**Sorting Algorithms**

|  |  |
| --- | --- |
| **Algorithm** | **Strategy** |
|
| Bubble Sort | Keep on iterating through the list, comparing two adjacent elements and swapping them if they are in the incorrect order. Repeat until the entire list has been iterated through, meaning the entire list has been sorted. |
| Selection Sort | Create “unseen” divisions of the list with a sorted section and a non-sorted section. In the non-sorted section, find the smallest element and move it to the correct position in the sorted section of the list, and repeat until there are no elements in the non-sorted section. |
| Insertion Sort | Like SelectionSort, divide the list into two sections. Iterate through the list and compare each element to the previous ones behind it. Shift the elements to the right to insert the current element until it is in the correct position of the sorted part of the list. |
| QuickSelect | Given a pivot element and n, we partition the list using recursion until the pivot element is the nth smallest element, which is what we’re trying to find. |
| QuickSort | Given a pivot element, we partition the list based on it. Then using recursion, we QuickSort the left and right partitions until we’ve sorted the entire list. |
| Merge Sort | We divide the list into two halves (that may or may not have equal length), then we recursively call Merge Sort for each half, then merge those two sorted halves into a single sorted list. |

**Pros/Cons and Time Complexity**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Advantages** | **Disadvantages** | **Time Complexity** | | |
| **Best Case** | **Avg. Case** | **Worst Case** |
| Bubble Sort | Simple to implement, and quite efficient for smaller lists. | Inefficient for large lists, high time complexity since it must loop/iterate through the entire list. | O(n) | O(n^2) | O(n^2) |
| Selection Sort | Also simple to implement, and efficient for smaller lists. | Similar to Bubble Sort, inefficient for large lists, and high time complexity stays the same throughout all cases. | O(n^2) | O(n^2) | O(n^2) |
| Insertion Sort | Relatively simple to implement, and efficient for small lists or lists that are nearly sorted. | Inefficient for large and mostly unsorted lists, time complexity in average and worst case is high. | O(n) | O(n^2) | O(n^2) |
| QuickSelect | Is an efficient way to find the nth smallest element in an unsorted list without fully sorting it. | Cannot sort the entire list, only used to find a single element, worst time complexity is high. | O(n) | O(n) | O(n^2) |
| QuickSort | Pretty efficient for large and random lists, average time complexity is pretty low compared to the other methods. | Worst time complexity is high, best case is the same as the average case (not necessarily a con) | O(n log n) | O(n log n) | O(n^2) |
| Merge Sort | Efficient for large and random lists, all time complexity is constant no matter what the case is | Harder to implement, may require an auxiliary method that merges two sorted lists | O(n log n) | O(n log n) | O(n log n) |