1. Merge sort: List1 = [1, 2, 3, 6] and List 2 = [-3, 0, 6, 7]

1 compared to –3 ------ > add –3 to list since its smaller

1 compared to 0 ------> add 0 to list since its smaller

1 compared to 6 -------> add 1 to list since its smaller

2 compared to 6 -------> add 2 to list since its smaller

3 compared to 6 ------> add 3 to list since its smaller

6 compared to 6 -----> add 6 to list since its smaller

Compare 6 and 7 and add 6 to the list, before finally adding 7.

Final merged list: [-3, 0, 1, 2, 3, 6, 6, 7]

2. Insertion Sort: List = [-21, 5, 7, -10, 61, 8, 2, 10]

Start with the first element –21, compare it to the next value to the right which is 5. Since they are sorted, they stay where they are. Then compare 7 to 5 and since they are sorted, we keep them there again. Then compare –10 with 7 and since –10 is smaller than 7 we swap them. Then compare -10 to 5 and swap again. Next, we compare –10 to –21, and no swap occurs. The sorted section is [-21, -10, 5, 7] at this point. Then comparing 61 to 7, they stay in their places. Next, compare 8 to 61 and swap them. Compare 8 to 7 and do not swap. Next compare 2 to 61 and swap. After comparing, swap 2 with 8, then 2 with 7, then swap 2 with 5. Compare 2 to –10, and do not swap. Finally, compare 61 and 10, swap them, and then compare 10 to 8 and do not swap them. The sorted list is [-21, -10, 2, 5, 7, 8, 10, 61]

3. QuickSort: List = [-5, 4, 2, 619, 11, 5, 620, -3]

I will use the first number as the pivot element.

- Step 1:
    - Pivot = -5, left partition = [], right partition = [4, 2, 619, 11, 5, 620, -3]
    - [] + [-5] + [4, 2, 619, 11, 5, 620, -3]
- Step 2:
    - Sort right partition
    - Pivot = 4, left partition = [2, -3], right partition = [619, 11, 5, 620]
    - [2, -3] + [4] + [619, 11, 5, 620]
- Step 3:
    - Recursively sort the right partition

    - Pivot =2, left partition = [-3], right partition = []
- Steo 4: Sort [619, 11, 5, 620]
    - Pivot = 619, right partition = [620], left partition = [11,5]
    - Sort [11, 5] ----->pivot==11, left partition = [5], right partition = []
    - [5, 11, 619, 620]

- Final sorted array: [-5, -3, 2, 4, 5, 11, 619, 620].

4. Shell Sort: List = [5, 10, 60, 0, -1, 34, 6, 10]

- Step 1: Gap = n/2 = 8/2 = 4
    - Compare elements 4 positions apart, creating 4 sublists.
    - Sublist 0 ([0]-[4]) | -1 < 5, so SWAP ---> [-1, 5]
    - Sublist 1 ([1]-[5]) | 10 < 34, so NO CHANGE ---> [10, 34]
    - Sublist 2 ([2]-[6]) | 6 < 60, so SWAP ---> [6, 60]
    - Sublist 3 ([3]-[7]) | 0 < 10, so NO CHANGE ---> [0,10]
      NOW - [-1, 10, 6, 0, 5, 34, 60, 10]
- Step 2: Gap = 4/2 = 2
    - Compare elements 2 positions apart, creating 2 sublists.
    - Sublist 0 (indices [0]-[2]-[4]-[6]): [-1, 6, 5, 60]
        - -1 stays. Then 6 > -1, stays. Then 5 < 6, insert before [2] ---> [-1, 5, 6, 60]. Then 60 > 6, stays ---> [-1, 5, 6, 60]
    - Sublist 1 (indices [1]-[3]-[5]-[7]): [10, 0, 34, 10]
        - 10 stays. 0 < 10, insert before [1] ---> [0, 10, 34, 10]. Then compare 34 > 10, stays ---> [0, 10, 34, 10]. Lastly, 10 < 34, insert before [5] ---> [0, 10, 10, 34]

NOW: [-1, 0, 5, 10, 6, 10, 60, 34]

- Step 3: **Gap = 2/2 = 1**
    - [-1, 0, 5, 10, 6, 10, 60, 34]
        - [0]-[3]: already sorted. Next compare 6 < 10, insert between [2] and [3]. 10 is already in place and so is 60. Move on to comparing 34 < 60, and insert between [5] and 6

Final list : [-1, 0, 5, 6, 10, 10, 34, 60]

6. Ranking

1. Quick Sort

I think this will be the fastest bc it has an best case runtime of O(n log n)

2. Merge Sort has the same time complexity, but a higher space complexity than quick sort, which is why I think it will be second.
3. Shell Sort
4. Selection Sort
5. Insertion Sort
6. Bubble Sort

10. When running the performance calculations I saw Bubble Sort was in fact the slowest, and Merge, Selection, and Shell Sorts were tied for the fastest. Bubble Sort in the middle, and funny enough, QuickSort was the last.