Keynotes and Preliminary Knowledge from Algorithm Design and Applications

Chapter 8.1: Merge-Sort

Merge-Sort Overview:

- Divide-and-Conquer Paradigm:
  1. Divide: Split the input data into two disjoint subsets.
  2. Recur: Recursively solve the subproblems.
  3. Conquer: Merge the solutions of the subproblems to solve the original problem.

Steps in Merge-Sort:

1. Divide: If the sequence has one or zero elements, it's already sorted. If not, split it into two sequences S1 and S2.
2. Recur: Recursively sort S1 and S2.
3. Conquer: Merge S1 and S2 into a single sorted sequence.

Merge-Sort Tree:

- Each node represents a recursive call.
- The height of the tree is log n, where n is the number of elements.
- The merge operation takes O(n) time.

Analysis:

- The merge step involves sequentially accessing data, which is efficient for large datasets.
- Merge-sort has a time complexity of O(n log n).

Pseudocode for Merge:

Algorithm merge(S1, S2, S):

Input: Two arrays, S1 and S2, of size n1 and n2, respectively, sorted in non-decreasing order, and an empty array, S, of size at least n1 + n2

Output: S, containing the elements from S1 and S2 in sorted order

```
i ← 1
j ← 1
while i ≤ n and j ≤ n do
    if S1[i] ≤ S2[j] then
        S[i + j − 1] ← S1[i]
        i ← i + 1
    else
        S[i + j − 1] ← S2[j]
        j ← j + 1
while i ≤ n do
    S[i + j − 1] ← S1[i]
    i ← i + 1
while j ≤ n do
    S[i + j − 1] ← S2[j]
    j ← j + 1
```

Chapter 8.2: Quick-Sort

Quick-Sort Overview:

- Steps in Quick-Sort:
  1. Divide: Choose a pivot and partition the sequence into L, E, and G.
  2. Recur: Recursively sort L and G.

3. Conquer: Concatenate L, E, and G back into S.

In-Place Quick-Sort:

- Utilizes the input array for storing subarrays.
- Partitions the array using a pivot, and recursively sorts subarrays.

Visualization:

- Uses a binary recursion tree where each node represents a recursive call and its pivot.

Chapter 11.1: Recurrences and the Master Theorem

Master Theorem for Divide-and-Conquer Recurrences:

- Used to solve recurrence relations of the form $T(n) = aT(n/b) + f(n)$.
- Cases:
  1. If $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$.
  2. If $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$.
  3. If $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$ and $af(n/b) \leq \delta f(n)$, then $T(n)$ is $\Theta(f(n))$.

Examples:

- $T(n) = 4T(n/2) + n$ resolves to $\Theta(n^2)$.
- $T(n) = 2T(n/2) + n \log n$ resolves to $\Theta(n \log^2 n)$.

Chapter 11.2: Integer Multiplication

Problem:

- Multiplying large integers efficiently, which has applications in data security.

Divide-and-Conquer Algorithm:

- Split integers into halves and recursively compute products.

- Time complexity can be reduced to $O(n^{1.585})$ using the divide-and-conquer approach.

Theorem:

- Multiplying two n-bit integers can be done in $O(n^{\log_2 3})$.

Chapter 11.3: Matrix Multiplication

Problem:

- Efficiently multiplying two n x n matrices.

Strassen’s Algorithm:

- Uses 7 recursive multiplications instead of 8.

- Running time: $O(n^{2.808})$.

Further Improvements:

- More advanced algorithms can achieve running times as low as $O(n^{2.376})$.

Chapter 11.4: The Maxima-Set Problem

Problem:

- Finding a set of maximal points from a set of points in a plane.

Divide-and-Conquer Algorithm:

- Sort points lexicographically.

- Recursively find maxima sets on the left and right.

- Merge these sets by removing dominated points.

Time Complexity:

- O(n log n).