**Time Complexity and Space Complexity**

Generally, there is always more than one way to solve a problem in computer science with different algorithms. Therefore, it is highly required to use a method to compare the solutions in order to judge which one is more optimal. The method must be:

* Independent of the machine and its configuration, on which the algorithm is running on.
* Shows a direct correlation with the number of inputs.
* Can distinguish two algorithms clearly without ambiguity.

**Time Complexity:**The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.

|  |  |  |
| --- | --- | --- |
| **Input Length** | **Worst Accepted Time Complexity** | **Usually type of solutions** |
| 10 -12 | O(N!) | [Recursion](https://www.geeksforgeeks.org/recursion/) and [backtracking](https://www.geeksforgeeks.org/backtracking-algorithms/) |
| 15-18 | O(2N\* N) | Recursion, backtracking, and[bit manipulation](https://www.geeksforgeeks.org/bits-manipulation-important-tactics/) |
| 18-22 | O(2N\* N) | Recursion, backtracking, and bit manipulation |
| 30-40 | O(2N/2\* N) | [Meet in the middle](https://www.geeksforgeeks.org/meet-in-the-middle/), [Divide and Conquer](https://www.geeksforgeeks.org/divide-and-conquer-introduction/) |
| 100 | O(N4) | [Dynamic programming](https://www.geeksforgeeks.org/dynamic-programming/), [Constructive](https://www.geeksforgeeks.org/basic/constructive-algorithms/) |
| 400 | O(N3) | Dynamic programming, Constructive |
| 2K | O(N2\* log N) | Dynamic programming, [Binary Search](https://www.geeksforgeeks.org/binary-search/),[Sorting](https://www.geeksforgeeks.org/sorting-algorithms/),  Divide and Conquer |
| 10K | O(N2) | Dynamic programming, [Graph](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/), [Trees](https://www.geeksforgeeks.org/binary-tree-data-structure/), Constructive |
| 1M | O(N\* log N) | Sorting, Binary Search, Divide and Conquer |
| 100M | O(N), O(log N), O(1) | Constructive, [Mathematical,](https://www.geeksforgeeks.org/mathematical-algorithms/) [Greedy Algorithms](https://www.geeksforgeeks.org/greedy-algorithms-general-structure-and-applications/) |



**Euclid Algo**

**class** GFG {

    // extended Euclidean Algorithm

**public** **static** **int** gcd(**int** a, **int** b)

    {

**if** (a == 0)

**return** b;

**return** gcd(b % a, a);

    }

    // Driver code

**public** **static** **void** main(String[] args)

    {

**int** a = 10, b = 15, g;

          // Function call

        g = gcd(a, b);

        System.out.println("GCD(" + a + " , " + b

                           + ") = " + g);

        a = 35;

        b = 10;

        g = gcd(a, b);

        System.out.println("GCD(" + a + " , " + b

                           + ") = " + g);

        a = 31;

        b = 2;

        g = gcd(a, b);

        System.out.println("GCD(" + a + " , " + b

                           + ") = " + g);

    }

}

**Linear Search**

**class** GFG {

**public** **static** **int** search(**int** arr[], **int** x)

    {

**int** N = arr.length;

**for** (**int** i = 0; i < N; i++) {

**if** (arr[i] == x)

**return** i;

        }

**return** -1;

    }

    // Driver code

**public** **static** **void** main(String args[])

    {

**int** arr[] = { 2, 3, 4, 10, 40 };

**int** x = 10;

        // Function call

**int** result = search(arr, x);

**if** (result == -1)

            System.out.print(

                "Element is not present in array");

**else**

            System.out.print("Element is present at index "

                             + result);

    }

}