



Analyzing Algorithm Control Structure

To analyze a programming code or algorithm, we must notice that each instruction affects the overall performance of the algorithm and therefore, each instruction must be analyzed separately to analyze overall performance. However, there are some algorithm control structures which are present in each programming code and have a specific asymptotic analysis.

Some Algorithm Control Structures are:

1. Sequencing
2. If-then-else
3. for loop
4. While loop

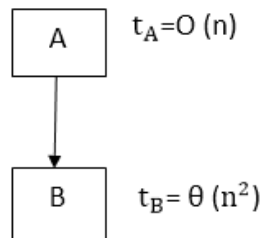
1. Sequencing:

Suppose our algorithm consists of two parts A and B. A takes time t_A and B takes time t_B for computation. The total computation " $t_A + t_B$ " is according to the sequence rule. According to maximum rule, this computation time is $(\max(t_A, t_B))$.

Example:

Suppose $t_A = O(n)$ and $t_B = \theta(n^2)$.

Then, the total computation time can be calculated as

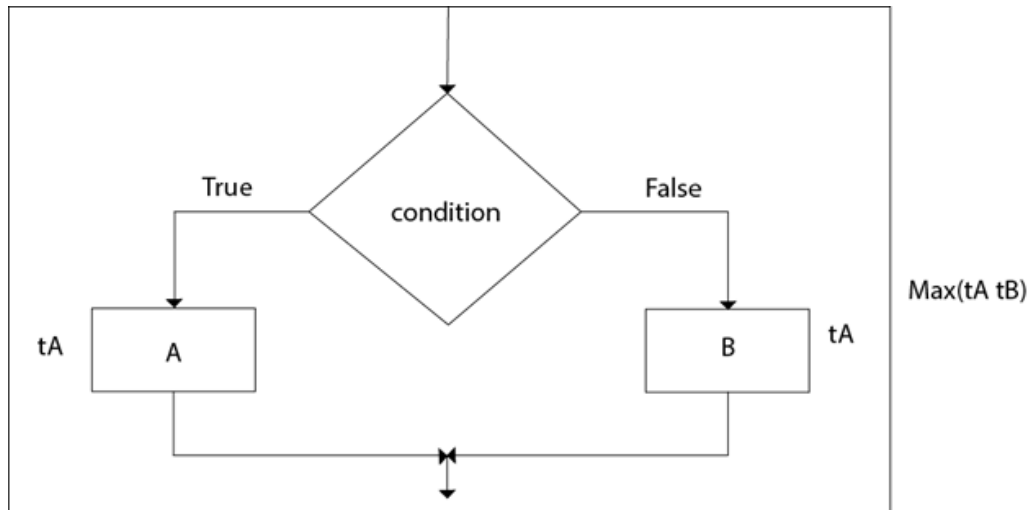


Computation Time = $t_A + t_B$

= $(\max(t_A, t_B))$

= $(\max(O(n), \theta(n^2))) = \theta(n^2)$

2. If-then-else:

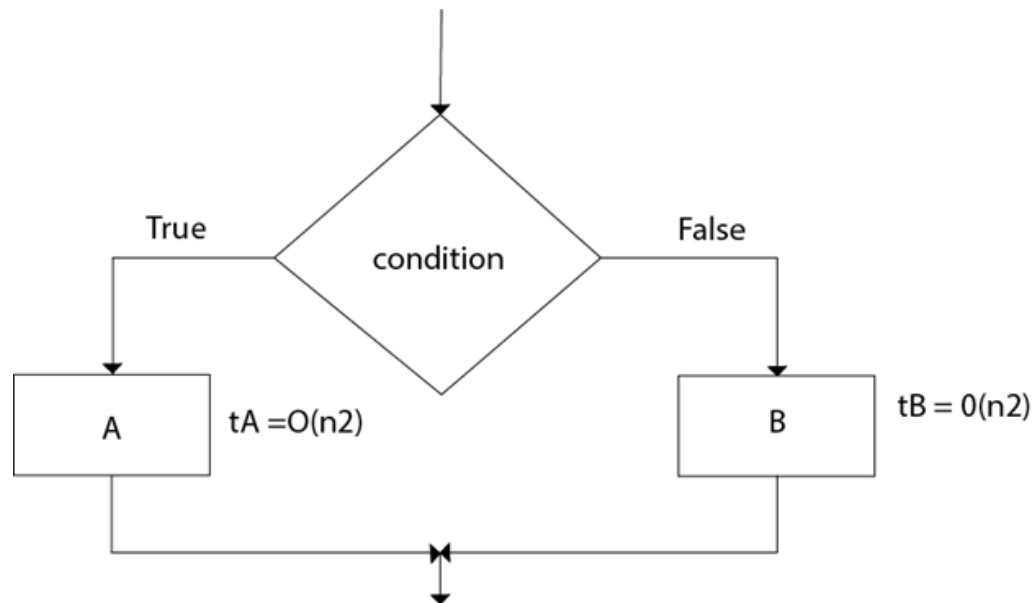


The total time computation is according to the condition rule-"if-then-else." According to the maximum rule, this computation time is $\max(t_A, t_B)$.

Example:

Suppose $t_A = O(n^2)$ and $t_B = O(n^2)$

Calculate the total computation time for the following:



```
Total Computation = (max (tA, tB))
                  = max (O (n2), θ (n2) = θ (n2)
```

3. For loop:

The general format of for loop is:

```
For (initialization; condition; updation)

Statement(s);
```

Complexity of for loop:

The outer loop executes N times. Every time the outer loop executes, the inner loop executes M times. As a result, the statements in the **inner** loop execute a total of N * M times. Thus, the total complexity for the two loops is O (N²)

Consider the following loop:

```
for i ← 1 to n
{
    P (i)
}
```

If the computation time t_i for (P_i) varies as a function of "i", then the total computation time for the loop is given not by a multiplication but by a sum i.e.

```
For i ← 1 to n
{
    P (i)
}
```

Takes $\sum_{i=1}^n t_i$ time, i.e. $\sum_{i=1}^n \theta(1) = \theta \sum_{i=1}^n \theta(n)$

If the algorithms consist of nested "for" loops, then the total computation time is

```
For i ← 1 to n
{
```



```

    For j ← 1 to n       $\sum_{i=1}^n \sum_{j=1}^n t_{ij}$ 
    {
        P (ij)
    }
}

```

Example:

Consider the following "for" loop, Calculate the total computation time for the following:

```

For i ← 2 to n-1
{
    For j ← 3 to i
    {
        Sum ← Sum+A [i] [j]
    }
}

```

Solution:

The total Computation time is:

$$\sum_{i=2}^{n-1} \sum_{j=3}^i t_{ij} = \sum_{i=2}^{n-1} \sum_{j=3}^i \theta(1)$$

$$\sum_{i=2}^{n-1} \theta(i)$$

$$= \theta\left(\sum_{i=2}^{n-1} i\right) = \theta\left(\frac{m^2}{2}\right) + \theta(m)$$

$$= \theta(m^2)$$

4. While loop:

The Simple technique for analyzing the loop is to determine the function of variable involved whose value decreases each time around. Secondly, for terminating the loop, it is necessary that value must be a positive integer. By keeping track of how many times the value of function decreases, one can obtain the number of repetition of the loop. The other approach for analyzing "while" loop is to treat them as recursive algorithms.

Algorithm:



```
1. [Initialize] Set k: =1, LOC: =1 and MAX: = DATA [1]
2. Repeat steps 3 and 4 while K≤N
3. if MAX<DATA [k],then:
    Set LOC: = K and MAX: = DATA [k]
4. Set k: = k+1
   [End of step 2 loop]
5. Write: LOC, MAX
6. EXIT
```

Example:

The running time of algorithm array Max of computing the maximum element in an array of n integer is $O(n)$.

Solution:

```
array Max (A, n)
1. Current max ← A [0]
2. For i ← 1 to n-1
3. do if current max < A [i]
4. then current max ← A [i]
5. return current max.
```

The number of primitive operation $t(n)$ executed by this algorithm is at least.

```
2 + 1 + n + 4 (n-1) + 1 = 5n
2 + 1 + n + 6 (n-1) + 1 = 7n-2
```

The best case $T(n) = 5n$ occurs when $A[0]$ is the maximum element. The worst case $T(n) = 7n-2$ occurs when element are sorted in increasing order.

We may, therefore, apply the big-Oh definition with $c=7$ and $n_0=1$ and conclude the running time of this is $O(n)$.

[< prev](#)[next >](#)



Engineering jobs in japan

Ad We support your VISA application pro housing, and learning Japanese and etc.

Human Resocia

[Learn more](#)

Please Share



Join Javatpoint Test Series

Placement Papers	AMCAT	Bank PO/Clerk	GATE
TCS	eLitmas	UPSSSC	NEET
HCL	Java	Government Exams	CAT
Infosys	Python	SSC	Railway
IBM	C Programming	Civil Services	CTET
Accenture	Networking	SBI	IIT JEE



Learn Latest Tutorials





Hive



Jenkins



Pytorch



QA



Agile



JIRA



Tableau

Preparation



Aptitude



Reasoning



Verbal A.



Interview



Company

Trending Technologies



AI



AWS



Selenium



IoT



Cloud



Hadoop



ReactJS



React Native
























Node.js



D. Science



B.Tech / MCA

 DBMS	 DS	 DAA	 OS	 C. Network
 Compiler D.	 COA	 D. Math.	 E. Hacking	 C. Graphics
 Software E.	 Web Tech.	 Cyber Sec.	 Automata	 C
 C++	 Java	 .Net	 Python	 Programs
 Control S.				



