Bubble sort

* Comparison based
* Compare first and second. If not in order, swap. Then keep on comparing second and third
* ***Optimized bubble sort : swapped flag*** . If the value does not change in an iteration of loop this means array is now ordered and no need to continue.

Insertion Sort :

* Comparison based
* Shift previous elements on the left side of current index to the right until you find the perfect spot of the current element.
* ***Inefficient for large data sets***
* ***Faster than others for small data set.***

Selection Sort :

* Comparison -based
* Check all the element and move the smallest one found (meaning swap occurs at the end of iteration) to the front.
* Inefficient for large data sets , prefer for small data sets
* Can be used to check if the list is ordered or not.
* Can be used when there are memory concerns (since swap occurs at the end of iteration)
* Prefer to use when the array is NOT partially sorted and checking of all elements is compulsory.

Merge Sort :

* Divide and conquer
* Recursively divide array into two halves until single element sub-arrays are reached (base -case)
* Merge by sorting two sub-sequent sub -arrays
* Highly parallelizable , time -efficient with time complexity of O(n log n) is guaranteed
* Not as space -efficient as other sorting algorithms.
* Preferred for linked list

Quick Sort

* Divide and conquer
* Partition array with random pivot(usually rightmost)
* Place smaller to left , larger to right of pivot
* Computer Architecture favours
* Preferred when time and space complexity matters.
* Perform better than merge sort in practice
* Preferred for arrays

Counting sort :

* Count of elements are stored in helper count array ( sized [max+1] where max is the max value found)
* Update count array to store cumulative sum of its elements (to place then into correct indices).
* ***Prefer to use it when the difference between different keys (values ) are not so big.***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Complexity | Bubble Sort | Insertion sort | Selection Sort | Merge Sort | Quick Sort | counting Sort |
| Time |  |  |  |  |  |  |
| Best : | O(n) | O(n) | O(n2) | O(nlogn) | O(nlogn) | O(n+k) |
| Worse / Avg : | O(n2) | O(n2) | O(n2) | O(nlogn) | O(n2) | O(n+k) |
| Space : | O(1) | O1) | O(1) | O(n) | O(logn) | O(max) |