, NetID: zcl190002

Draft February 19, 2024

1 Master Theorem

Give Asymptotic upper and lower bounds (e.g., big-Θ) for $T(n)$ in each of the following recurrences. Assume $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible.

1.1 $T(n) = 16(n/4) + n^2$

$T(n) = \Theta\left(n^2 \lg n\right)$, per Master Theorem case 2.

1.2 $T(n) = 7(n/3) + n^2$

$T(n) = \Theta\left(n^2\right)$, per Master Theorem case 3.

1.3 $T(n) = 7(n/2) + n^2$

$T(n) = \Theta\left(n^{\lg 7}\right)$, per Master Theorem case 1.

1.4 $T(n) = 2(n/4) + \sqrt{n}$

$T(n) = \Theta\left(\sqrt{n} \lg n\right)$, per Master Theorem case 2.

2 Substitution Method

3 Strassen's Algorithm

```
/* return the four submatrices A11, A12, A21, A22 */
function divide_matrix(matrix M)

/* return the combined matrix C */
function combine_matrices(matrix C11, matrix C12,
    matrix C21, matrix C22)

function strassen(matrix A, matrix B) {
       n = size of matrices A and B
       if (n == 1) {
              return A * B
       }
       else {
              A11, A12, A21, A22 = divide_matrix(A)
              B11, B12, B21, B22 = divide_matrix(B)

              P1 = strassen(A11 + A22, B11 + B22)
              P2 = strassen(A21 + A22, B11)
              P3 = strassen(A11, B12 - B22)
              P4 = strassen(A22, B21 - B11)
              P5 = strassen(A11 + A12, B22)
              P6 = strassen(A21 - A11, B11 + B12)
              P7 = strassen(A12 - A22, B21 + B22)

              C11 = P1 + P4 - P5 + P7
              C12 = P3 + P5
              C21 = P2 + P4
              C22 = P1 - P2 + P3 + P6

              return combine_matrices(C11, C12, C21,
                   C22)
       }
}
```

4 Recursion Tree

5 Local minimum pseudocode

```
function findLocalMinimum(arr, low, high) {
       if (low > high) {
              return
       }
       /* calculate if mid element is a local minimum */
       mid = (low + high) / 2

       /* check if mid is a local minimum */
       if (mid == 0 or arr[mid-1] > arr[mid]) {
              if (mid == n-1 or arr[mid] < arr[mid+1])
                   {
                     return mid
              }
       }

       /* if left neighbor is less than mid, recurse on
            left half */
       else if (mid > 0 and arr[mid-1] < arr[mid]) {
              return findLocalMinimum(arr, low, mid-1)
       }

       /* otherwise, recurse on right half */
       else {
```

```
            return findLocalMinimum(arr, mid+1, high)
        }
}
```