**Sorting Algorithms**

|  |  |
| --- | --- |
| **Algorithm** | **Strategy** |
|
| Bubble Sort | Repeatedly sort adjacent elements until the entire array is sorted. |
| Selection Sort | Starting at index 0, traverse the array to find the smallest element. Replace element at index 0 with smallest element. Repeat until the second-to-last index. The array is now sorted. |
| Insertion Sort | Starting at index 1 (current index = 1), traverse the array (towards index 0) and insert the element in the correct position (i.e. when we encounter the first element smaller than the element at the current index, insert the element at the current index after that element). Increment the current index until the last index and repeat the operation. |
| QuickSelect | Select randomly a pivot. Use the partition strategy to find the index of that pivot. If the index of pivot + 1<k, repeat the operation on the right side of the pivot. If the index of pivot + 1>k, repeat the operation on the left side. If the index of the pivot+1 = k, return the element at that index. |
| QuickSort | Partition the array around a random pivot. Recursively apply QuickSort to the left and right side of the pivot until each side has a size of 1 (a single element). The array is now sorted. |
| Merge Sort | Continuously divide the array in half until each subarray has size = 1. Each subarray is now sorted. Merge the subarrays together so that the merged arrays are sorted (by taking the smallest element of each subarray and compare with the smallest element of the other subarray, repeated until all elements of the 2 subarrays are sorted). When all subarrays have been merged, the array is sorted. |

**Pros/Cons and Time Complexity**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Advantages** | **Disadvantages** | **Time Complexity** | | |
| **Best Case** | **Avg. Case** | **Worst Case** |
| Bubble Sort | Easy to implement  Reduces memory needed | Takes a long time to sort big arrays | O(n) | O(n^2) | O(n^2) |
| Selection Sort | Easy to implement  Reduces memory needed | Takes a long time to sort big arrays | O(n^2) | O(n^2) | O(n^2) |
| Insertion Sort | Easy to implement  Reduces memory needed | Takes a long time to sort big arrays | O(n) | O(n^2) | O(n^2) |
| QuickSelect | Relatively quick for large arrays | Poor performance if array contains repeated values. | O(n log(n)) | O(n log(n)) | O(n^2) |
| QuickSort | Relatively quick for large arrays | Poor performance if array contains repeated values. | O(n log(n)) | O(n log(n)) | O(n^2) |
| Merge Sort | Relatively quick for large arrays  Same time complexity for best and worst case | Slower for smaller arrays  Requires more memory | O(n log(n)) | O(n log(n)) | O(n log(n)) |