Time and Space Complexity

Selvamani Kannan

Agenda

- Algorithm analysis
  - Time
  - Space
- String

Analysis of Algorithm

- What is algorithm
- Need for analysis of algorithm
- Goal
- Running **time analysis**
  - How time increases when input size increases?
    - Depends on input creating array
  - Sample input types
    - Size of the array
    - Total number of elements in matrix

Time complexity

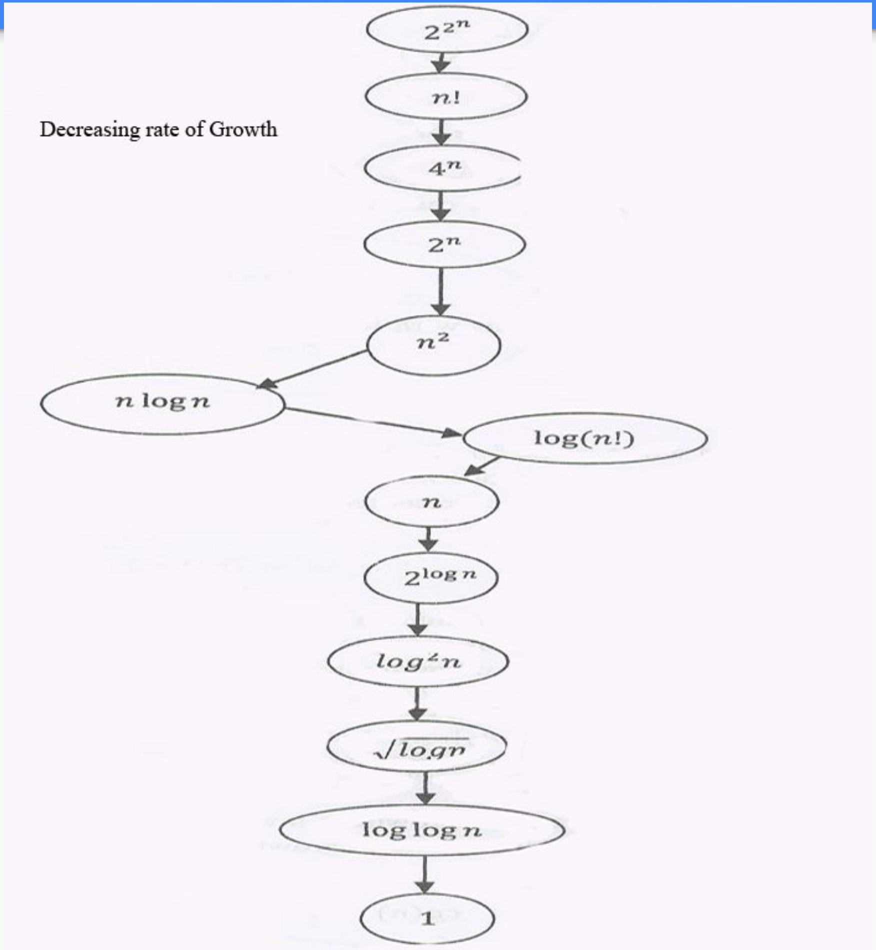
- How to compare?
  - Execution time - 1 min or 2min
  - No of statements gets executed -> lines of code
  - Ideal solution
    - Input n for a function f(n)
    - Independent of machine and statements
  - Rate of growth
  - Approximation
    - Buy new car and bicycle

$$\begin{aligned} \text{Total Cost} &= \text{cost\_of\_car} + \text{cost\_of\_bicycle} \\ \text{Total Cost} &\approx \text{cost\_of\_car (approximation)} \end{aligned}$$

  - Function f(n) -> for set of n values time taken

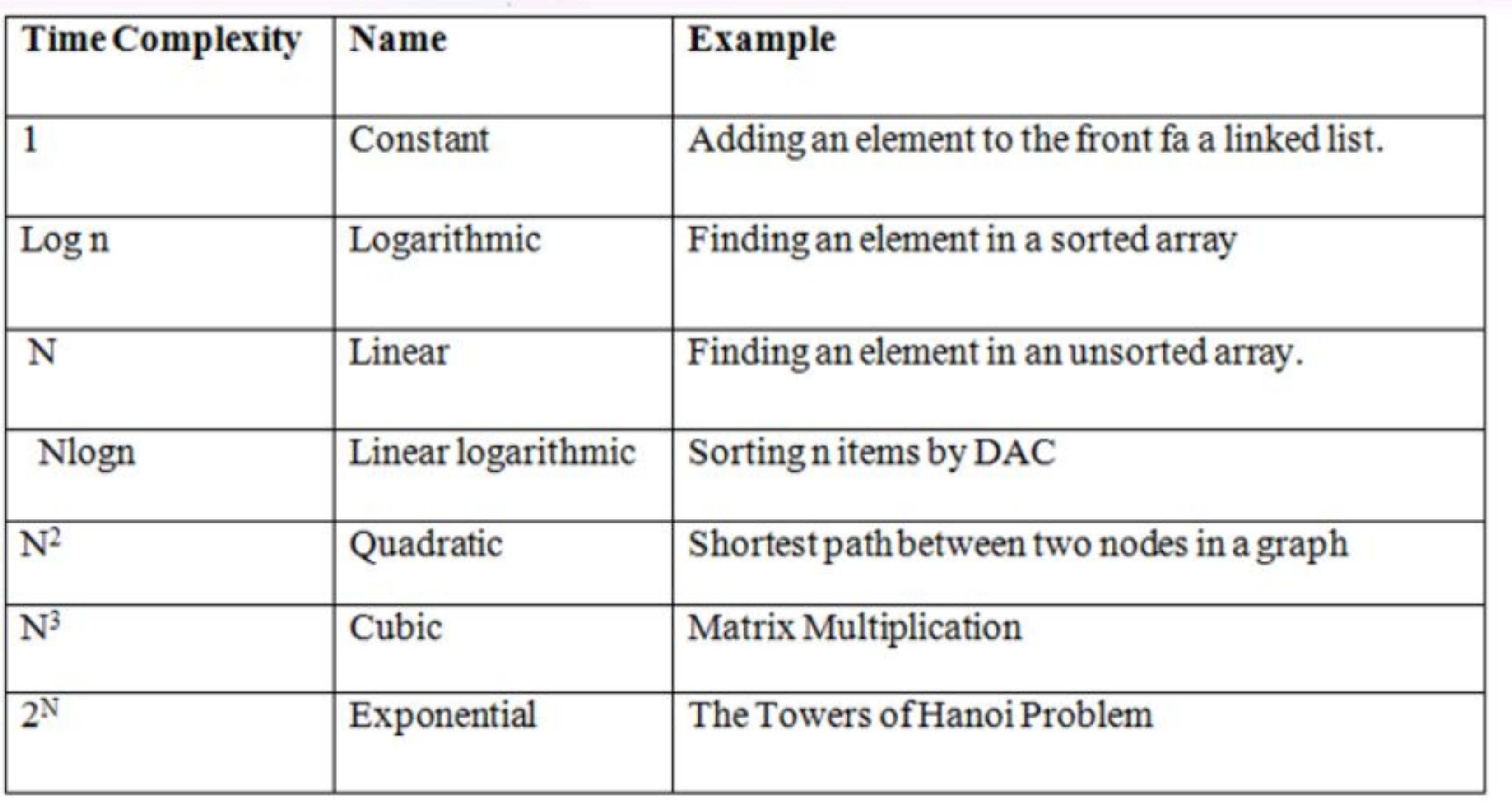
$$n^4 + 2n^2 + 100n + 500 \approx n^4$$

Decreasing rate of growth



Decreasing rate of Growth

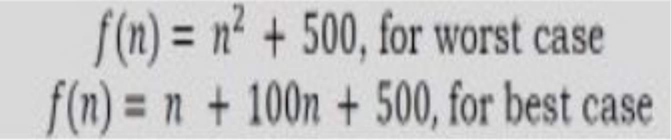
Decreasing rate of growth



Time Complexity	Name	Example
1	Constant	Adding an element to the front fa a linked list.
Log n	Logarithmic	Finding an element in a sorted array
N	Linear	Finding an element in an unsorted array.
Nlogn	Linear logarithmic	Sorting n items by DAC
N^2	Quadratic	Shortest path between two nodes in a graph
N^3	Cubic	Matrix Multiplication
2^N	Exponential	The Towers of Hanoi Problem

Types of analysis

- Worst case (Big O)
  - Slow
  - 10 days to finish a work
- Best case (Big Ω - Omega)
  - Fastest
  - 2 day to finish a work
- Average case (Big Θ - Theta)
  - Medium (Tight)
  - assume input is random
- **Lower Bound <= Average Time <= Upper Bound**

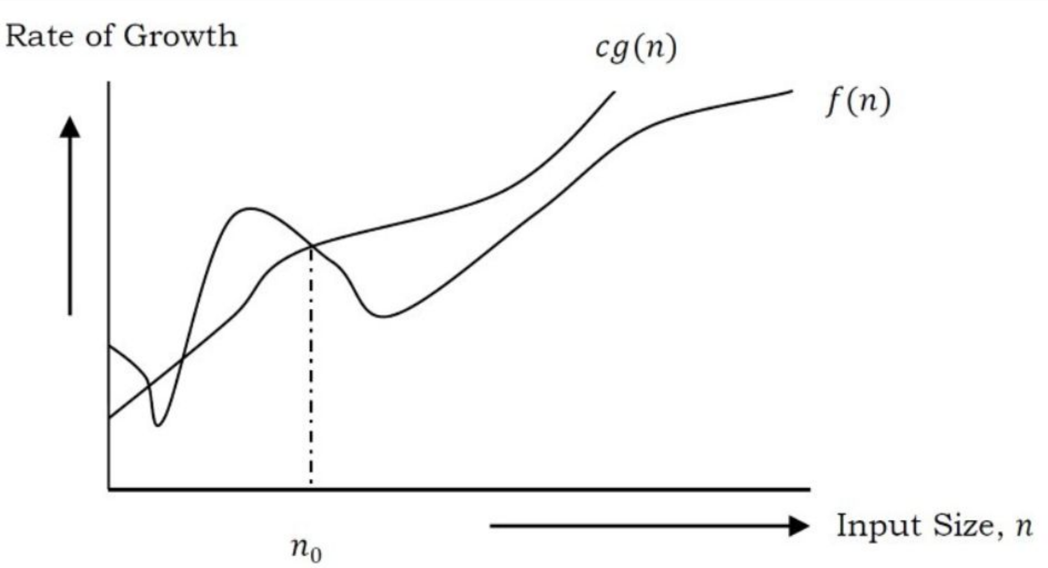


$$f(n) = n^2 + 500, \text{ for worst case}$$

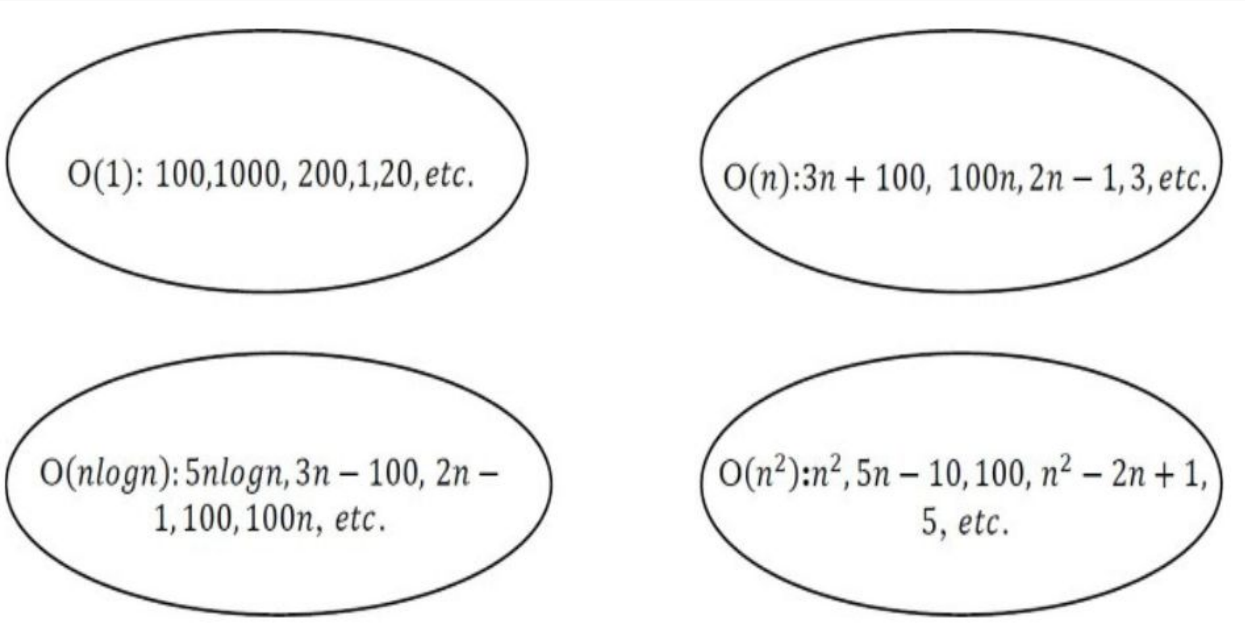
$$f(n) = n + 100n + 500, \text{ for best case}$$

Asymptotic Notation Big O Worst Case

- Big - O Notation Upper bound
- O(g(n)) = {f(n): there exist positive constants c and n0 such that 0 ≤ f(n) ≤ cg(n) for all n > n0} - n4 + 100n2 + 10n + 50 — g(n) = n4



Asymptotic Notation Big O Worst Case



O(1): $100, 1000, 200, 1, 20, etc.$

O(n):$3n + 100,\ 100n, 2n - 1, 3, etc.$

O$(nlogn)$:$5nlogn, 3n - 100,\ 2n - 1, 100, 100n,\ etc.$

O(n^2):$n^2, 5n - 10, 100,\ n^2 - 2n + 1,\ 5,\ etc.$

Find upper bound for $f(n) = 3n + 8$

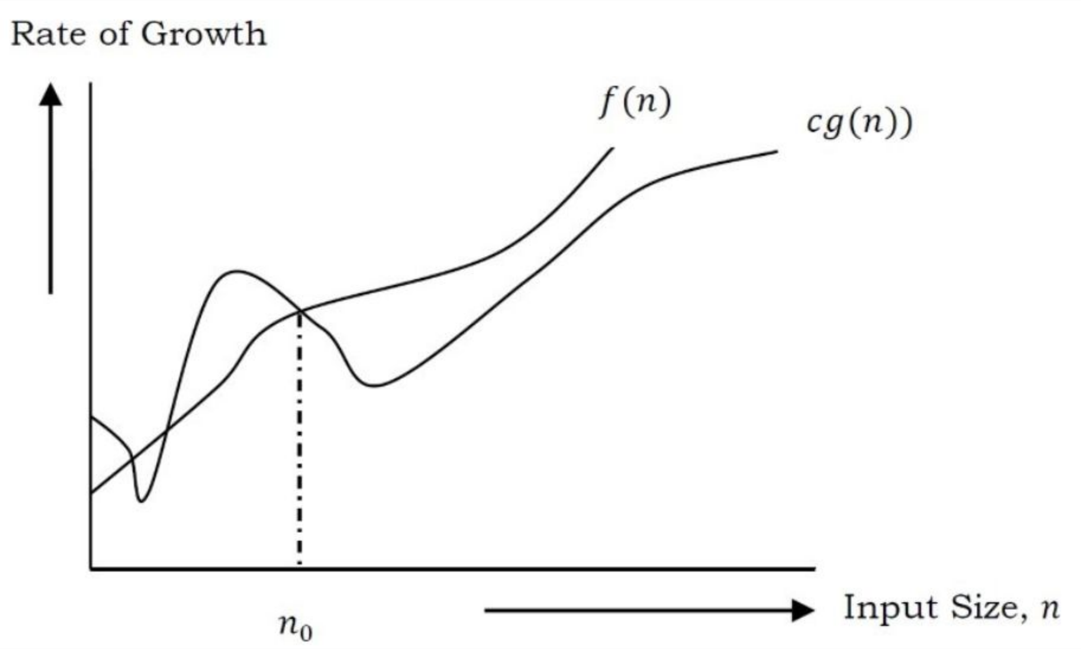
- $f(n) = 3n+8$
- Apply $n = 1$, $3(1)+8 = 11$
- $n=2$ $3(2)+8 = 14$
- $n=3$ $3(3)+8 = 17$
- $n = 4$ $3(4)+8 = 20$
- $n = 5$ $3(5)+8 = 23$

- $3n + 8 \leq 4n$, for all $n \geq 8$
  $3n + 8 = O(n)$ with $c = 4$ and $n0 = 8$

- $3n + 8 \leq 8n$, for all $n \geq 2$
  $3n + 8 = O(n)$ with $c = 8$ and $n0 = 2$

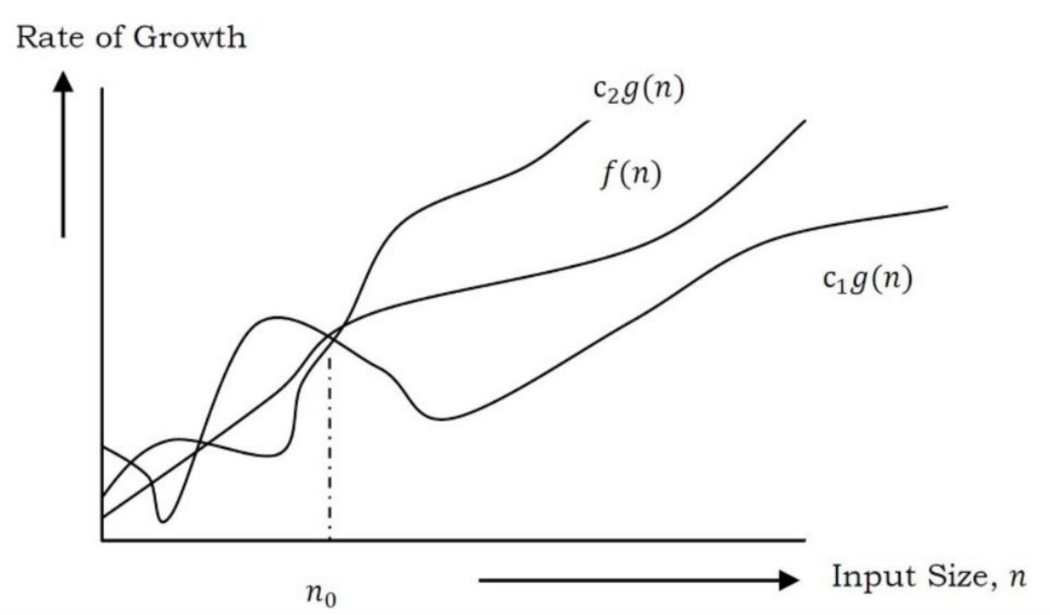
Asymptotic Notation Omega Best Case Ω

- Omega Notation Lower bound
- Ω(g(n)) = {f(n): there exist positive constants c and n0 such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0} - f(n) = 100n2 + 10n + 50, g(n) is Ω(n2).



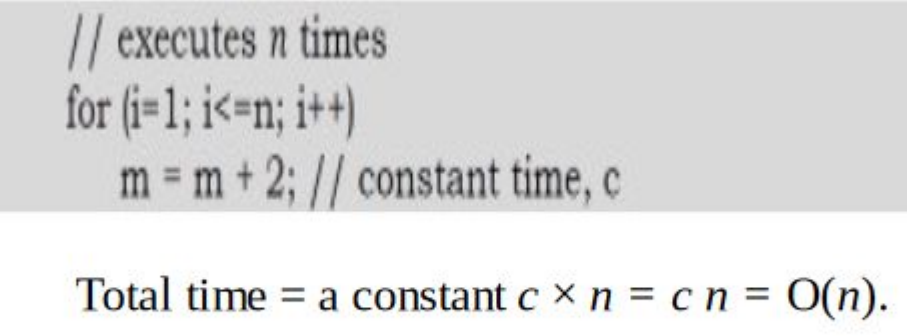
Asymptotic Notation Theta Average Case Θ

- Theta Order Function
- Θ(g(n)) = {f(n): there exist positive constants c1,c2 and n0 such that 0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) for all n ≥ n0}



Guidelines for Asymptotic Notation

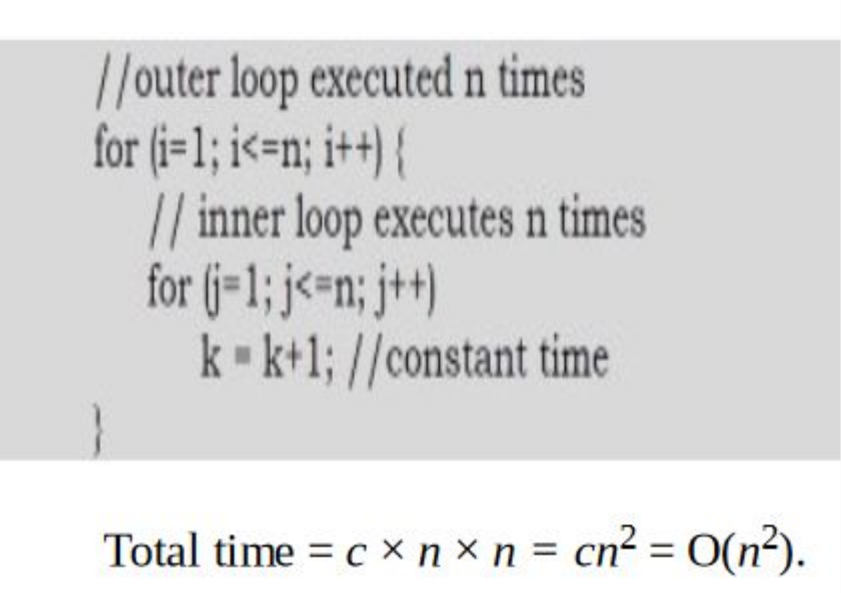
- Loops



```
// executes n times
for (i=1; i<=n; i++)
    m = m + 2; // constant time, c
```

Total time = a constant $c \times n = c\, n = O(n)$.

- Nested Loops

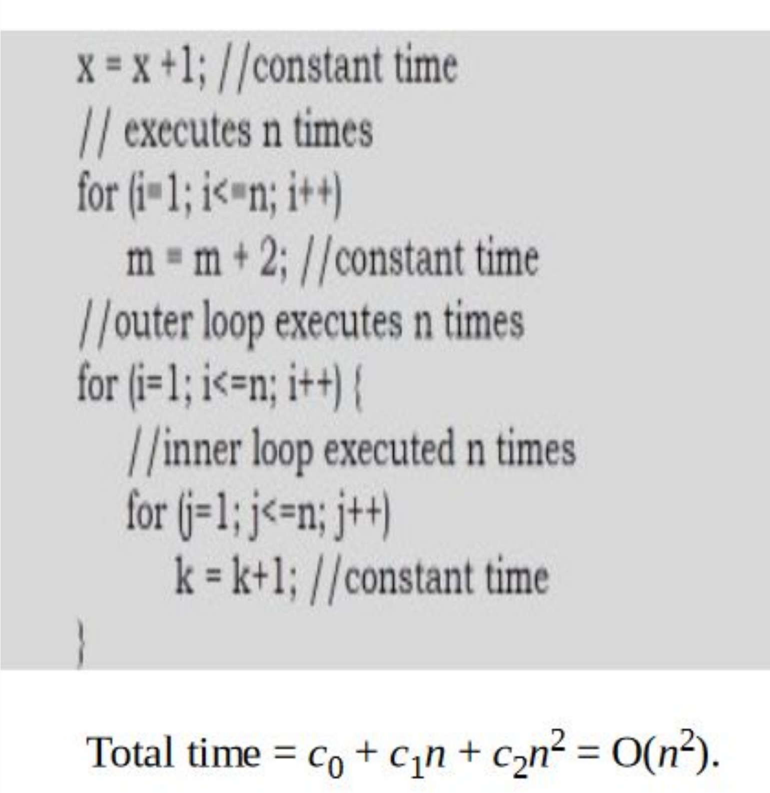


```
//outer loop executed n times
for (i=1; i<=n; i++) {
    // inner loop executes n times
    for (j=1; j<=n; j++)
        k = k+1; //constant time
}
```

Total time $= c \times n \times n = cn^2 = O(n^2)$.

Guidelines for Asymptotic Notation

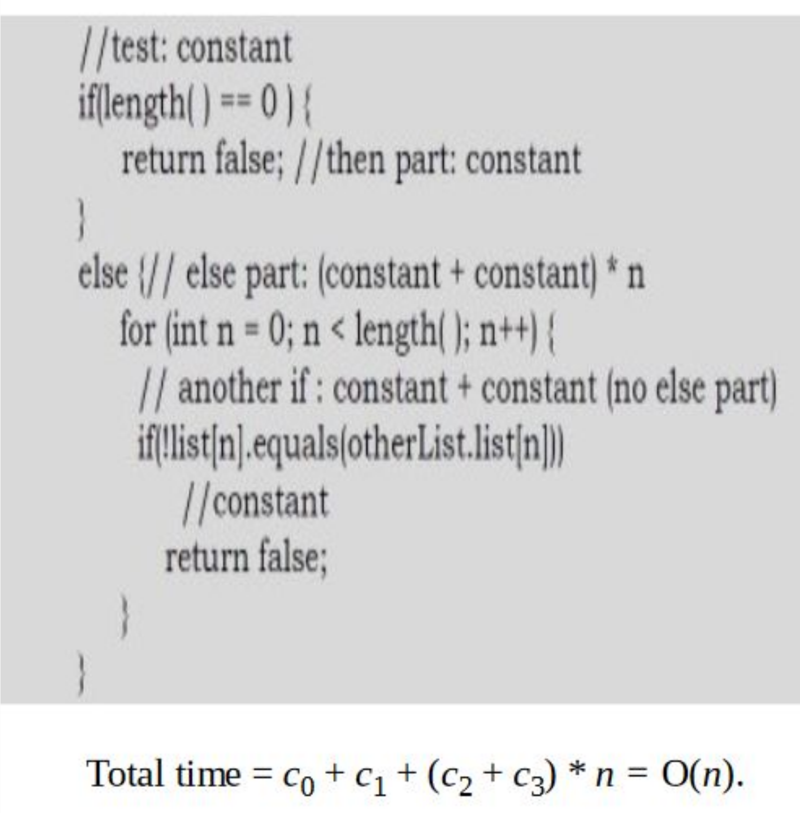
- ➢ Consecutive statement



```
x = x +1; //constant time
// executes n times
for (i=1; i<=n; i++)
    m = m + 2; //constant time
//outer loop executes n times
for (i=1; i<=n; i++) {
    //inner loop executed n times
    for (j=1; j<=n; j++)
        k = k+1; //constant time
}
```

Total time = $c_0 + c_1 n + c_2 n^2 = O(n^2)$.

- ➢ If else statement

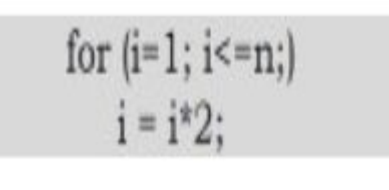


```
//test: constant
if(length( ) == 0 ) {
    return false; //then part: constant
}
else {// else part: (constant + constant) * n
    for (int n = 0; n < length( ); n++) {
      // another if : constant + constant (no else part)
      if(!list[n].equals(otherList.list[n]))
          //constant
        return false;
    }
}
```

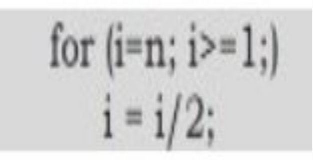
Total time = $c_0 + c_1 + (c_2 + c_3) * n = O(n)$.

Guidelines for Asymptotic Notation

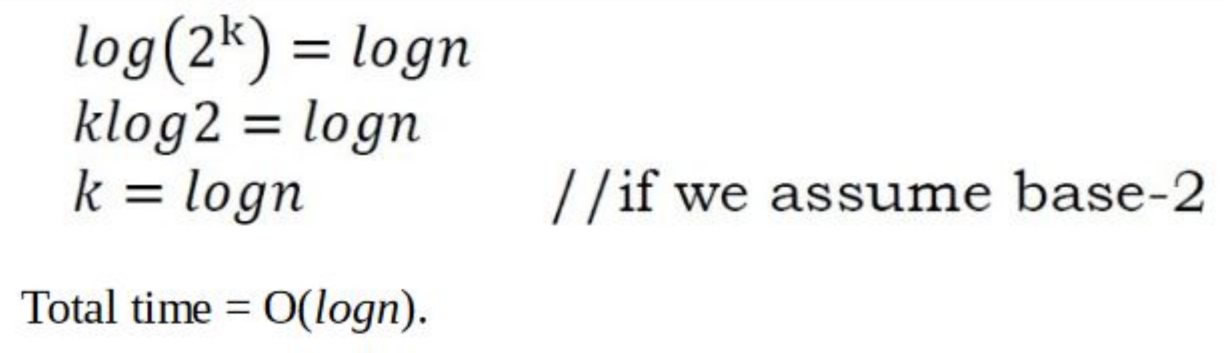
- Logarithmic time complexity



```
for (i=1; i<=n;)
    i = i*2;
```



```
for (i=n; i>=1;)
    i = i/2;
```



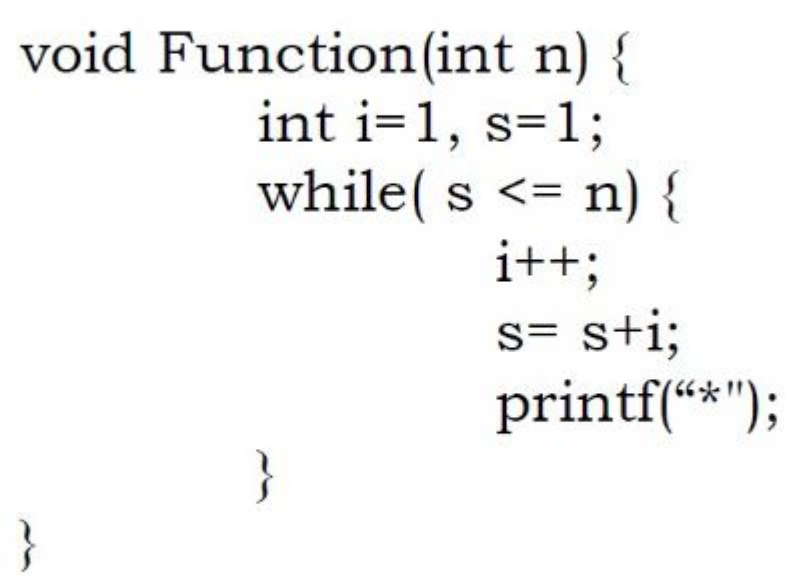
$$log(2^k) = logn$$

$$klog2 = logn$$

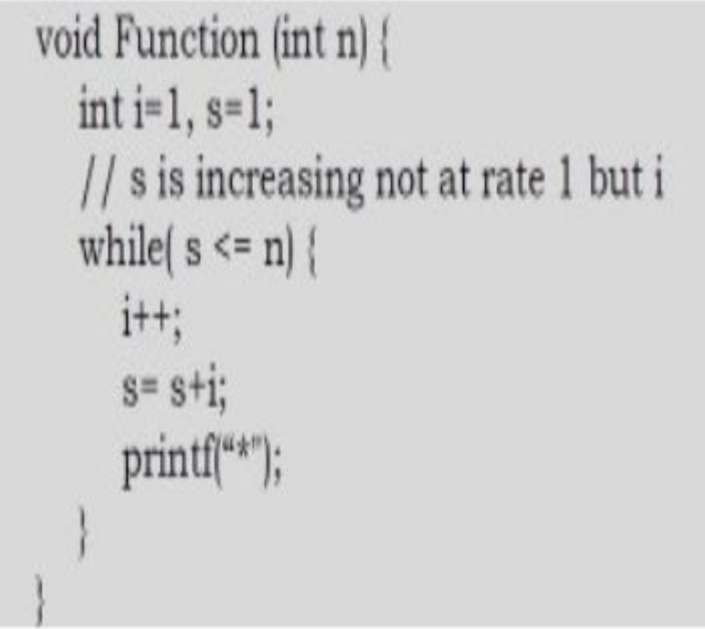
$$k = logn \qquad //\text{if we assume base-2}$$

Total time = $O(logn)$.

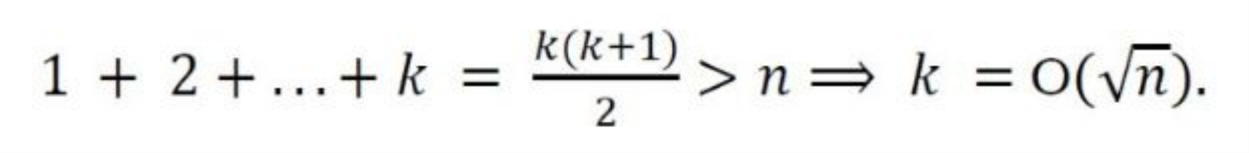
Find time complexity



```
void Function(int n) {
        int i=1, s=1;
        while( s <= n) {
                i++;
                s= s+i;
                printf("*");
        }
}
```

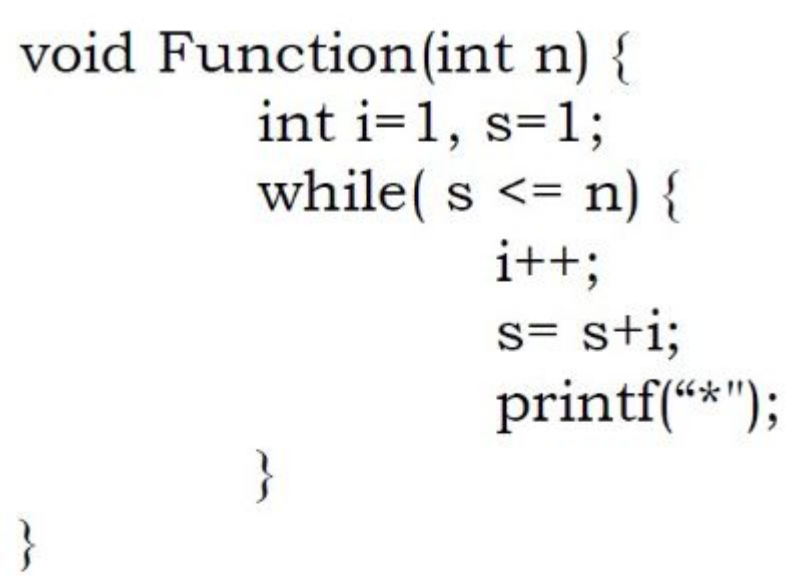


```
void Function (int n) {
    int i=1, s=1;
    // s is increasing not at rate 1 but i
    while( s <= n) {
        i++;
        s= s+i;
        printf("*");
    }
}
```

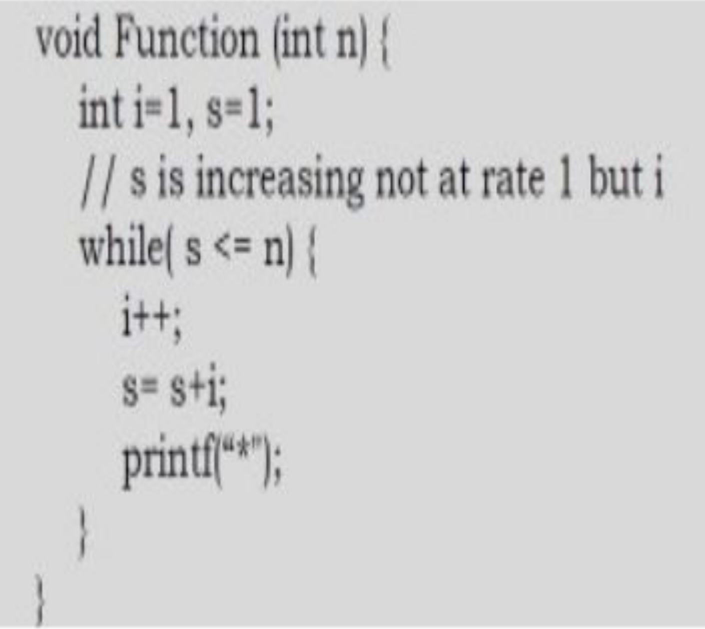


$$1 + 2 + \ldots + k = \frac{k(k+1)}{2} > n \Rightarrow k = O(\sqrt{n}).$$

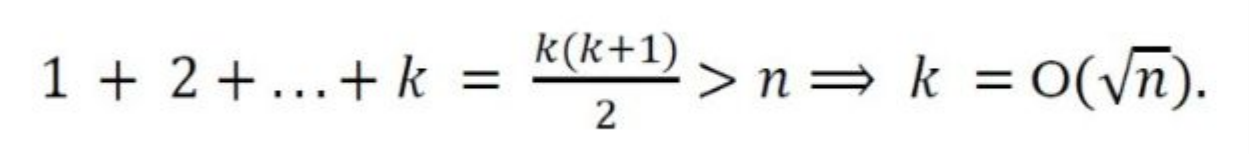
Find time complexity - 1



```
void Function(int n) {
        int i=1, s=1;
        while( s <= n) {
                i++;
                s= s+i;
                printf(“*");
        }
}
```

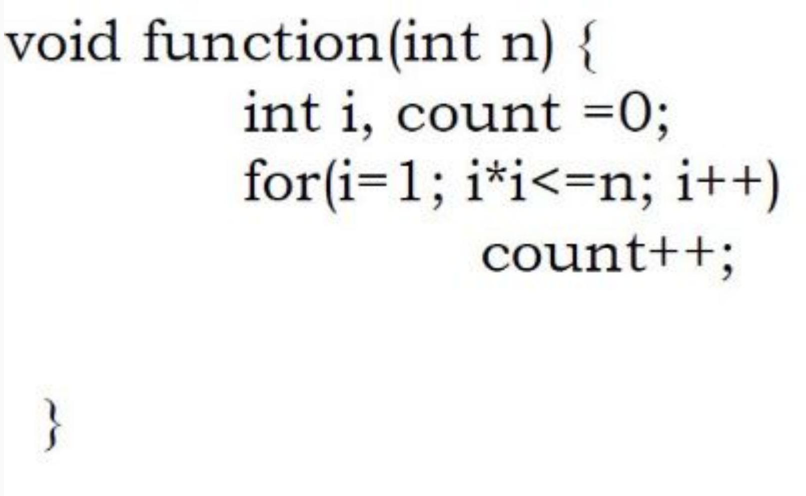


```
void Function (int n) {
    int i=1, s=1;
    // s is increasing not at rate 1 but i
    while( s <= n) {
        i++;
        s= s+i;
        printf(“*");
    }
}
```



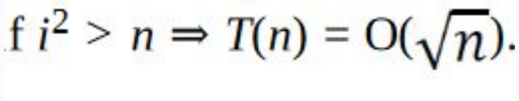
$$1 + 2 + \ldots + k = \frac{k(k+1)}{2} > n \Rightarrow k = O(\sqrt{n}).$$

Find time complexity -2



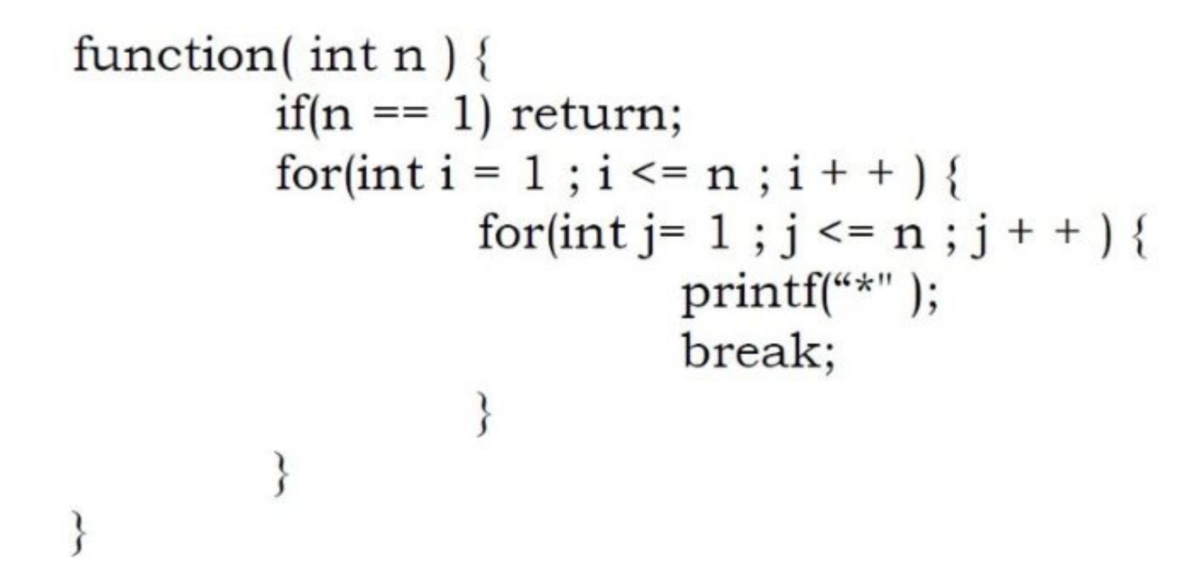
```
void function(int n) {
            int i, count =0;
            for(i=1; i*i<=n; i++)
                        count++;


  }
```



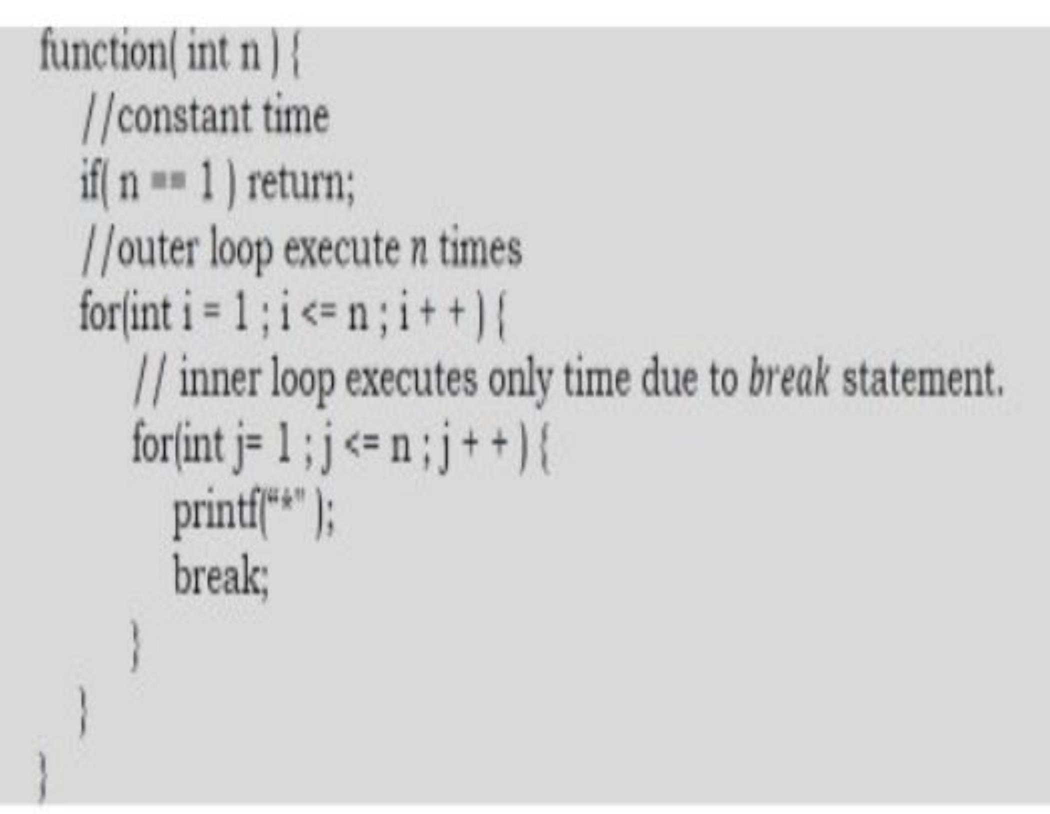
f $i^2 > n \Rightarrow T(n) = O(\sqrt{n})$.

Find time complexity -3



```
function( int n ) {
            if(n == 1) return;
            for(int i = 1 ; i <= n ; i + + ) {
                        for(int j= 1 ; j <= n ; j + + ) {
                                    printf(“*" );
                                    break;
                        }
            }
}
```

Find time complexity -3



```
function( int n ) {
    //constant time
    if( n == 1 ) return;
    //outer loop execute n times
    for(int i = 1 ; i <= n ; i + + ) {
        // inner loop executes only time due to break statement.
        for(int j= 1 ; j <= n ; j + + ) {
            printf("*" );
            break;
        }
    }
}
```

O(N)

Strings

-> Object that represents sequence of character (char)

-> Immutable, fixed length

String name= “Newton School”

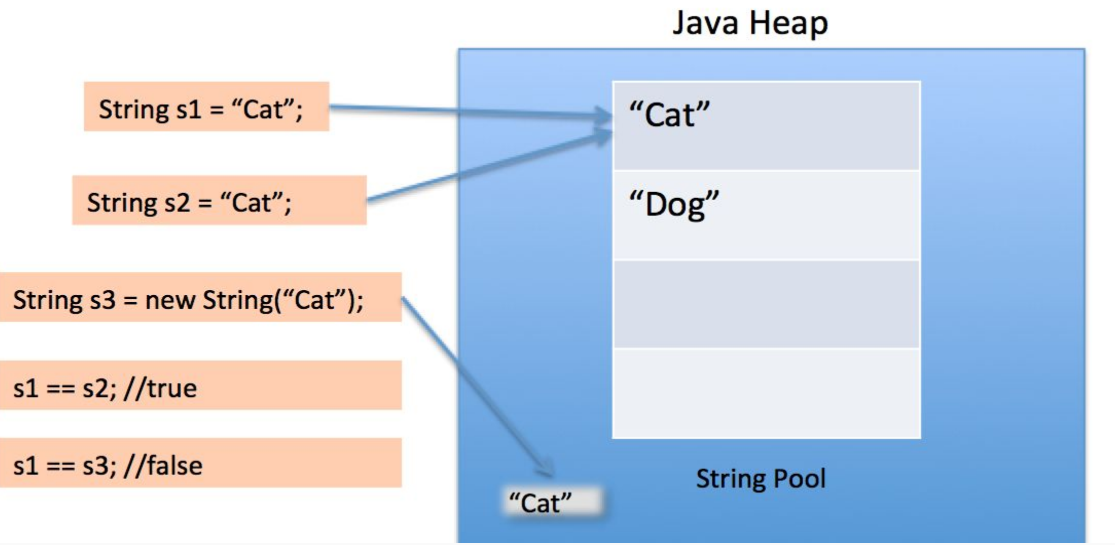


Image src: google

Mutable Strings

- String Buffer - Thread Safe
- String Builder - Not Thread safe

```
StringBuffer sbf = new StringBuffer("java");
sbf.append("122");
System.out.println(sbf);
```

Questions

Thank you!!!