Algorithms and Complexity

Algorithm Complexity

Lecturer: Asst. Prof. Dr. Alaa Ahmed Abbood
Lecture 4.
Class 2nd .
Time: 8:30-10:30
Department: Businesses Information Technology (BIT)

Outline

Lecture 4

- **Algorithm Complexity**
- **Asymptotic Notations**
- **Definition of "big O“**
- **Standard Method to Prove Big-Oh with Examples**
- **Constant, linear, Logarithmic, and Quadratic time complexity**
- **Definition of "big Omega“ and "big Theta“**
- **Analysis of Algorithms**
- **Amortized Time Complexity**

Outline

Lecture 2

- Algorithm Complexity
- Asymptotic Notations
- Definition of "big O“
- Standard Method to Prove Big-Oh with Examples
- Constant, linear, Logarithmic, and Quadratic time complexity
- Definition of "big Omega“ and "big Theta“
- Analysis of Algorithms
- Amortized Time Complexity

1- **Algorithmic Complexity** :

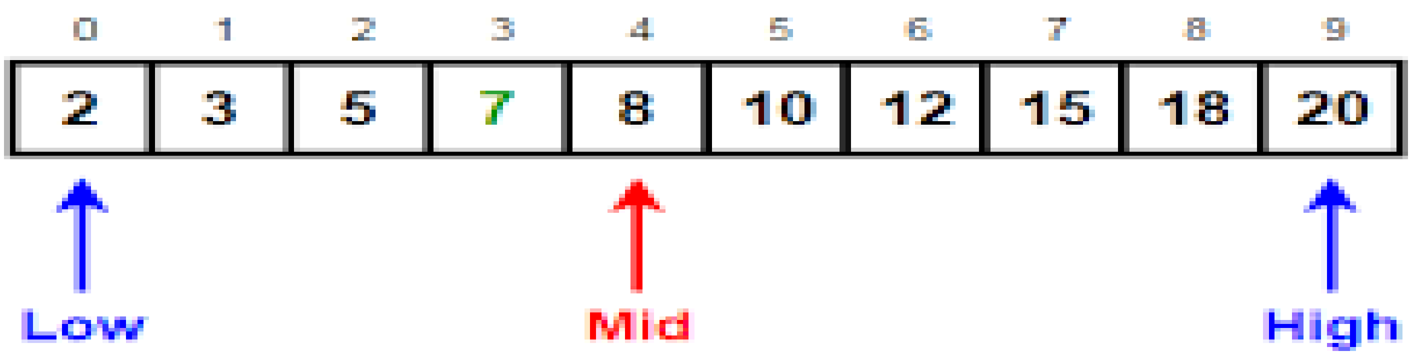
- Algorithmic complexity is concerned about how fast or slow particular algorithm performs.
- We define complexity as a numerical function T(n) - time versus the input size n.
- We want to define time taken by an algorithm without depending on the implementation details. But you agree that T(n) does depend on the implementation.

Algorithm Complexity. Cont...

- A given algorithm will take different amounts of time on the same inputs depending on such factors as: processor speed, instruction set, disk speed, brand of compiler and etc.
- The way around is to estimate efficiency of each algorithm asymptotically. We will measure time T(n) as the number of elementary "steps" (defined in any way), provided each such step takes constant time.

Algorithm Complexity. Cont...

- Let us consider two classical examples which search an element in a list.



Best case

Average case

Worse case

Algorithm Complexity. Cont...

- There are three cases

1. Best case (Omega notation Ω)
2. Average case (Theta notation Θ)
3. Worst case (Big O notation)

Constant Time: O(1)

- An algorithm is said to run in constant time if it requires the same amount of time regardless of the input size.

Examples:

- array: accessing any element.
- All the operations such as (+,- *, /, %, ˆ,= etc.)
- If – else statement.
- Any program statement.

Constant Time: O(1)

Examples: consider the following code :-

```
{
int n;  // O(1)
int sum; // O(1)

        {
        Sum = n*(n+1)/2 // O(1)

        }
Print sum // O(1)

}
```

Therefore the worst case of this code = O(1).

Linear Time: O(n)

Examples: consider the following code :-

```
{
int n;
int i;  // O(1)
i=0; // O(1)

        {
        for (i=0, i<n, n++) // O(n)

        }
Print i ; // O(1)

}
```

Big O= 1+n

O(n)

Linear Time: O(n)

Examples: consider the following code :-

```
{
int n;
int i;  // O(1)
i=0; // O(1)

        for (i=0, i<n, n++) {  //(n)


Print i } // O(1)
        for ( int j=0; j<n; j++) //(n)
        for (int k=0; k<n; k++) //(n)
        for ( intl=0; l<n; l++) //(n)


}
```

Big O= 1+n+ n*n*n

$O(n)= 1+n+n^3$

$O(n)=n^3$

Logarithmic Time: O(log n)

Examples: consider the following code :-

```
{
int n;
int i;  // O(1)
i=1; // O(1)

        for (i, i<n, i=i*2) {  // log (n)


Print i }
}
```

Big O= $1+\log_2 (n)$

Note: that only if the increment * or /

Logarithmic Time: O(log n)

Examples: consider the following code :-

```
{
int i, j;
i=j=0;  // O(1)

        for (i, i<n, i++) {  //(n)
        for ( j; j<n; j=j/3) //(log3 n)

Print i +j } // O(1)

}
```

Big O= $1+n * \log_3 n$

Big O= $n \log_3 n$

Logarithmic Time: O(log n)

Examples: consider the following code :-

```
{
int i, j, k;
i=j=0;  // O(1)

        for (i, i<n, i++) {  //(n)
        for ( j; j<n; j=j++) //(n)
         for ( k; j<n; k=k*2) log2 n

Print i +j +k} // O(1)

}
```

Big O= $1+n * n * \log_2 n$

O= $n^2 \log n$

Logarithmic Time: O(log n)

Examples: consider the following code :-

```
{
int i, j, k;
i=j=0;  // O(1)

        for (i, i<n/2, i++) {  //(n/2)
        for ( j; j<n; j=j/2) //log2 n
         for ( k; j<n; k=k*2) log2 n

Print i +j +k} // O(1)

}
```

$$\text{Big } O = n/2 * n * \log_2 n$$

$$O = n^* \log_2 n * \log_2 n$$

$$O = n^* \log_2 n^2$$

summary

- The term analysis of algorithms is used to describe approaches to the study of the performance of algorithms.

❖ In this course we will perform the following types of analysis:

- **The worst-case runtime complexity of the algorithm** is the function defined by the maximum number of steps taken on any instance of size $\boldsymbol{a}$.
- **The best-case runtime complexity of the algorithm** is the function defined by the minimum number of steps taken on any instance of size $\boldsymbol{a}$.
- **The average case runtime complexity of the algorithm** is the function defined by an average number of steps taken on any instance of size $\boldsymbol{a}$.

THANK YOU