[CS203] Design and Analysis of Algorithms

Course Instructor: Dr. Dibyendu Roy
Scribed by: Deepanshu (202351025)

Autumn 2024-25
Lecture (Week 1)

1 Insertion Sort

Introduction Insertion Sort is one of the simplest sorting algorithms, often used in teaching basic algorithmic concepts. It works well on small datasets or nearly sorted data and is known for its straightforward implementation.

Algorithm Insertion Sort builds the final sorted array one item at a time. It iterates through an input array and removes one element per iteration, finds the place the element belongs in the sorted array, and inserts it there.

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

Time Complexity

- Worst-case: $O(n^2)$
- Best-case: $O(n)$
- Average-case: $O(n^2)$

Space Complexity $O(1)$ - In-place sorting algorithm

Characteristics

- Simple implementation
- Efficient for small data sets
- Adaptive and stable
- In-place algorithm
- Inefficient for large data sets

Conclusion Insertion Sort, despite its simplicity, is highly effective for small or nearly sorted datasets. Its in-place operation and stability make it a practical choice for scenarios where memory usage is a concern, though it struggles with larger datasets due to its quadratic time complexity.

2 Merge Sort

Introduction Merge Sort is a classic example of a divide-and-conquer algorithm. It offers a consistent performance of $O(n \log n)$ across all cases, making it a reliable choice for sorting large datasets. Unlike simpler algorithms, Merge Sort requires additional space for merging.

Algorithm Merge Sort is a divide-and-conquer algorithm that divides the input array into two halves, recursively sorts them, and then merges the two sorted halves.

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

Time Complexity $O(n \log n)$ for all cases (worst, average, and best)

Space Complexity $O(n)$ - Requires additional space for merging

Characteristics

- Consistent performance
- Stable sorting algorithm
- Suitable for large datasets
- Can be parallelized

- Requires additional space
- Slower for small datasets compared to simpler algorithms

Conclusion Merge Sort is a powerful sorting algorithm that shines with large datasets and guarantees stable sorting with a consistent time complexity of $O(n \log n)$. However, its additional space requirement and relatively slower performance on small datasets make it less suitable for memory-constrained environments or when dealing with smaller arrays.

3 Comparison

- Insertion Sort is simpler and performs better for small or nearly sorted datasets.
- Merge Sort is more efficient for large datasets and has consistent performance.
- Insertion Sort is in-place, while Merge Sort requires additional space.
- Both algorithms are stable, preserving the relative order of equal elements.