**Sorting Algorithm Summary**

| **Sorting Algorithm** | **Time Complexity** | **Space Complexity** | **Stable?** |
| --- | --- | --- | --- |
| Bubble Sort | *O*(*N*2) | *O*(1) | Yes |
| Insertion Sort | *O*(*N*2) | *O*(1) | Yes |
| Selection Sort | *O*(*N*2) | *O*(1) | No |
| Heap Sort | O(*N*log*N*) | *O*(1) | No |
| Counting Sort | *O*(*N*+*K*) | *O*(*N*+*K*) | Yes |
| Radix Sort | *O*(*WN*+*WK*) | *O*(*N*+*K*) | Yes |
| Bucket Sort | *O*(*N*2) -- *O*(*N*+*K*) average | *O*(*N*+*K*) | Yes |

Note: For radix sort, *W* is the maximum number of characters/numbers of each element in the array, and *K* is the maximum alphabet size. For counting sort, assuming non-negative integers, *K* is the maximum element in the input. For all sorting methods, *N* is the size of the input array.

In this explore card, we learned about the intricacies of a variety of sorting algorithms. We dived into comparison sorts and non-comparison based sorts. All sorts have tradeoffs, whether its for simplicity or efficiency, so the most important aspect of sorting to really understand is what sort of tradeoffs were made in the algorithm. There are a lot of good sorting algorithms out there, but there is no perfect algorithm. Depending on the situation, a sort that may be less efficient may actually be the best sort (e.g. insertion sort for almost sorted arrays). And other times, a non-comparison sort may be the best approach for an input. But there may also be situations where even though non-comparison sorts theoretically perform better, a comparison-based sort with less memory footprint may be a better choice. Being able to understand the advantages and disadvantages of a variety of sorts in various contexts is the most important lesson to take away from this explore card.