|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algorithms | Best | Worst | Average | Space | Explanation | Evolution |
| Selection sort | 0(n^2) | 0(n^2) | 0(n^2) | 0(1) | 1. Array is divided into first half and second half. First half will be sorted, second half will be unsorted. 2. Find the lowest element in the unsorted array. 3. Place at the start of First sorted array. 4. Repeat this until array is sorted. | 1. Selection sort can be used on list DS, that make add and remove efficient. 2. Not stable sort. Keys will be changed. 3. Selection sort makes 0(n) swaps which is minimum among all sorting. |
| Bubble sort | 0(n),0(1) | 0(n^2) | 0(n^2) | 0(1) | 1. Repeatedly comparing and swaps adjacent elements in every pass. 2. In i-th pass, last element (i-1) is already sorted. 3. i-th largest element is placed at the (n-i)the position. | 1. Stable sort, it doesn’t change the key (index) values. 2. In place sort 3. Slower algorithm comparatively. |
| Insertion sort | 0(n) | 0(n^2) | 0(n^2) | 0(1) | 1. Iterating over the array. Choose i-th index as key. 2. Compare i-th element with predecessor. 3. If key smaller than predecessor, compare with before elements. Moving the greater elements to one position up. | 1. When the range is smaller it is preferred. 2. Useful when the input array is sorted. 3. Stable sorting. |
| Meger sort | 0(nlogn) | 0(nlogn) | 0(nlogn) | 0(n) | 1. Divides the array into two equal half. 2. Then will be sorted merged arrays will be combined. 3. Find the mid element, made the array into half. From First element to the middle. 4. Middle to last element | 1. Merge Sort is useful for sorting in Linked Lists. 2. Insertion in Linked Lists is 0(1) 3. Implemented without extra space. |
| Quick sort | 0(nlogn) | 0(nlogn) | 0(n^2) | 0(n) | 1. Choose the pivot value, either first, middle,last. 2. Put the pivot element in such way, right side of pivot element will have larger values. Pivot elements left side will be have smaller value | 1. Quick Sort is not stable. However any sorting algorithm can be made stable by considering the indexes as parameters. |
| Counting sort | 0(N+K) | 0(N+K) | 0(N+K) | 0(K) | 1. Counting sort is based on keys with specific range. 2. It counts number of objects having distinct | 1. It is efficient when range of input data is not significantly greater than number of objects to be sorted. 2. Subroutine to radix sort 3. Counting sort uses partial hashing to count the occurrence of data |
| Bucket sort | 0(N+K) | 0(N^2) | 0(N+K) | 0(N) | 1. Creating n empty bucket lists. 2. Inserting each input array element by using n\*arr[i]. 3. Sorting the buckets. 4. Concatenate the buckets | 1. Bucket sort is mainly useful, when the data is uniformly distributed over a range |
| Radix sort | 0(d\*(N+b)) | 0(d\*(N+b)) | 0(d\*(N+b)) | 0(N+K) | 1. Finding the maximum element from the given array. 2. Iterate over the digits of elements in range of LSB to MSB array. 3. Depending on the digits place them in ascending order. 4. Once all the digits are iterated array is sorted. | 1. Radix sort, quicker for larger range of inputs. Better than quick sort. 2. Radix sort uses counting sort as subroutine and counting sort takes extra space to sort the elements. |