Sec project | Rand Hussein & Asia Assi |10-12-25 | Algorithms | 202310684 & 202310445

Summary of Results

The performance analysis of sorting algorithms shows that Insertion Sort works well only on small or nearly sorted data, while it becomes impractical with large, reverse, or random data, where the time can reach 1109 milliseconds for a sample of 100,000 elements, but it is the fastest on sorted or few-unique data (0–1 milliseconds). In contrast, the Merge Sort algorithm shows stable performance with O(n log n) time on all types of data, unaffected by their order, and is always among the fastest with times ranging from 3–14 milliseconds for a 100,000-element sample, while Heap Sort maintains the same behavior with O(n log n) time, is stable on all types of data, but slightly slower than Merge Sort with performance times ranging from 5–14 milliseconds for the same sample. As for deterministic QuickSort, it performs well on random data (8–10 milliseconds for a 100,000-element sample), but it becomes worst-case $O(n^2)$ with sorted, reverse, or few-unique data, which may cause a StackOverflowError, so it was skipped in these cases. On the other hand, Randomized QuickSort ensures excellent stability and very high speed, as it avoids worst-case scenarios thanks to random pivot selection and achieves 0–1 milliseconds even for very large data, making it the fastest and safest algorithm for all data types. The overall conclusion indicates that Merge Sort and Heap Sort are the most stable and consistently performing algorithms, while deterministic QuickSort is prone to failure on some data, Randomized QuickSort is the preferred version for practical applications, and Insertion Sort becomes impractical except for small or nearly sorted data. The time complexity of these algorithms shows that Insertion Sort ranges from O(n) in the best case to $O(n^2)$ in the worst case, Merge Sort and Heap Sort are consistently O(n log n), and QuickSort ranges between O(n log n) in the average and best cases and $O(n^2)$ in the worst case. The performance differences are explained by the fact that Randomized QuickSort and Merge Sort provide the best overall performance, Heap Sort is a reliable and stable alternative, deterministic QuickSort may fail due to poor partitions and deep recursion, while Insertion Sort is suitable only for small or nearly sorted data.

Random data

```
Size = 1000
Insertion Sort: 1 ms
Merge Sort: 2 ms
Heap Sort: 1 ms
Quick Sort: 1 ms
RandomizedQuick Sort: 0 ms

Size = 10000
Insertion Sort: 18 ms
Merge Sort: 4 ms
Heap Sort: 3 ms
Quick Sort: 1 ms
RandomizedQuick Sort: 0 ms

Size = 50000
Insertion Sort: 452 ms
Merge Sort: 7 ms
Heap Sort: 7 ms
Quick Sort: 9 ms
RandomizedQuick Sort: 0 ms

Size = 100000
Insertion Sort: 548 ms
Merge Sort: 14 ms
Heap Sort: 14 ms
Quick Sort: 10 ms
RandomizedQuick Sort: 0 ms
```

FewUnique data

```
Size = 1000
Insertion Sort: 0 ms
Merge Sort: 0 ms
Heap Sort: 0 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 10000
Insertion Sort: 4 ms
Merge Sort: 4 ms
Heap Sort: 0 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 50000
Insertion Sort: 102 ms
Merge Sort: 7 ms
Heap Sort: 3 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 100000
Insertion Sort: 462 ms
Merge Sort: 6 ms
Heap Sort: 6 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms
```

Sorted data

```
Size = 1000
Insertion Sort: 0 ms
Merge Sort: 0 ms
Heap Sort: 0 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 10000
Insertion Sort: 0 ms
Merge Sort: 0 ms
Heap Sort: 1 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 50000
Insertion Sort: 0 ms
Merge Sort: 2 ms
Heap Sort: 5 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 100000
Insertion Sort: 0 ms
Merge Sort: 10 ms
Heap Sort: 7 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms
```

Reverse data

```
Size = 1000
Insertion Sort: 1 ms
Merge Sort: 0 ms
Heap Sort: 1 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 10000
Insertion Sort: 12 ms
Merge Sort: 1 ms
Heap Sort: 0 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 50000
Insertion Sort: 278 ms
Merge Sort: 2 ms
Heap Sort: 3 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

Size = 100000
Insertion Sort: 1109 ms
Merge Sort: 3 ms
Heap Sort: 8 ms
Quick Sort: skipped (worst-case input) causes StackOverFlow error
RandomizedQuick Sort: 0 ms

 *** finish ***
```