Divide and Conquer

Madiba Hudson-Quansah

Contents

Chapter 1

Introduction

Divide and Conquer is characterized by three steps:

1. Divide the problem into several sub problems of the same type, ideally of equal size
2. Solve the sub problems, typically recursively.
3. If necessary, combine the solutions of all the sub problems to get the solution to the original problem.

This set of operations often result in running time that is in the form of the recurrence relation:

$$T(n) = aT(n/b) + f(n)$$

Where:

a is the number of sub problems.

b is the number of instances the problem is divided into

$f(n)$ is the time spent on dividing an instance of size n into instances of size n/b and combining their solutions.

This is called the **general divide and conquer recurrence**, and its asymptotic time can be found using the **Master Theorem**, where if $f(n) \in \Theta\left(n^d\right)$ where $d \geqslant 0$:

$$T(n) \in \begin{cases} \Theta\left(n^d\right) & \text{if } a < b^d \\ \Theta\left(n^d \log n\right) & \text{if } a = b^d \\ \Theta\left(n^{\log_b a}\right) & \text{if } a > b^d \end{cases}$$

Chapter 2

Merge Sort

Definition 2.0.1: Merge Sort

Divide a given array $A[0 \ldots n-1]$ into two halves, $A[0 \ldots \lfloor n/2 \rfloor - 1]$ and $A[\lfloor n/2 \rfloor \ldots n-1]$, sorting each of them recursively, and then merging the two smaller sorted arrays into a single sorted array.

Algorithm 1 MergeSort ($A[0 \ldots n-1]$)

▷

▷ Sorts array $A[0 \ldots n-1]$ by recursive merge sort
▷ Input: An array $A[0 \ldots n-1]$ of orderable elements
▷ Output: Array $A[0 \ldots n-1]$ sorted in nondecreasing order

1: **if** $n > 1$ **then**
2: copy $A[0 \ldots \lfloor n/2 \rfloor - 1]$ to $B[0 \ldots \lfloor n/2 \rfloor - 1]$
3: copy $A[\lfloor n/2 \rfloor \ldots n-1]$ to $C[0 \ldots \lfloor n/2 \rfloor - 1]$
4: MergeSort(B)
5: MergeSort(C)
6: Merge(B, C, A)
7: **end if**

Algorithm 2 Merge $\left(B\left[0\ldots p-1\right], C\left[0\ldots q-1\right], A\left[0\ldots p+q-1\right]\right)$

▷

▷ Merges two sorted arrays B and C into A
▷ Input: Arrays B and C both sorted
▷ Output: Sorted array A of the elements of B and C

```
1:  i ← 0; j ← 0; k ← 0
2:  while i < p and j < q do
3:      if B_i ⩽ C_j then
4:          A_k ← B_i
5:          i ← i + 1
6:      else
7:          A_k ← C_j
8:          j ← j + 1
9:      end if
10:     k ← k + 1
11: end while
12: if i = p then
13:     copy C[j ... q − 1] to A[k ... p + q − 1]
14: else
15:     copy B[i ... p − 1] to A[k ... p + q − 1]
16: end if
```

The cost of the merge sort algorithm can is also in the form of the general divide and conquer recurrence:

$$T(n) = 2T(n/2) + T_{\text{merge}}(n) \text{ for } n > 1,\ T(1) = 0$$

Using master theorem the worst case time complexity is:

$$O\left(n^1 \log n\right)$$
$$O(n \log n)$$

And the space complexity due to the recursive calls and copying the items of the array is $O(\log n)$.

Chapter 3

Quick Sort

Definition 3.0.1: Quick Sort

Like merge sort, quick sort divides the array into two sub arrays but unlike merge sort quicksort divides the elements according to their value based on a pivot element (Partitioning). This means that quick sort does not require additional space and also sorts the array as it divides it.

Algorithm 3 QuickSort ($A[l \dots r]$)

▷

▷ Sorts a subarray by quicksort
▷ Input: Subarray of array $A[0 \dots n-1]$, defined by its left and right indices l and r
▷ Output: Subarray $A[l \dots r]$ sorted in nondecreasing order

1: **if** $l < r$ **then**
2: $s \leftarrow$ Partition($A[l \dots r]$) ▷ s is a split position
3: QuickSort($A[l \dots s-1]$)
4: QuickSort($A[s+1 \dots r]$)
5: **end if**

The partitioning algorithm can be Lomuto's partitioning discussed in 3_Decrease_Conquer or Hoare's partitioning algorithm.

In the best case where the pivot is the median of the array, i.e. perfectly splits the array into two equal halves, the recurrence relation is:

$$T(n) = 2T(n/2) + O(n)$$

Where $O(n)$ is the time spent finding the position of the pivot. Using the master theorem, the best case time complexity is:

$$O(n \log n)$$

In the worst case where the pivot is the smallest or largest element in the array, the recurrence relation becomes:

$$T(n) = 2T(n-1) + O(n)$$

Solving this recurrence relation using the iterative method, where $T(1) = 0$:

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

$$
\begin{aligned}
T(n) &= 2(2T(n-2) + n) + n \\
&= 4T(n-2) + 3n \\
&= 4(2T(n-3) + n) + n \\
&= 8T(n-3) + 5n \\
T(n) &= 2^k T(n-k) + (2k-1)n
\end{aligned}
$$

$$
\begin{aligned}
n - k &= 1 \\
n &= k + 1 \\
k &= n - 1 \\
T(k+1) &= 2^k T(1) + (2k-1)(k+1) \\
&= 2k^2 + 2k - k + 1 \\
&= 2k^2 + k + 1 \\
&= 2(n-1)^2 + n - 1 + 1 \\
&= 2(n-1)(n-1) + n \\
&= 2n^2 - n - n + 1 + n \\
&= 2n^2 - n + 1 \\
&\therefore O\left(n^2\right)
\end{aligned}
$$

Chapter 4

Binary Tree Traversals

The divide and conquer strategy can be used in relation to binary trees in many ways.

4.1 Height

Definition 4.1.1: Height of a Binary Tree

The height of a binary tree is the length of the longest path from the root to a leaf. It is also convenient to define the height of an empty tree as -1.

Algorithm 4 Height (T)

▷ Finds the height of a binary tree T recursively
▷ Input: A binary tree T
▷ Output: The height of T

1: **if** T is empty **then**
2: **return** -1
3: **end if**
4: **return** $\max\left\{\textsc{Height}(T_{\text{left}}), \textsc{Height}(T_{\text{right}})\right\} + 1$

The size of the problem instance is the number of nodes in the tree, n. Where the number of comparisons made to compute the maximum of two numbers and the number of additions $A(n)$ made by the algorithm are the same, giving us the recurrence relation:

$$A(n) = A(n_{\text{left}}) + A\left(n_{\text{right}}\right) + 1 \text{ for } n > 0$$
$$A(0) = 0$$

Solving this recurrence where:

Chapter 5

Multiplication of Large Integers and Strassen's Matrix Multiplication

5.1 Multiplication of Large Integers

The multiplication of large integers can be done using the divide and conquer strategy which reduces the number of total digit multiplications from n^2. To demonstrate given the numbers 23 and 14. These numbers can be represented as:

$$23 = 2 \cdot 10^1 + 3 \cdot 10^0 \text{ and } 14 = 1 \cdot 10^1 + 4 \cdot 10^0$$

Multiplying them in the usual way:

$$\begin{aligned} 23 \times 14 &= \left(2 \cdot 10^1 + 3 \cdot 10^0\right) \times \left(1 \cdot 10^1 + 4 \cdot 10^0\right) \\ &= (2 \times 1)\, 10^2 + (2 \times 4 + 3 \times 1)\, 10^1 + (3 \times 4)\, 10^0 \end{aligned}$$

Continuing with the last formula would use the same 4 multiplications as the usual method. However, we can compute the middle term with just one digit multiplication by taking advantage of the fact that the products 2×1 and 3×4 need to be computed anyway. We then express the middle term in a form that has both these products:

$$2 \times 4 + 3 \times 1 = (2 + 3) \times (1 + 4) - 2 \times 1 - 3 \times 4$$

This can be generalized for any pair of two-digit numbers $a = a_1 a_0$ and $b = b_1 b_0$, where c is their product.

$$\begin{aligned} c &= a \times b \\ &= c_2 10^2 + c_1 10^1 + c_0 \end{aligned}$$

Where:

$c_2 = a_1 \times b_1$ is the product of their first digits

$c0 = a_0 \times b_0$ is the product of their second digits

$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$ is the product of the sum of the a's digits and the sum of the b's digits minus the sum of c_2 and c_0

Further generalizing this trick to n-digit numbers a and b, where n is a positive even number, we first divide both numbers into two halves of $n/2$ digits each, denoting the first half of a's digits as a_1 and the second half as a_0, and similarly for b. This then implies that:

$$\begin{aligned} a &= a_1 a_0 = a_1 10^{n/2} + a_0 \\ b &= b_1 b_0 = b_1 10^{n/2} + b_0 \end{aligned}$$

This then implies that:

$$\begin{aligned} c = a \times b &= \left(a_1 10^{n/2} + a_0\right) \times \left(b_1 10^{n/2} + b_0\right) \\ &= (a_1 \times b_1)\, 10^n + (a_1 \times b_0 + a_0 \times b_1)\, 10^{n/2} + (a_0 \times b_0) \\ &= c_2 10^n + c_1 10^{n/2} + c_0 \end{aligned}$$

Where:

$c_2 = a_1 \times b_1$ is the product of the first halves of the numbers

$c_0 = a_0 \times b_0$ is the product of the second halves of the numbers

$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$ is the product of the sum of the a's halves and the sum of the b's halves minus the sum of c_2 and c_0

If $n/2$ is even the same method for computing the values of c_2, c_1 and c_0 can be used. Thus is n is a power of 2 this algorithm can be run recursively until the base case of $n = 1$ is reached or when n is deemed small enough to be computed using the usual method. This them results in the number of multiplication taking the form the recurrence relation:

$$M(n) = 3M(n/2) \text{ for } n > 1,\ M(1) = 1$$

Which using the master theorem gives the time complexity of:

$$O\left(n^{\log_2 3}\right) \approx O\left(n^{1.585}\right)$$

5.2 Strassen's Matrix Multiplication

Strassen's matrix multiplication is a divide and conquer algorithm that reduces the number of multiplications needed to multiply two matrices. Given two matrices A and B of size $n \times n$, the product $C = A \times B$ can be computed as:

$$\begin{aligned} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} &= \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} \\ &= \begin{bmatrix} m_1 + m_4 - m_5 + +m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix} \end{aligned}$$

Where:

$$\begin{aligned} m_1 &= (a_{00} + a_{11}) \times (b_{00} + b_{11}) \\ m_2 &= (a_{10} + a_{11}) \times b_{00} \\ m_3 &= a_{00} \times (b_{01} - b_{11}) \\ m_4 &= a_{11} \times (b_{10} - b_{00}) \\ m_5 &= (a_{00} + a_{01}) \times b_{11} \\ m_6 &= (a_{10} - a_{00}) \times (b_{00} + b_{01}) \\ m_7 &= (a_{01} - a_{11}) \times (b_{10} + b_{11}) \end{aligned}$$

This can then be generalized for any $n \times n$ matrix where n is a power of 2, i.e.:

$$\begin{aligned} \left[\begin{array}{c|c} C_{00} & C_{01} \\ \hline C_{10} & C_{11} \end{array}\right] &= \left[\begin{array}{c|c} A_{00} & A_{01} \\ \hline A_{10} & A_{11} \end{array}\right] \times \left[\begin{array}{c|c} B_{00} & B_{01} \\ \hline B_{10} & B_{11} \end{array}\right] \\ &= \begin{bmatrix} M_1 + M_4 - M_5 + +M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix} \end{aligned}$$

Where the formulae are the same but in terms of the sub-matrices of A and B. This results in the following recurrence relation, where M is the number of multiplications:

$$M(n) = 7M(n/2) \text{ for } n > 1,\ M(1) = 1$$

Using the master theorem the time complexity of Strassen's matrix multiplication is:

$$O\left(n^{\log_2 7}\right) \approx O\left(n^{2.807}\right)$$

Chapter 6

The Closest-Pair and Convex-Hull Problems