Advanced Sorting Algorithms

Reynard Gunawan

Thursday 31st August, 2023

Contents

1 Merge Sort **3**

1.1 Time Complexity 3

1.2 Space Complexity 3

2 Quick Sort **3**

2.1 Time Complexity 3

2.2 Space Complexity 3

3 Heap Sort **3**

3.1 Time Complexity 4

3.2 Space Complexity 4

1 Merge Sort

The Merge Sort algorithm is a sorting algorithm that follows a divide-and-conquer strategy. The algorithm works by recursively breaking down the input array into smaller sub-arrays until each sub-array contains a single element. The merging step involves comparing the elements of the sub-array and merging them together in a manner that maintains order. The merging process continues until the entire array is reconstructed in its sorted form.

1.1 Time Complexity

One of the advantages of Merge Sort lies in its consistent time complexity of $O(n \log n)$, which is suitable for large datasets. The stability of Merge Sort ensures that the relative order of "equal" elements remains unchanged after sorting.

1.2 Space Complexity

In addition to Merge Sort's stability, it's space complexity of $O(n)$ requires an amount of memory space proportional to the size of the input array, which may be a consideration when working with constrained memory resources.

2 Quick Sort

The Quick Sort algorithm also follows a divide-and-conquer approach, but it distinguishes itself from Merge Sort through its pivot-based partioning strategy. The algorithm begins by selecting a pivot element from the input array. The goal of the partitioning phase is to rearrange the array elements so that all elements to the left of the pivot are less than the pivot and all the elements to the right are greater than the selected pivot element. The algorithm recursively applies the same partitioning process to the sub-arrays eventually resulting in a fully sorted array.

2.1 Time Complexity

Quick Sort's efficiency is primarily from its average case time complexity of $O(n \log n)$, which makes it once of the fastest sorting algorithm. The performance of Quick Sort is heavily reliant on the choice of pivot. So assuming the worst-case scenario would result in a time complexity about $O(n^2)$.

2.2 Space Complexity

Alongside with Quick Sort's speed would be its efficient space complexity of $O(\log n)$, making it an excellent choice for situations when space is restricted.

3 Heap Sort

The HeapSort algorithm is a comparison-based sorting algorithm that employs the concept of a binary heap data structure to achieve efficient sorting. The algorithm consists of two main phases: building a max-heap and the repeatedly extracting the maximum element from the heap.

The first phase, heap construction, involves converting the input array into a binary tree where the value of each node is greater than or equal to the values of its child nodes. The algorithm goes down each element to its correct position within the heap maintaining the max-heap property.

The second phase involves repeatedly extracting the maximum element from the bottom of the max-heap and swapping it with the last element of the heap. The heap size is then decreased by one repeatedly until the entire array is sorted.

3.1 Time Complexity

HeapSort's main advantage is its ability to achieve a consistent worst-case time complexity, which can be advantageous in scenarios where the dataset is large and the time performance needs to be predictable. In this case, the time complexity would be $O(n \log n)$.

3.2 Space Complexity

Due to HeapSort's nature of being an in-place sorting algorithm, it does not use any extra data structure. Therefore, its space complexity is $O(1)$, meaning the amount of memory used is constant and does not depend on the data that it is processing.