2025 Algorithm hw3

109062327 Hsu, Hung-Che

April 2025

Problem 1: Asymptotic Upper Bounds of Recurrences

We are given the following recurrences. We will use the Master Theorem to find the asymptotic upper bounds.

(a) $T(n) = 2T(n/4) + 1$

Here, we have:

$$a = 2, \quad b = 4, \quad f(n) = 1.$$

The critical exponent is

$$n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}}.$$

Since

$$f(n) = 1 = \Theta(n^0),$$

and $0 < \frac{1}{2}$, we are in **Case 1** of the Master Theorem (if $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$). Therefore,

$$T(n) = \Theta\left(n^{\frac{1}{2}}\right).$$

Thus, the asymptotic upper bound is:

$$T(n) = O\left(n^{0.5}\right).$$

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

Here, we have:

$$a = 2, \quad b = 4, \quad f(n) = \sqrt{n} = n^{0.5}.$$

We still have:

$$n^{\log_b a} = n^{\log_4 2} = n^{0.5}.$$

Since $f(n) = \Theta\left(n^{0.5}\right)$, we are in **Case 2** of the Master Theorem. Therefore,

$$T(n) = \Theta\left(n^{0.5} \log n\right).$$

Thus, the asymptotic upper bound is:

$$T(n) = O\left(n^{0.5} \log n\right).$$

(c) $T(n) = 2T(n/4) + n^2$

Here, we have:

$$a = 2, \quad b = 4, \quad f(n) = n^2.$$

Again,

$$n^{\log_b a} = n^{\log_4 2} = n^{0.5}.$$

Since n^2 grows much faster than $n^{0.5}$ (indeed, $n^2 = \Omega\big(n^{0.5+\epsilon}\big)$ for any $\epsilon \leq 1.5$), we are in **Case 3** (assuming the regularity condition holds). Hence,

$$T(n) = \Theta\left(n^2\right).$$

Thus, the asymptotic upper bound is:

$$T(n) = O\left(n^2\right).$$

Problem 2: Maximum Subarray Sum (Divide and Conquer)

Detailed Algorithm

Algorithm 1 `MaxSubarray`(A,ℓ,r)

if $\ell = r$ **then**
  return $\big(\ell,\ r,\ A[\ell]\big)$
end if
$m \leftarrow \lfloor(\ell + r)/2\rfloor$
$(\ell_L, r_L, S_L) \leftarrow$ `MaxSubarray`(A, ℓ, m)
$(\ell_R, r_R, S_R) \leftarrow$ `MaxSubarray`$(A, m+1, r)$
maxLeft $\leftarrow -\infty$, temp $\leftarrow 0$, start $\leftarrow m$
for $i = m$ **to** ℓ **step** -1 **do**
  temp $\leftarrow$ temp $+ A[i]$
  if temp $>$ maxLeft **then**
    maxLeft $\leftarrow$ temp, start $\leftarrow i$
  end if
end for
maxRight $\leftarrow -\infty$, temp $\leftarrow 0$, end $\leftarrow m+1$
for $j = m+1$ **to** r **do**
  temp $\leftarrow$ temp $+ A[j]$
  if temp $>$ maxRight **then**
    maxRight $\leftarrow$ temp, end $\leftarrow j$
  end if
end for
$S_C \leftarrow$ maxLeft + maxRight
$(\ell_C, r_C) \leftarrow$ (start, end)
if $S_L \geq S_R$ and $S_L \geq S_C$ **then**
  return (ℓ_L, r_L, S_L)
else if $S_R \geq S_C$ **then**
  return (ℓ_R, r_R, S_R)
else
  return (ℓ_C, r_C, S_C)
end if

Time Complexity

Let $T(n)$ be the running time. We have

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$$

so by the Master Theorem $T(n) = \Theta(n \log n)$.

Problem 3: Inversion Counting

Given a sequence $S = [a_1, a_2, \ldots, a_n]$ of n distinct numbers, an inversion is a pair (a_i, a_j) such that $i < j$ and $a_i > a_j$. The goal is to count the total number of inversions in S.

Algorithm Design

We use a divide and conquer approach based on merge sort:

1. **Divide:** Split the sequence S into two halves.
2. **Conquer:** Recursively count the inversions in the left half and the right half.
3. **Combine:** Count the inversions during the merge step. When merging the two sorted halves, if an element in the right half is placed before an element in the left half, then all remaining elements in the left half form inversions with this element.
4. **Return:** The total inversion count is the sum of inversions from the left half, right half, and the merging step.

Algorithm 2 `CountInversions`(S, ℓ, r)

if $\ell = r$ **then**
  return $(0, [S[\ell]])$ {Zero inversions; single element}
end if
$m \leftarrow \lfloor (\ell + r)/2 \rfloor$ {Midpoint}
$(\text{inv}_L,\ L) \leftarrow$ `CountInversions`(S, ℓ, m)
$(\text{inv}_R,\ R) \leftarrow$ `CountInversions`$(S, m+1, r)$
$i \leftarrow 1,\ j \leftarrow 1,\ \text{inv}_{\text{split}} \leftarrow 0$
for $k = \ell$ **to** r **do**
  if $i \leq |L|$ **and** $\big(j > |R|$ OR $L[i] \leq R[j]\big)$ **then**
    $S[k] \leftarrow L[i];\ i \leftarrow i + 1$
  else
    $S[k] \leftarrow R[j];\ j \leftarrow j + 1$
    $\text{inv}_{\text{split}} \leftarrow \text{inv}_{\text{split}} + (|L| - i + 1)$ {All remaining in L form inversions}
  end if
end for
$\text{inv}_{\text{total}} \leftarrow \text{inv}_L + \text{inv}_R + \text{inv}_{\text{split}}$
return $(\text{inv}_{\text{total}},\ S[\ell..r])$

Time Complexity Analysis

Let $T(n)$ denote the time complexity. The recurrence is:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$$

By the Master Theorem (with $a = 2$, $b = 2$, and $f(n) = n$), we obtain:

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

Thus, the algorithm achieves a time complexity better than $O(n^2)$.

Problem 4: Strassen's Matrix Multiplication

Strassen's algorithm is a divide and conquer method for matrix multiplication that reduces the number of recursive multiplications.

Example: Compute the Matrix Product

Let

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}.$$

Strassen's algorithm computes 7 products as follows:

$$\begin{aligned}
M_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) = (1+4)(3+2) = 5 \cdot 5 = 25, \\
M_2 &= (a_{21} + a_{22})b_{11} = (3+4) \cdot 3 = 7 \cdot 3 = 21, \\
M_3 &= a_{11}(b_{12} - b_{22}) = 1 \cdot (4-2) = 1 \cdot 2 = 2, \\
M_4 &= a_{22}(b_{21} - b_{11}) = 4 \cdot (1-3) = 4 \cdot (-2) = -8, \\
M_5 &= (a_{11} + a_{12})b_{22} = (1+2) \cdot 2 = 3 \cdot 2 = 6, \\
M_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) = (3-1)(3+4) = 2 \cdot 7 = 14, \\
M_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) = (2-4)(1+2) = (-2) \cdot 3 = -6.
\end{aligned}$$

Then, the resulting matrix $C = AB$ is computed as:

$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 = 25 - 8 - 6 - 6 = 5, \\
C_{12} &= M_3 + M_5 = 2 + 6 = 8, \\
C_{21} &= M_2 + M_4 = 21 - 8 = 13, \\
C_{22} &= M_1 - M_2 + M_3 + M_6 = 25 - 21 + 2 + 14 = 20.
\end{aligned}$$

Thus, the final result is:

$$AB = \begin{bmatrix} 5 & 8 \\ 13 & 20 \end{bmatrix}.$$

Time Complexity Explanation

For multiplying two $n \times n$ matrices using Strassen's algorithm, the recurrence is:

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2).$$

Here, $a = 7$ and $b = 2$, so using the Master Theorem we get:

$$T(n) = \Theta\left(n^{\log_2 7}\right),$$

where $\log_2 7 \approx 2.81$. Hence, the time complexity is:

$$O\left(n^{\log_2 7}\right),$$