Sunbeam Institute of Information Technology

Pune and Karad

PreCAT

Module – Data Structures

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Algorithm Analysis

- Analysis is done to determine how much resources it require.
- Resources such as time or space
- There are two measures of doing analysis of any algorithm
  - Space Complexity
    - Unit space to store the data into the memory (Input space) and additional space to process the data (Auxiliary space)

    e.g. Algorithm to find sum of all array elements.

    int arr[n] – n units of input space

    sum, index, size – 3 units of auxiliary space

    Total space required = input space + auxiliary space = n + 3 = n units

  - Time Complexity
    - Unit time required to complete any algorithm
    - Approximate measure of time required to complete algorithm
    - Depends on loops in the algorithm
    - Also depends on some external factors like type of machine, no of processed running on machine.
    - That’s why we can not find exact time complexity.
  - Method used to calculate complexities, is “**Asymptotic Analysis**”

Asymptotic Analysis

- It is a mathematical way to calculate complexities of an algorithm.
- It is a study of change in performance of the algorithm, with the change in the order of inputs.
- It is not exact analysis
- Few mathematical notations are used to denote complexities.
- These notations are called as “Asymptotic notations” and are
  - Omega notation (Ω)
    - Represents lower bound of the running algorithm
    - It is used to indicate the best case complexity of an algorithm
  - Big – Oh notation (O)
    - Represents upper bound of the running algorithm
    - It is used to indicate the worst case complexity of an algorithm
  - Theta notation (ϴ)
    - Represents upper and lower bound of the running time of an algorithm (tight bound)
    - It is used to indicate the average case complexity of an algorithm

Time Complexity

```
Statement;
```

constant

```
for(i=0; i< n; i++)
{
        statements;
}
```

Linear

```
for(i=0; i< n; i++)
{
        for(j=0; j< n; j++)
        {
                statements;
        }

}
```

Quadratic

```
for(i=n; i>0; i/=2)
{
        statement
}
```

Logarithmic

Searching Algorithms : Time Complexity

Linear Search :

	No of Comparisons		Running Time	Time Complexity
Best Case	1	Key found at very first position	O(1)	O(1)
Average Case	n/2	Key found at in between position	O(n/2) = O(n)	O(n)
Worst Case	n	Key found at last position or not found	O(n)	O(n)

Binary Search :

	No of Comparisons		Running Time	Time Complexity
Best Case	1	Key found in very first iteration	O(1)	O(1)
Average Case	log n	Key found at non-leaf position	O(log n)	O(log n)
Worst Case	log n	if either key is not found or key is found at leaf position	O(log n)	O(log n)

Sorting Algorithms : Comparisons

- Selection sort algorithm is too simple, but performs poor and no optimization possible.
- Bubble sort can be improved to reduce number of iterations.
- Insertion sort performs well if number of elements are too less. Good if adding elements and resorting.
- Quick sort is stable if number of elements increased. However worst case performance is poor.
- Merge sort also perform good, but need extra auxiliary space.

Algorithm	Best Case	Average Case	Worst Case
Selection sort	O(n^2)	O(n^2)	O(n^2)
Bubble sort	O(n)	O(n^2)	O(n^2)
Insertion sort	O(n)	O(n^2)	O(n^2)
Merge sort	O(n log n)	O(n log n)	O(n log n)
Quick sort	O(n log n)	O(n log n)	O(n^2)

Thank you!

Devendra Dhande
<devendra.dhande@sunbeaminfo.com>