Asymptotic Notations

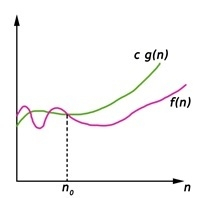
Asymptotic notations are used to represent the [**complexities of algorithms**](https://www.tutorialspoint.com/Algorithms-and-Complexities) **for asymptotic analysis.**

Big Oh Notation

It represent upper bound

We write f(n) = O(g(n)), If there are positive constants n0  and c,  f(n) always below c\*g(n). 0 ≤ f(n) ≤ c g(n), for all n ≥ n0}.

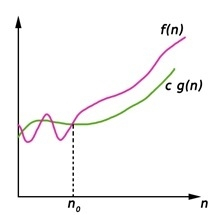
**Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order.



Big Omega Notation

We write f(n) = Ω(g(n)), If there are positive constants n0  and c such that f(n) always above c\*g(n). 0 ≤ c g(n) ≤ f(n), for all n ≥ n0}

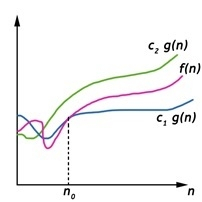
**Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted.



Big Theta Notation

We write f(n) = Θ(g(n)), If there are positive constants n0  and c1 and c2 f(n) always average point between c1\*g(n) and c2\*g(n). 0 ≤ c1 g(n) ≤ f(n) ≤ c2 g(n), for all n ≥ n0}

**Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending.



**Linear Search Algorithm**

Linear search is also called a sequential **search algorithm.** It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

1. Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(1) |
| **Average Case** | O(n) |
| **Worst Case** | O(n) |

Binary Search Algorithm

In this article, we will discuss the Binary Search Algorithm. Searching is the process of finding some particular element in the list. If the element is present in the list, then the process is called successful, and the process returns the location of that element. Otherwise, the search is called unsuccessful.

1. Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(1) |
| **Average Case** | O(logn) |
| **Worst Case** | O(logn) |

Data > A[mid], high= mid+1

Data < A [mid] low= mid-1

Data == A[mid]  return mid

**Bubble sort Algorithm**

Bubble short is majorly used where -

* complexity does not matter
* simple and shortcode is preferred

1. Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n) |
| **Average Case** | O(n2) |
| **Worst Case** | O(n2) |

**Insertion Sort Algorithm**

Insertion sort has various advantages such as -

* Simple implementation
* Efficient for small data sets
* Adaptive, i.e., it is appropriate for data sets that are already substantially sorted.

https://lh7-us.googleusercontent.com/vWLaqiRc6fDjOAo1iaAGjtqDY_6u4W8lOxNAptE3eSXP29ZFnGkKbxMDS5b9UiOGULkHAwzzL9NyiF6X1Q1UNgOOxEHwTk7dl2kaNdsP316RTWTzw1xBJDSrCBxB63zaECnV69qsFCQEmXDx9U8pemY

1. Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n) |
| **Average Case** | O(n2) |
| **Worst Case** | O(n2) |

Selection Sort Algorithm

1. Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n2) |
| **Average Case** | O(n2) |
| **Worst Case** | O(n2) |

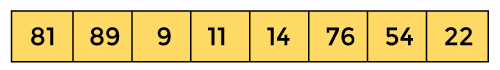
Heap Sort Algorithm

What is a heap?

A heap is a complete binary tree, and the binary tree is a tree in which the node can have the utmost two children. A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node, should be completely filled, and all the nodes should be left-justified.

What is heap sort?

Heapsort is a popular and efficient sorting algorithm. The concept of heap sort is to eliminate the elements one by one from the heap part of the list, and then insert them into the sorted part of the list.



1. Time Complexity

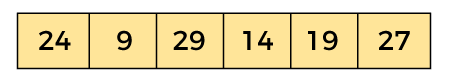
|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n logn) |
| **Average Case** | O(n logn) |
| **Worst Case** | O(n logn) |

Merge Sort Algorithm

. Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n\*logn) |
| **Average Case** | O(n\*logn) |
| **Worst Case** | O(n\*logn) |

Quick Sort Algorithm



### A[i] <= pivot

I++

### A[j] > pivot

j--

### Swap (i,j)

I← j

swap(j, pivot)

1. Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n\*logn) |
| **Average Case** | O(n\*logn) |
| **Worst Case** | O(n2) |