Report

Method	Input size:1000 (in millisecond)	Input size:31K (in millisecond)	Input size:1M (in millisecond)
Standard Sort	0.14 ms	6.45 ms	278.433 ms
Half selection sort	3.388 ms	3368.824 ms	17126.942 ms
Merge Sort	0.453 ms	16.82 ms	623.866 ms
InPlace Merge Sort	0.302 ms	12.68 ms	476.145 ms
Half heap Sort	0.105 ms	4.623 ms	189.071 ms
QuickSelect	0.053 ms	1.603 ms	79.918 ms

Worst Case QuickSelect: 0.18274 milliseconds (20000 inputs)
Result: 10000

Algorithm Analysis:

Standard Sort: A sorting method that uses the Introsort algorithm which combines QuickSort,Heap Sort and InsertionSort and has the complexity of O(N log(N)), with an average-case time complexity of O(n log n). The timed results are consistent with expectations, demonstrating a logarithmic increase in time with larger input sizes.

Half Selection Sort: This algorithm's time complexity is O(n^2/2). Its performance is significantly worse than expected, especially with larger input sizes, showing a drastic increase in time due to its quadratic time complexity. This method doesn't fully sort the array, instead, it sorts only the first half of the array while finding the median. In the worst case, it performs half the number of comparisons compared to a full selection sort.

Merge Sort: Known for its O(n log n) time complexity, the results align well with this expectation. The timed results demonstrate a logarithmic growth rate with input size increment.

InPlace Merge Sort: A variation of Merge Sort with a similar time complexity of O(n log n). Its performance closely matches that of the standard Merge Sort across various input sizes. Based on the results presented, for all input sizes like 1000, 31K, and 1M, the InPlace Merge Sort seems to perform better than the regular Merge Sort method.

Half Heap Sort: With an expected time complexity of O(n log n), this method's performance matches expectations, demonstrating a logarithmic increase in time as input size grows. By only having to remove n/2 elements from the heap rather than n, cut down on the time significantly.

QuickSelect: Shows very low times even with larger input sizes. With an average-case time complexity of O(n), QuickSelect provides efficient selection. The results confirm its efficiency, showing very low times even with larger input sizes.

In QuickSelect, once the pivot is placed in its correct position the algorithm recurses on only one side of the pivot. This approach significantly reduces the number of comparisons needed compared to traditional sorting algorithms.

Compared to the QuickSort algorithm,QuickSelect excels in selecting specific elements in linear time on average but might be less consistent in worst case scenarios.