1. Merging with MergeSort:

* Lists: [1, 25, 31, 16] and [-3, 0, 16, 27]
  + 1. Create an empty array to store the merged list.
  + 1. The first array is not properly sorted, so the array will have to be divided into single elements. Since the first two elements are in order, they say the same. But because 31 is higher than 16, 16 will be moved before 31. Now that both halves of the first array are sorted, they can be re-merged.
    - Compare 1 to 16. 1 is added. Pointer for first half is moved. [1]
    - Compare 16 to 25. 16 is added. Pointer for second half is moved. [1,16]
    - Compare 25 to 31. 25 is added before 31. [1,16,25,31]
  + Now that both arrays are sorted, they can be merged.

Sorted Lists: [1,16,25,31] and [-3,0,16,27]

* + - Start with index 0 for both arrays and compare the elements 1 and -3.
      * -3 < 1, so -3 is added. Move pointer for 2nd array. [-3]
    - Compare 1 to 0
      * 0 < 1, so 0 is added. Move 2nd array pointer. [-3, 0]
    - Compare 1 to 16
      * 1 < 16, so 1 is added. Move 1st array pointer. [-3,0,1]
    - Compare 16 to 16.
      * 16 = 16, so both are added. Move both pointers.[-3,0,1,16,16]
    - Compare 25 to 27.
      * 25 < 27, so 25 is added. Move 1st array pointer. [-3,0,1,16,16,25]
    - Compare 31 to 27.
      * 27 < 31, so 27 is added before 31. [-3,0,1,16,16,25,27,31].

1. Insertion Sort:

* List: [-1, -5, 67, -10, 21, 8, 4, 1]
  + Start with index 1, -5, and move it to it’s correct place. -5 is inserted before -1. [-5,-1,67,-10, 21, 8, 4, 1].
  + Move on to 67. 67 is already greater than -5, so it stays in its current position.
  + Move to -10. -10 is less than -5, so it’s added before -5. [-10,-5,-1,67,21,8,4,1]
  + Move to 21. 21 is less than 67, so its moved before 67. [-10,-5,-1,21,67,8,4,1]
  + Move to 8. 8 is less than 21, so it’s moved before 21. [-10,-5,-1,8,21,67,4,1]
  + Move to 4. 4 is less than 8, so it’s moved before 8. [-10,-5,-1,4,8,21,67,1]
  + Move to 1. 1 is less than 4, so it’s moved before 4. [-10,-5,-1,1,4,8,32,67]

1. Quicksort:

List: [-5,42,6,19,11,25,26,-3]

Left = -5, Right = -3, Center = 11.

|  |  |  |
| --- | --- | --- |
| S1 = -5, 6, -3 | P = 11 | S2 = 42, 19, 25, 26 |
| S1 = -5,-3 | P = 6 | 11 | P = 19 | S2 = 42, 25, 26 |
| -5,-3,6 | 11 | 19 | P = 25 | S2 = 42,26 |
| -5,-3,6 | 11 | 19, 25 | 26, 42 |

Sorted List: -5,-3,6,11,19,25,26, 42

1. Shell sort:

* List: [15, 14, -6, 10, 1, 15, -6, 0]
* First iteration:
  + Length = 8, gap = 4.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | -6 | 10 | 1 | 15 | -6 | 0 |
| 1 | 14 | -6 | 0 | 15 | 15 | -6 | 10 |

* Second iteration:
  + Gap = 2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 14 | -6 | 0 | 15 | 15 | -6 | 10 |
| -6 | 0 | 1 | 14 | 15 | 15 | 15 | 15 |
|  |  | 1 | 14 | -6 | 10 |  |  |
|  |  |  |  |  |  |  |  |
| -6 | 0 | 1 | 14 | -6 | 10 | 15 | 15 |

* Third iteration:
  + Gap = 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -6 | 0 | 1 | 14 | -6 | 10 | 15 | 15 |
| -6 | 0 | 1 | 14 | 14 | 14 | 15 | 15 |
|  | 0 | 1 | -6 | 10 | 14 | 15 |  |
| -6 | 0 | 1 | -6 | 10 | 14 | 15 | 15 |

1. Ranking from 1(best) to 6(worst):
2. Merge Sort – time complexity is (O(n log n)) in all cases, which is the slowest-growing function.
3. Quick Sort – best case scenario O(n log n), worst case scenario O(n^2) if the function picks a poor pivot element.
4. Shell Sort – Best case scenario O(nlog^2n), which grows at a faster rate than the complexities for merge or quick sort, worst case O(n^2).
5. Bubble Sort and Insertion – Best case O(n), worst case O(n^2)
6. Selection Sort – O(n^2) for all complexities