Matthew Ostin
CSCI 1112

Selection Sort
100 Classes
profileSelection::allocs:3, compares:4950, swaps:99
1000 Classes
profileSelection::allocs:3, compares:499500, swaps:999
10000 Classes
profileSelection::allocs:3, compares:49995000, swaps:9999

Selection Sort works by dividing the input array into two parts which are the sorted and unsorted parts. This repeatedly finds the minimum element from the unsorted part and swaps it with the first element of the sorted part until the array is finally sorted. The growth of the allocations, comparisons and swaps in Selection Sort is consistent with the algorithm's strategy. Selection Sort in terms of memory only requires a constant amount of additional memory. This means that it does not depend on the size of the input array. The number of comparisons grows quadratically with the size of the input. For n input elements, the algorithm makes n-1 comparisons then in the next iterations n-2 and so on. This results in (n-1) + (n-2) + ... + 1 = n(n-1)/2 comparisons. The number of swaps in Selection Sort grows linearly with the size of the input array, this can be seen from the profile data.

Bubble Sort
100 Classes
profileBubble::allocs:3, compares:4935, swaps:2949
1000 Classes
profileBubble::allocs:3, compares:499490, swaps:313555
1000 Classes
profileBubble::allocs:3, compares:49995000, swaps:44522036

Bubble Sort works by comparing adjacent elements and swapping them if they are in the wrong order. The growth of allocations, comparisons, and swaps is also consistent with its strategies. It requires a constant amount of additional memory, and its allocations remain constant. Bubble Sort grows quadratically.

Insertion Sort
100 Classes
profileInsertion::allocs:3, compares:2466, swaps:2375
1000 Classes

profileInsertion::allocs:3, compares:250009, swaps:249016
10000 Classes
profileInsertion::allocs:3, compares:24985192, swaps:24975201

Insertion Sort works by dividing the input array into two parts, the sorted part and the unsorted part. Then it repeatedly interests elements from the unsorted part into the correct position in the sorted part. Only requires an additional amount of memory. The number of sorts grows quadratically with the size of the input array. The number of sways is proportional to the number of inversions in the input array.

Quick Sort
100 classes
profileQuicksort::allocs:808, compares:783, swaps:286
1000 Classes
profileQuicksort::allocs:8224, compares:13984, swaps:7827
10000 Classes
profileQuicksort::allocs:105160, compares:201583, swaps:75984

Quick Sort is known for its “Divide-and-conquer” strategy to sort the input array. The algorithm divides the input array into two subarrays around a pivot element and then sorts the two subarrays. The growth of allocations, comparison, and swaps is also consistent with its strategy. Only requires a small amount of additional memory for the recursive call stack. The number of allocations in Quick Sort is small. The number of comparisons in Quicksort is proportional to the size of the input array times the log of the size of the input array.