Time Complexity

1. Print 1D array on console

```
void print1DArray(int arr[], int n){
        for(int i =  0 ; i < n ; i++)
                sysout(arr[i]);
}
```

No of iterations = n

time $\propto$ No. of iterations

Time $\propto$ n

Time Complexity = T(n) = O(n)

2. Print 2D Array on console

```
void print2DArray(int arr[][], int m, int n){
        for(int i  = 0  ; i < m ; i++){
                for(int j = 0 ; j < n ; j++)
                        sysout(arr[i][j]);
        }
}
```

No of iterations of outer loop = m

No. of iterations of inner loop = n

Total iterations = m*n

Time $\propto$ m*n

Time Complexity = T(m,n) = O(m*n)

$\therefore$ if m==n, Time Complexity = T(n) = O(n^2)

Time Complexity

3. add two numbers

```
int addition(int num1, int num2){
        return num1 + num2;
}
```

Time requirement of this algorithm is not dependent on values of num1 & num2

Means, this algorithm is going to take constant amount of time

Time complexity = T(n) = O(1)

4. print table of given number

```
void printTable(int num){
        for(int i = 1 ; i <= 10  ; i++)
                sysout(i * num);
}
```

This loop is going to iterate 10 times for any value of num.

constant time requirement

Time complexity = T(n) = O(1)

5. Print Binary of given Decimal

2	9	
	4	1
	2	0
	1	0
		1

$(9)_{10} = (1001)_2$

```
void printBinary(int n){
        while(n > 0){
                sysout(n % 2);
                n = n / 2;
        }
}
```

n	n > 0	rem
9	T	1
4	T	0
2	T	0
1	T	1
0	F	

$n = 9, 4, 2, 1, 0$

$n = n, n/2, n/4, n/8 \dots\dots$

$= \frac{n}{2^0}, \frac{n}{2^1}, \frac{n}{2^2}, \frac{n}{2^3} \dots\dots \frac{n}{2^{itr}}$

for n=1, last time condition will be true

$\frac{n}{2^{itr}} = 1$

$n = 2^{itr}$

$itr \log 2 = \log n$

$itr = \frac{\log n}{\log 2}$

$Time \propto itr$

$Time \propto \frac{\log n}{\log 2}$

$T(n) = O(\log n)$

Time Complexity

Time Complexities :

O(1), O(lon n), O(n), O(n log n), O(n^2), O(n^3), O(2^n), ...

Modifiaction : '+' or '-' : in terms of n

Modifiaction : '*' or '/' : in terms of log n

```
for(i=0; i<n; i++)        →  O(n)
for(i=n; i>0; i--)        →  O(n)
for(i=0; i<20; i++)       →  O(1)
for(i=n; i>0; i/=2)       →  O(log n)
for(i=1; i<n; i*=2)       →  O(log n)

for( i=0; i<n; i++)       →  O(n^2)
   for(j=0; j<n; j++)

for( i=0; i<n; i++);      →  O(n)
for(j=0; j<n; j++);
```

Space Complexity

Total = Input space + Auxillary space

(space required to store actual input)

(space required to store variables which are needed to process actual input)

Find sum of array elements

```
int sumofArray(int arr[], int size){
    int sum = 0;
    for(int i = 0 ; i < size ; i++)
        sum += arr[i];
    return sum;
}
```

Input space = Input variable = arr = n

Auxillary space = Processing variables = size, i, sum = 3

Total space = Input space + Auxillary space

= n + 3

space ∝ total space

space ∝ n+3

Space complexity = S(n) = O(n) ∵ n >>>

$\boxed{\text{Auxillary space complex} = O(1)}$

Linear Search

for seraching and sorting algorithms
time is directly proportional to number of comparisions

1. Best case - if key is found in intial locations - O(1)
2. Average case - if key is found at middle of array - O(n)
3. Worst case - if key is found at last locations
   - if key is not found - O(n)

Binary Search

1. Best case - if key is found at few initial levels - O(1)
2. Average case - if key is found at middle levels - O(log n)
3. Worst case - if key is found at last few levels
   - if key is not found - O(log n)

Algorithm Solving Approches

Iterative

loops are used

```
int factorial(int num){
    int fact = 1;
    for(int i = 1 ; i <= num ; i++)
        fact *= i;
    return fact;
}
```

Time is proportional to number of iterations

T(n) = O(n)

Recursive

Recursion is used

Formula : n! = n * (n-1)!

Terminating condition : 1!=1

```
int recFactorial(int num){
    if(num == 1)
        return 1;
    return num * recFactorial(num - 1);
}
```

120

recFactorial(5) 5 * 24
recFactorial(4) 4 * 6
recFactorial(3) 3 * 2
recFactorial(2) 2 * 1
recFactorial(1) 1

Time is proportional to number of recursive function calls

T(n) = O(n)

Binary Search Recursive

	0	1	2	3	4	5	6	7	8
arr	11	22	33	44	55	66	77	88	99
					1	3	2		

Key = 66

5

BS(arr, 66, 0, 8) m = 4

5

BS(arr, 66, 5, 8) m = 6

5

BS(arr, 66, 5, 5) m = 5

Selection Sort

No. of comparisions $= 1 + 2 + 3 + \cdots + n$

$$= \frac{n(n+1)}{2}$$

$$= \frac{n^2 + n}{2}$$

$$\text{Time} \propto \frac{n^2 + n}{2}$$

$$\text{Time} \propto n^2 + n$$

$$\text{Time} \propto n^2$$

In case of mathematical polynomial only consider term which hase degree in power because it is highest growing term

n	n^2
1	1
10	100
100	10 000

$$\left.\begin{array}{r}\text{Best case}\\ \text{Average case}\\ \text{Worst case}\end{array}\right\} \rightarrow T(n) = O(n^2)$$

Degree of polynomial — highest power of variable