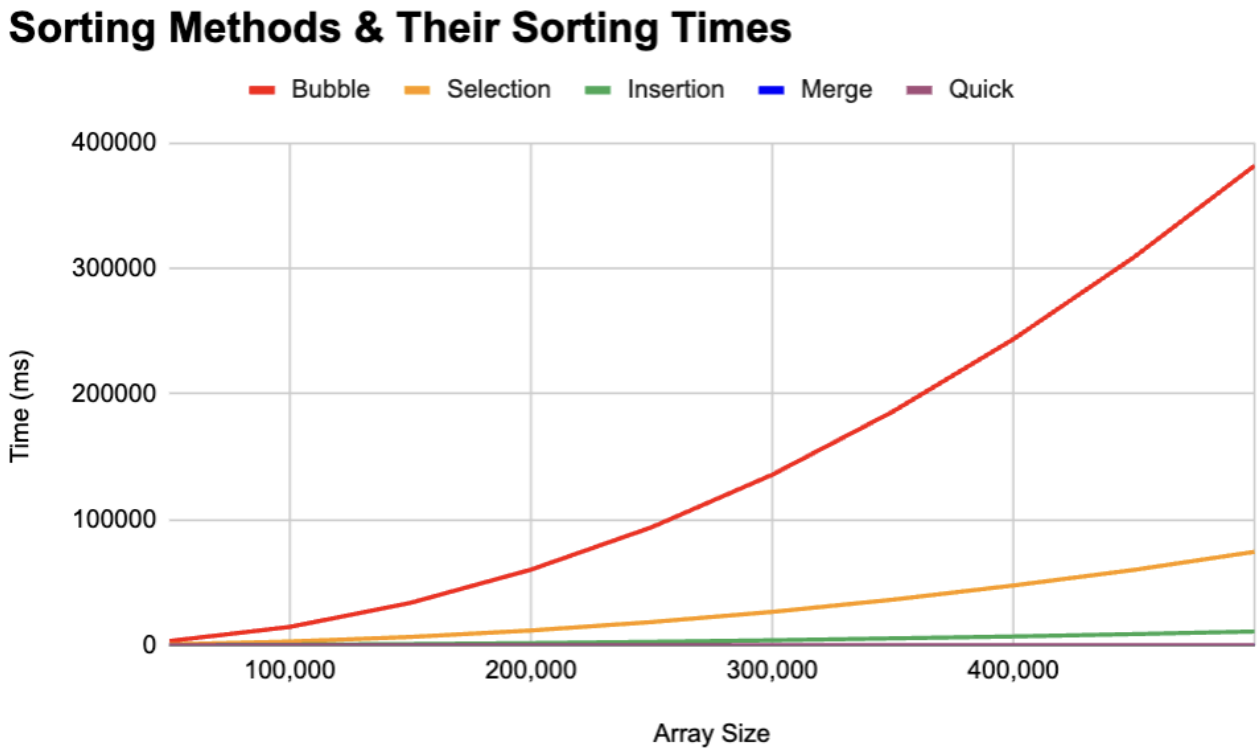
Project 1: Sorting Algorithms & Quicker than Quicksort

Sorting Algorithms

In the program Sorting Algorithms.java, we compare different sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort. An array of floats is passed into these functions ; it is randomly generated. My program has a "copy" function, which ensures that the same array is being sorted by all 5 algorithms. After each array is sorted, the sorted array is passed into checkSort to verify it is sorted. They are timed in milliseconds.



This graph illustrates the time it takes for each sorting algorithm to sort arrays of lengths increasing by 50,000 (starting at 50,000 and ending at 500,000). A quick recap of the sorting algorithms:

Bubble Sort: If value to the right is smaller, swap with use of temp variable.
O(n^2)

Selection Sort: find the smallest element of the array, and place it at the front of the unsorted array.

O(n^2)

Merge Sort: divides the array into smaller sub-arrays, sorts each array recursively, and then merge them.

O(nlogn)

Quick Sort: picks an element as a pivot and partitions the given array around the picked pivot, by placing the pivot in its correct position in the array. Then, the subarrays around the pivot are independently sorted recursively.

O(nlogn)

Quicker than QuickSort

For large arrays, quicksort is generally seen as the fastest of these five. However, on smaller arrays, it is more likely for quicksort to be less efficient than the quadratic sorting methods, such as Bubble Sort. In the program quickerthanquicksort.java, we combine Quick Sort and Bubble Sort. We implement bubble sort at the cutoff point in which we find quick sort becomes inefficient (The cutoff point being the size of a certain subarray). I chose a subarray with length 50 or lower.

I chose bubble sort as the quadratic algorithm because bubble sort is the simplest and the shortest. In comparison to insertion and selection, bubble is only a few lines. It is often seen as the slowest sorting algorithm, so I thought it would be interesting to see if even bubble sort could lead to an improvement over quicksort.

To try to find the cutoff point in which quicksort would turn to bubble sort, I began by simply choosing an n (n = arr.length) of randomly generated floats. The sort time for bubble sort could be approximately an^2 and the time to sort the same array with quick sort is approximately $bn\log(n)$. The value of a can be calculated by $a = t_b/n^2$ (t_b being the time bubble sort took). The value of b can be calculated by $b = t_q/n\log(n)$ (t_q being the time quicksort took). The values of a and b dependent heavily on the n chosen.

I then proceed as though the time to sort the array using bubble sort was a times n^2 , and the time to sort using quicksort was b times $n\log(n)$. I expect that bubble sort on an array size n

will take less time than quicksort if a times n^2 is less than b times $n\log(n)$. That can be represented by the inequality a/b < log(n)/n.

We will then introduce the variable c, for cutoff. This will represent the cutoff point, such that a/b < log(c)/c . The cutoff point can be seen by inspection of a graph log(x)/x and a/b. We find the x for which a/b = log(x)/x. Afterwards, the suspect cutoff point will be the floor of x.

I found my cutoff point to be about 55 on the graph. I tested this by implementing bubble sort after quick sort, when the sub-array became about 55 elements or less. I saw that for large n, 50 seemed to be a good cutoff point. Hybrid sort was generally faster than quick sort using 50 for the cutoff point.