Jeremiah McDonald

Intro to Algorithms

Assignment 2

Sorting Algorithms

1. Show the step-by-step process of merging two sorted lists using the merge sort algorithm. list1 = [1, 16, 25, 31] and list2 = [-3, 0, 16, 27]

Step 1: result = [-3]	list1 = [1, 16, 25, 31]	list2 = [0, 16, 27]
Step 2: result = [-3, 0]	list1 = [1, 16, 25, 31]	list2 = [16, 27]
Step 3: result = [-3, 0, 1]	list1 = [16, 25, 31]	list2 = [16, 27]
Step 4: result = [-3, 0, 1, 16]	list1 = [25, 31]	list2 = [16, 27]
Step 5: result = [-3, 0, 1, 16, 16]	list1 = [25, 31]	list2 = [27]
Step 6: result = [-3, 0, 1, 16, 16, 25]	list1 = [31]	list2 = [27]
Step 7: result = [-3, 0, 1, 16, 16, 25, 27]	list1 = [31]	list2 = []
Step 8: result = [-3, 0, 1, 16, 16, 25, 27, 31]	list1 = []	list2 = []

2. Show the step-by-step process of sorting a list using the insertion sort algorithm. The unsorted list is: [-1, -5, 67, -10, 21, 8, 4, 1]

Pass 1: [-5, -1, 67, -10, 21, 8, 4, 1] // Swap -1 and -5

Pass 2: [-5, -1, 67, -10, 21, 8, 4, 1] // No change

Pass 3: [-5, -1, -10, 67, 21, 8, 4, 1] // Swap 67 and -10

[-5, -10, -1, 67, 21, 8, 4, 1] // Swap -1 and -10

[-10, -5, -1, 67, 21, 8, 4, 1] // Swap -5 and -10

Pass 4: [-10, -5, -1, 21, 67, 8, 4, 1] // Swap 67 and 21

Pass 5: [-10, -5, -1, 21, 8, 67, 4, 1] // Swap 67 and 8

[-10, -5, -1, 8, 21, 67, 4, 1] // Swap 21 and 8

Pass 6: [-10, -5, -1, 8, 21, 4, 67, 1] // Swap 67 and 4

[-10, -5, -1, 8, 4, 21, 67, 1] // Swap 21 and 4

[-10, -5, -1, 4, 8, 21, 67, 1] // Swap 8 and 4

Pass 7: [-10, -5, -1, 4, 8, 21, 1, 67] // Swap 67 and 1

[-10, -5, -1, 4, 8, 1, 21, 67] // Swap 21 and 1

[-10, -5, -1, 4, 1, 8, 21, 67] // Swap 8 and 1

[-10, -5, -1, 1, 4, 8, 21, 67] // Swap 4 and 1

The last element is part of the sorted portion (it doesn't need to be compared to anything), the list is sorted.

3. Show the step-by-step process of sorting a list using the quicksort algorithm. The unsorted list is: list = [-5, 42, 6, 19, 11, 25, 26, -3].

Step 1: [-5, 42, 6, 19, 11, 25, 26, |-3] // Partition element is -3

left = [-5] right = [42, 6, 19, 11, 25, 26] //

list = [-5, -3, 42, 6, 19, 11, 25, 26]

Step 2: left = [] right = [42, 6, 19, 11, 25, |26] // Partition element is 26

left = [6, 19, 11, 25] right = [42] // Single element in right means greatest element

list = [-5, -3, 6, 19, 11, 25, 26, 42]

Step 3: left = [-5, -3, 6, 19, 11, |25] right = [] // Partition element is 25

left = [6, 19, 11] right = []

list = [-5, -3, 6, 19, 11, 25, 26, 42]

Step 4: left = [6, 19, |11] right = [] // Partition element is 11

left = [6] right = [19] // Seeing that there is only one element is each we are done

Sorted list = [-5, -3, 6, 11, 19, 25, 26, 42]

4. Show the step-by-step process of sorting a list using the shell sort algorithm. The unsorted list is: [15, 14, -6, 10, 1, 15, -6, 0]

Step 1: list = [15, 14, -6, 10, 1, 15, -6, 0] // arr.length = 8, so gap = 8/2 = 4

[1, 14, -6, 10, 15, 15, -6, 0] // Compare elements gap (4) apart and swap

[1, 14, -6, 10, 15, 15, -6, 0] // No swap

[1, 14, -6, 10, 15, 15, -6, 0] // No swap

[1, 14, -6, 0, 15, 15, -6, 10]

Step 2: list = [1, 14, -6, 0, 15, 15, -6, 10] // gap/ 2 = 2 Compare elements gap (2) apart and swap

[-6, 14, 1, 0, 15, 15, -6, 10]

[-6, 0, 1, 14, 15, 15, -6, 10] // No swap

[-6, 0, 1, 14, 15, 15, -6, 10] // No swap

[-6, 0, 1, 14, -6, 15, 15, 10]

[-6, 0, 1, 14, -6, 10, 15, 15]

Step 3: list = [-6, 0, 1, 14, -6, 10, 15, 15] //gap/2 = 1 (final pass) No swap // Insertion sort logic

[-6, 0, 1, 14, -6, 10, 15, 15] // No swap

[-6, 0, 1, 14, -6, 10, 15, 15] // No swap

[-6, 0, 1, -6, 14, 10, 15, 15]

[-6, 0, -6, 1, 14, 10, 15, 15]

[-6, -6, 0, 1, 14, 10, 15, 15]

[-6, -6, 0, 1, 10, 14, 15, 15]

Sorted List = [-6, -6, 0, 1, 10, 14, 15, 15]

5. Rank the six sorting algorithms in order of how you expect their runtimes to compare (fastest to slowest). Your ranking should be based on the asymptotic analysis of the algorithms. You may predict a tie if you think that multiple algorithms have the same runtime. Provide a brief justification for your ranking.

		Best Case:	Worse Case:	
1.	Merge Sort:	O(n log n)	O(n log n)	Ω(n log n)
2.	Quick Sort:	O(n log n)	O(n^2)	
	Shell Sort:	O(n log n)	O(n^2)	
3.	Bubble Sort:	O(n)	O(n^2)	
	Insertion Sort:	O(n)	O(n^2)	
	Selection Sort:	O(n)	O(n^2)	

Merge Sort is the fastest sort algorithm but requires extra space. It repeatedly divides the input list in half then merges the two halves back together.

Quick Sort and Shell Sort are tied for second place.

Quick Sort chooses a pivot element that compares the pivot to the left and right arrays. The worst case is an already sorted array, and you need to compare everything.

Shell Sort uses a gap int that continuously divides the array by 2. Once the gap = 1 it uses insertion sort logic for the final pass.

Bubble Sort, Insertion Sort, and Selection Sort all share the same time complexity.

All three of these sorting algorithms use nested loops that iterate through the array to compare the elements.

10. In my tests, I observed significant differences in the relative performance of the sorting algorithms. As expected, Bubble Sort performed the worst by a large margin, taking considerably more time than the other algorithms. Insertion Sort and Selection Sort were both noticeably faster than Bubble Sort, which was surprising given their similar worst case scenario. Shell Sort performed much better than I had originally thought given that is utilizes insertion sort logic.