**SORTING ALGORITHMS:**

**SORTING:**

Why do we do sorting?

Sorting Algorithms:

* Selection sort
* Bubble Sort
* Insertion sort
* Merge Sort
* Quick Sort

Techniques and Concepts:

* Recursive or Iterative
* Comparison or non-comparison based
* Divide and Conquer
* Best/Worst/Average case bounds
* Randomized algorithms

**Selection Sort:**

**(Context: Ascending Order)**

**The Idea:**

Given an array of n items

1.Find the largest item x, in the range of [0...n−1]

2.Swap x with the (n−1)th item

3.Reduce n by 1 and go to Step 1

**Illustration:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 10 | 14 | 37 | 13 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 10 | 14 | 13 | 37 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13 | 10 | 14 | 29 | 37 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13 | 10 | 14 | 29 | 37 |  |
| 10 | 13 | 14 | 29 | 37 |  |

**Code:**

**Efficiency and Cost: O(n2)**

**Bubble Sort:**

**The Idea:**

Given an array of n items

1 Compare pair of adjacent items

2.Swap if the items are out of order

3.Repeat until the end of array

- The largest item will be at the last position

4.Reduce n by 1 and go to Step 1

**Illustration:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 10 | 14 | 37 | 13 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 14 | 29 | 13 | 37 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 14 | 13 | 29 | 37 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 13 | 14 | 29 | 37 |  |

**Code:**

Efficiency and Cost: O(n2)

Worst Case: Best Case:

**Insertion Sort:**

Similar to how people play cards:

Start with one card

Pick another card;

Insert the card in another into the correct position

**Illustration:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 10 | 14 | 37 | 13 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 29 | 14 | 37 | 13 |  |
| 10 | 14 | 29 | 37 | 13 |  |
| 10 | 14 | 29 | 37 | 13 |  |
| 10 | 13 | 14 | 29 | 39 |  |

**Code:**

**Efficiency and Cost:**

Best Case: O(n) ; Worst Case: O(n2)

**Merge Sort:**

**The Idea:**

Merge Sort is a divide-and-conquer sorting algorithm

Divide step

* Divide step
* Divide the array into two (equal) halves
* Recursively sort the two halves

Conquer step

Merge the two halves to form a sorted array

Illustration:

7263845

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7 | 2 | 6 | 3 | 8 | 4 | 5 |

7263 | 845

72| 63|| 84| 5

27 36|| 48 5

273 6 || 485

237 6 || 458

2376 || 458

2367 || 458

23674 || 58

23467 || 58

234675| 8

234567| 8

2345678

Time complexity and Efficiency: n log n

**Quick Sort:**

Quick Sort is a divide-and-conquer algorithm

Divide step

Choose an item p (known as pivot) and partition the items of a[i...j] into two parts

* Items that are smaller than p
* Items that are greater than or equal to p
* Recursively sort the two parts

Conquer step

Do nothing

In comparison, Merge Sort spends most of the time in conquer step but very little time in divide step

3, 42, 21, 6, 12, 56

3, 6, 12, 21, 42, 56