**1. Quick Sort:** Quicksort is a [divide-and-conquer algorithm](https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm). It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted [recursively](https://en.wikipedia.org/wiki/Recursion_(computer_science)). This can be done [in-place](https://en.wikipedia.org/wiki/In-place_algorithm), requiring small additional amounts of [memory](https://en.wikipedia.org/wiki/Main_memory) to perform the sorting.

**Application: when space & time complexity matters. Good for the programming language handles recursion well.**

**Time Complexity**

|  |  |  |
| --- | --- | --- |
| **Best case** | **O(nlogn)** | In the most balanced case, each time divide the list into two nearly equal pieces. Divide operation happens n times. |
| **Worst case** | **O(n2)** | When pivot element happens to be the smallest or largest element in the list of array or if all the element in the list are equal. |

**Space Complexity**

|  |  |  |
| --- | --- | --- |
| **Best case** | **O(nlogn)** | Quick sort use in-place partitioning and at most use O(logn) space in both best and worst case. |
| **Worst case** | **O(nlogn)** | Quick sort use in-place partitioning and at most use O(logn) space in both best and worst case. |

**2. Heap Sort:** In [computer science](https://en.wikipedia.org/wiki/Computer_science), **heapsort** is a [comparison-based](https://en.wikipedia.org/wiki/Comparison_sort) [sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm). Heapsort can be thought of as an improved [selection sort](https://en.wikipedia.org/wiki/Selection_sort): like selection sort, heapsort divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element from it and inserting it into the sorted region.

**Application**: Heapsort is use in real time embedded system when data receives in packets. Its best choice if need to find the greatest or largest or kth place element from the array.

**Time Complexity**

For array with equal data takes O(n)

|  |  |  |
| --- | --- | --- |
| **Best case** | **O(nlogn)** | *Build\_maxheap* has complexity ***O(N)*** and we run max\_heapify *N−1* times in heap\_sort function, therefore complexity of heap\_sort function is *O(NlogN).* |
| **Worst case** | **O(nlogn)** | *Build\_maxheap* has complexity ***O(N)*** and we run max\_heapify *N−1* times in heap\_sort function, therefore complexity of heap\_sort function is *O(NlogN).* |

**Space Complexity**

|  |  |  |
| --- | --- | --- |
| **Best case** | **O(1)** | Heap sort heapify the element and then swap the root element with the leaf node at bottom with in-place recursion |
| **Worst case** | **O(1)** | Heap sort heapify the element and then swap the root element with the leaf node at bottom with in-place recursion |

**3. Merge Sort:**   Its an efficient, general-purpose, [comparison-based](https://en.wikipedia.org/wiki/Comparison_sort) [sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm). Most implementations produce a [stable sort](https://en.wikipedia.org/wiki/Sorting_algorithm#Stability), which means that the order of equal elements is the same in the input and output. Merge sort is a [divide and conquer algorithm](https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm)

**Application**: merge sort is a stable sort and is more efficient at handling slow-to-access sequential media. Merge sort is often the best choice for sorting a [linked list](https://en.wikipedia.org/wiki/Linked_list). It’s not the best chose where space is the constraint.

**Time Complexity**

For array with equal data takes O(n)

|  |  |  |
| --- | --- | --- |
| **Best case** | **O(nlogn)** |  |
| **Worst case** | **O(nlogn)** |  |

**Space Complexity**

|  |  |  |
| --- | --- | --- |
| **Best case** | **O(1)** | O(1) with linked list |
| **Worst case** | **O(n)** | O(n) with O(n) auxiliary |

**Binary Search Tree**

|  |
| --- |
| [**Time complexity**](https://en.wikipedia.org/wiki/Time_complexity)**in**[**big O notation**](https://en.wikipedia.org/wiki/Big_O_notation) |
| |  |  |  |  | | --- | --- | --- | --- | | **Algorithm** |  | **Average** | **Worst case** | | **Space** |  | O(*n*) | O(*n*) | | **Search** |  | O(log *n*) | O(*n*) | | **Insert** |  | O(log *n*) | O(*n*) | | **Delete** |  | O(log *n*) | O(*n*) | |

**Worst case time is O(n2) if list is already sorted and all the element chained into a linked list.**