Fateha Zaman

1) A brief description of the algorithms and their theoretical complexity
   a) Bubble sort- An algorithm that traverses the array *n* times, since each traversal is at maximum *n* items long.
      i) Best case: Ω (*n*)
      ii) Average case: Θ(n^2)
      iii) Worst case: O(n^2)
   b) Optimized Bubble sort- An updated version of Bubble sort that stops when the inner loop doesn't swap any value.
      i) Best case: Ω (*n*)
      ii) Average case: Θ(n^2)
      iii) Worst case: O(n^2)
   c) Insertion Sort- An algorithm that gets the final sorted array by adding one item at a time. It divides the array in two parts, the first part is sorted. Initially, the sorted part contains the first item of the array. The algorithm inserts one item at a time from the unsorted part of the array into the sorted part.
      i) Best case: Ω (n)
      ii) Average case: Θ(n^2)
      iii) Worst case: O(n^2)
   d) Selection Sort- An algorithm that gets the smallest item from an unsorted array in each iteration, and places this item at the beginning of the unsorted array.
      i) Best case: Ω(*n*^2)
      ii) Average case: Θ(n^2)
      iii) Worst case: O(n^2)
   e) Merge sort- Merge sort divides an unsorted array into *n* subarrays until subarrays contain one item. Repeatedly, it merges the sorted subarrays by interleaving their items to get a new sorted array
      i) Best case Ω (n log n)
      ii) Average case: Θ(n log n)
      iii) Worst case: O(n log n)
   f) Heap Sort- An algorithm that transforms the input array into a max heap by using the function heapify. Then, it repeatedly swaps the value at the root node with the last element in the heap and restores the max heap property until the array is sorted. The last element from the heap is excluded on each iteration.
      i) Best case Ω (n log n)
      ii) Average case: Θ(n log n)
      iii) Worst case: O(n log n)

g) Quicksort using Lomuto's partition- It selects a pivot item from the array and divides the rest of the items into two subarrays, according to whether they are less than or greater than the pivot. The subarrays are then sorted recursively.
   i) Best case Ω (n log n)
   ii) Average case: Θ(n log n)
   iii) Worst case: O(n^2)

h) Quicksort using Hoare's partition- It selects a pivot item from the array and divides the rest of the items into two subarrays, according to whether they are less than or greater than the pivot. The subarrays are then sorted recursively.
   i) Best case Ω (n log n)
   ii) Average case: Θ(n log n)
   iii) Worst case: O(n^2)

2) Total elementary operations executed by each algorithm. Analyze the experimental results and compare them with the theoretical complexity of the algorithms.
   a) Based on the theoretical complexity of the algorithm, both bubble sort, selection sort, and insertion sort algorithms have the longest average and worst cases time complexity. However, only in the best scenario, does insertion sort and the bubble sort algorithms run the fastest.
   b) Based on the experimental results, that was proven to be true.The bubble sort algorithms took the longest in every input tested. Right above bubble sort and optimized bubble sort came selection sort in almost every scenario except for the input size of 1024.
   c) According to the theoretical complexity of the algorithm, overall as a sorting algorithm, heap sort and merge sort are the fastest with a time complexity of O(n log n) in every case. Quick sort has a similar time complexity to heap sort and merge sort with a running time of O(n log n) for the best and average case. However, for the worst case, quick sort's running time is O(n^2). Therefore, based on average cases, the fastest performance will be between heap sort, merge sort, and quick sort. Based on the worst case, the fastest performance will be between heap sort and merge sort.
   d) After analyzing the results, one of the quick sort algorithms came first every time, with the shortest time. Then it was between merge and heap sort, depending on the input size.

Performance on Random Input - Size 1024

BubbleSort

Time elapsed: 16495032 nanoseconds

Elementary Instructions: 2375363

OptimizedBubbleSort

Time elapsed: 9943562 nanoseconds

Elementary Instructions: 1587879

InsertionSort

Time elapsed: 5632297 nanoseconds

Elementary Instructions: 1066588

SelectionSort

Time elapsed: 5520651 nanoseconds

Elementary Instructions: 529836

HeapSort

Time elapsed: 2084846 nanoseconds

Elementary Instructions: 70538

HoareQuicksort

Time elapsed: 2580025 nanoseconds

Elementary Instructions: 33877

MergeSort

Time elapsed: 1301022 nanoseconds

Elementary Instructions: 30720

LomutoQuickSort

Time elapsed: 1299968 nanoseconds

Elementary Instructions: 30578

Performance on Random Input - Size 2048

BubbleSort

Time elapsed: 9358948 nanoseconds

Elementary Instructions: 9479958

OptimizedBubbleSort

Time elapsed: 8474948 nanoseconds

Elementary Instructions: 6340045

InsertionSort

Time elapsed: 3747595 nanoseconds

Elementary Instructions: 4244512

SelectionSort

Time elapsed: 5659060 nanoseconds

Elementary Instructions: 2108230

HeapSort

Time elapsed: 959950 nanoseconds

Elementary Instructions: 157631

HoareQuicksort

Time elapsed: 479556 nanoseconds

Elementary Instructions: 76470

MergeSort

Time elapsed: 729723 nanoseconds

Elementary Instructions: 67584

LomutoQuickSort

Time elapsed: 598086 nanoseconds

Elementary Instructions: 69105

Performance on Random Input - Size 4096

BubbleSort

Time elapsed: 24671942 nanoseconds

Elementary Instructions: 37464944

OptimizedBubbleSort

Time elapsed: 24914644 nanoseconds

Elementary Instructions: 24768081

InsertionSort

Time elapsed: 8565596 nanoseconds

Elementary Instructions: 16385176

SelectionSort

Time elapsed: 6499200 nanoseconds

Elementary Instructions: 8410818

HeapSort

Time elapsed: 786094 nanoseconds

Elementary Instructions: 348990

HoareQuicksort

Time elapsed: 340137 nanoseconds

Elementary Instructions: 168305

MergeSort

Time elapsed: 790655 nanoseconds

Elementary Instructions: 147456

LomutoQuickSort

Time elapsed: 339675 nanoseconds

Elementary Instructions: 196731

Performance on Random Input - Size 8192

BubbleSort

Time elapsed: 113560021 nanoseconds

Elementary Instructions: 150376779

OptimizedBubbleSort

Time elapsed: 96798446 nanoseconds

Elementary Instructions: 99804355

InsertionSort

Time elapsed: 29998243 nanoseconds

Elementary Instructions: 66257340

SelectionSort

Time elapsed: 27769100 nanoseconds

Elementary Instructions: 33599008

HeapSort

Time elapsed: 1595013 nanoseconds

Elementary Instructions: 762546

HoareQuicksort

Time elapsed: 704210 nanoseconds

Elementary Instructions: 362865

MergeSort

Time elapsed: 1382532 nanoseconds

```
		Elementary Instructions: 319488
	LomutoQuickSort
		Time elapsed: 781946 nanoseconds
		Elementary Instructions: 566639

Performance on Random Input - Size 16384
	BubbleSort
		Time elapsed: 442181615 nanoseconds
		Elementary Instructions: 602933948
	OptimizedBubbleSort
		Time elapsed: 350932418 nanoseconds
		Elementary Instructions: 401145541
	InsertionSort
		Time elapsed: 79367915 nanoseconds
		Elementary Instructions: 266986408
	SelectionSort
		Time elapsed: 71972236 nanoseconds
		Elementary Instructions: 134306910
	HeapSort
		Time elapsed: 1751003 nanoseconds
		Elementary Instructions: 1651976
	HoareQuicksort
		Time elapsed: 831321 nanoseconds
		Elementary Instructions: 793340
	MergeSort
		Time elapsed: 1674607 nanoseconds
		Elementary Instructions: 688128
	LomutoQuickSort
		Time elapsed: 1463795 nanoseconds
		Elementary Instructions: 1809272
```

Performance on Random Input - Size 32768

BubbleSort

Time elapsed: 1651131418 nanoseconds

Elementary Instructions: 2410947318

OptimizedBubbleSort

Time elapsed: 1668282322 nanoseconds

Elementary Instructions: 1603759290

InsertionSort

Time elapsed: 356886181 nanoseconds

Elementary Instructions: 1067003552

SelectionSort

Time elapsed: 302190724 nanoseconds

Elementary Instructions: 537049174

HeapSort

Time elapsed: 3752064 nanoseconds

Elementary Instructions: 3564267

HoareQuicksort

Time elapsed: 1677490 nanoseconds

Elementary Instructions: 1746810

MergeSort

Time elapsed: 3445239 nanoseconds

Elementary Instructions: 1474560

LomutoQuickSort

Time elapsed: 4916255 nanoseconds

Elementary Instructions: 6253461

Performance on Random Input - Size 65536

BubbleSort

Time elapsed: 6912917010 nanoseconds

Elementary Instructions: 9629555846

OptimizedBubbleSort

Time elapsed: 5989896000 nanoseconds

Elementary Instructions: 6396400261

InsertionSort

Time elapsed: 1488513020 nanoseconds

Elementary Instructions: 4249254752

SelectionSort

Time elapsed: 1396728740 nanoseconds

Elementary Instructions: 2147840250

HeapSort

Time elapsed: 6907619 nanoseconds

Elementary Instructions: 7643906

HoareQuicksort

Time elapsed: 4140296 nanoseconds

Elementary Instructions: 3683883

MergeSort

Time elapsed: 8247455 nanoseconds

Elementary Instructions: 3145728

LomutoQuickSort

Time elapsed: 19545537 nanoseconds

Elementary Instructions: 23112887

Performance on Random Input - Size 131072

BubbleSort

Time elapsed: 31001560156 nanoseconds

Elementary Instructions: 38561602380

OptimizedBubbleSort

Time elapsed: 26507172295 nanoseconds

Elementary Instructions: 25644290443

InsertionSort

Time elapsed: 5792607107 nanoseconds

Elementary Instructions: 17055294568

SelectionSort

Time elapsed: 5530731536 nanoseconds

Elementary Instructions: 8590647592

HeapSort

Time elapsed: 12541593 nanoseconds

Elementary Instructions: 16330741

HoareQuicksort

Time elapsed: 7605017 nanoseconds

Elementary Instructions: 8106268

MergeSort

Time elapsed: 20544124 nanoseconds

Elementary Instructions: 6684672

LomutoQuickSort

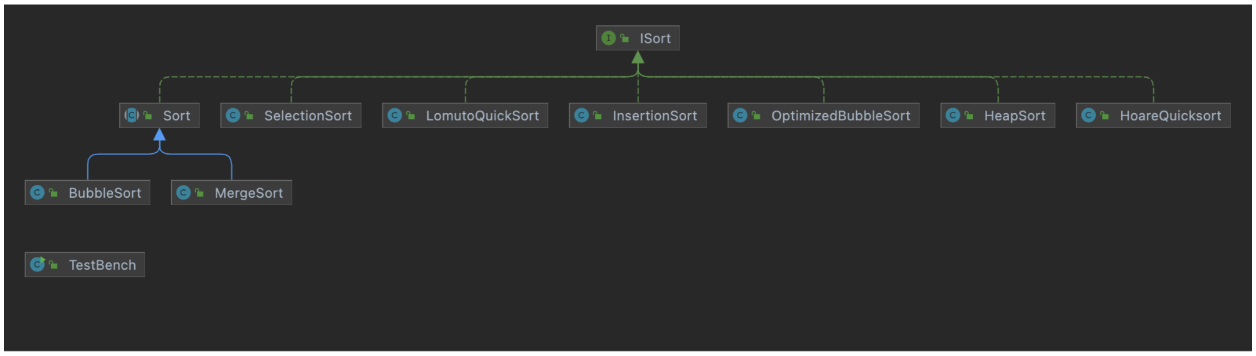
Time elapsed: 71934170 nanoseconds

Elementary Instructions: 89794171

3) Plots showing the growth rate of the algorithms as the input data size increases.

PDF attached

4) Test bench class diagram.


ISort
Sort
SelectionSort
LomutoQuickSort
InsertionSort
OptimizedBubbleSort
HeapSort
HoareQuicksort
BubbleSort
MergeSort
TestBench