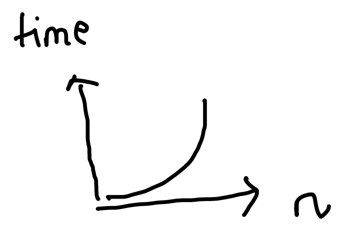
## Algorithm Analysis

The analysis of an algorithm is done in terms of the **time it takes** ( time complexity ) and the **space it consumes** (space complexity) to perform its operation.

sorting → Bubble, Insertion, Selection $\}$ $O(n^2)$

sorting → $\boxed{\text{sort()}}$ → $n\log_2 n$




search → linear ⇒ $n$

search → binary ⇒ $\log_2 n$ ✓

The analysis of an algorithm is always done in terms of the **input size**. Moreover, we mostly interest in the **worst-case analysis** and use the **Big-O notation** to report the worst-case time and space complexity of an algorithm.

$\Omega$ $\Theta$ $O$ ✓

To analyze the **time complexity** of an algorithm, we've to first identify algorithm type

- Iterative ( contains only loops ) * #iterations * time spent in each it
- recursive ( contains recursive calls ) # rec. calls * time spent in each call
- neither iterative nor recursive - mostly take **constant** time i.e. **O(1)**

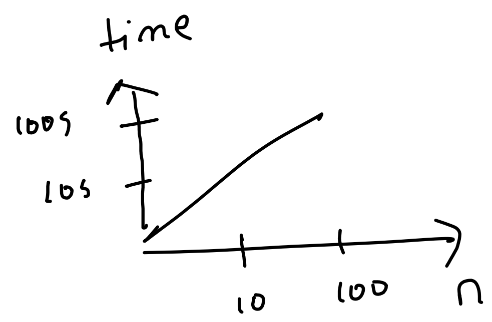
To analyze the **space complexity** of an algorithm, you've to analyze how much **extra** space or **auxiliary** space the algorithm consumes with respect to the size of the input.

## Time Complexity Analysis of Iterative Algorithms

```
void f(int n) {
    for(int i=1; i<=n; i++) {
        cout << "*" << " ";
    }
}
```

$\underbrace{i=1 \quad i<=n \quad i++}$

n times




const

$\underbrace{n * \text{const}} = O(n)$

$\overbrace{i=1;\ i<=n;\ i++}^{n \text{ times}}$

$\overbrace{j=1;\ j<=n;\ j++}^{n \text{ times}}$

```
void f(int n) {
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) {
            cout << "*";
        }
    }
}
```

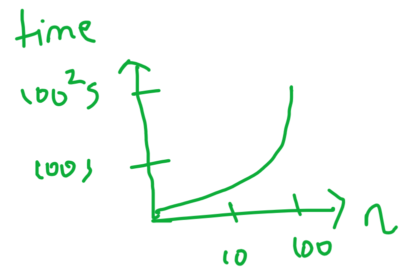
$\boxed{n \cdot \text{const}}$

const

$n * (n * \text{const})$

$= n^2 \cdot \text{const}$

$\sim O(n^2)$




```
void f(int n) {
    int i = 1;
    int s = 1;
    while(s < n) {
        i++;
        s += i;
    }
}
```

const

| | | |
|---|---|---|
| i = 1 | s = 1 | < n ? true — 1 |
| i = 2 | s = 3 | < n ? true — 2 |
| i = 3 | s = 6 | < n ? true — 3 |
| i = 4 | s = 10 | < n ? true — 4 |
| i = 5 | s = 15 | |
| ⋮ | ⋮ | |

< n ? true — $K^{th}$ time

$\sqrt{n} \cdot \text{const}$

$\hookrightarrow O(\sqrt{n})$

$\underbrace{i = K}$ $\boxed{S = ?}$ $\underbrace{1+2+\dots+K} \to \frac{K(K+1)}{2} < n$

$\frac{K(K+1)}{2} \cancel{<} n$

false

$\boxed{(K-1) \text{ times}}$

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

? $(\sqrt{n}-1)$ times

$\sim K^2 = n$

$\sim (\sqrt{n})$ times $\boxed{K = \sqrt{n}}$

```
void f(int n) {
    for(int i=1; i*i <= n; i++) {
        cout << "*";
    }
}
```

$\underbrace{\text{cout} \ll \text{"*"}}$ → const

$i=1 ;\ i^2 <= n ;\ i{+}{+}$

$\underbrace{i=1;\ i <= \sqrt{n} ;\ i{+}{+}}_{\sqrt{n} \text{ times}}$

$\sqrt{n} \cdot \text{const}$ $O(\sqrt{n})$

$\underbrace{10^8 - 10^9}_{1s}$ steps

$10^3 \sim 1024 = 2^{10}$

$\boxed{10^6 \sim 2^{20}}$

$10^9 \sim 2^{30}$

$10^{12} \sim 2^{40}$

$\vdots$

$\log_a a^b = b$

$n \le 10^6$ $O(n^2)$ $(10^6)^2 = 10^{12}$ = ↓ TLE

$n \log_2 n$ $\quad 10^6 \cdot \underbrace{\log_2 10^6}$ $= 10^6 \cdot \log_2 2^{20}$

$= 10^6 \cdot 20$ steps

```
void f(int n) {
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=i; j++) {
            cout << "*";
        }
    }
}
```

→ const

| i | j | # stars |
|---|---|---|
| 1 | 1 to <=1 | 1 |
| 2 | 1 to <=2 | 2 |
| 3 | 1 to <=3 | 3 |
| ⋮ | ⋮ | ⋮ |
| n | 1 to <=n | n |

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

$\sim n^2 \cdot \text{const}$

$O(n^2)$

```
void f(int n) {
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=i*i; j++) {
            cout << "*";
        }
    }
}
```

| i | j | # stars |
|---|---|---|
| 1 | 1 to $<=1^2$ | 1 $(1^2)$ |
| 2 | 1 to $<=2^2$ | 4 $(2^2)$ |
| 3 | 1 to $<=3^2$ | 9 $(3^2)$ |
| 4 | 1 to $<=4^2$ | 16 $(4^2)$ |
| ⋮ | ⋮ | ⋮ |
| n | 1 to $<=n^2$ | $n^2$ |

$\sum = 1^2 + 2^2 + 3^2 + \dots + n^2$

$= \frac{n(n+1)(2n+1)}{6}$

$\sim n^3 \cdot \text{const}$

$O(n^3)$

```cpp
void f(int n) {
    for(int i=1; i<n; i*=2) {
        cout << "*";
    }
}
```

i = i*2

const

$\log_2 n \cdot$ const

$\therefore \quad O(\log_2 n)$

| i | | |
|---|---|---|
| $2^0 = 1$ | $<n$ ? true | 1 |
| $2^1 = 2$ | $<n$ ? true | 2 |
| $2^2 = 4$ | $<n$ ? true | 3 |
| $2^3 = 8$ | $<n$ ? true | 4 |
| $2^4 = 16$ | ⋮ | |
| ⋮ | | |
| $2^k$ | $<n$ ? true | ? |

k+1

false

$\ngtr n$

assume

$i = n$

$\boxed{2^k = n}$

take $\log_2$

$\underbrace{\log_2 2^k}_{k} = \log_2 n$

$k = \log_2 n$

```cpp
void f(int n) {
    for(int i=n/2; i<=n; i++) {
        for(int j=1; j<=n/2; j++) {
            for(int k=1; k<n; k*=2) {
                cout << "*";
            }
        }
    }
}
```

n/2 times

n/2 times

$\underbrace{k=1;\ k<n;\ k*=2}_{\log_2 n}$

k times

const

$= \frac{n}{2}\left(\frac{n}{2}\left(\log_2 n\right)\right)$

$= \frac{n^2}{\cancel{4}} \cdot \log_2 n \quad = n^2 \log_2 n \cdot \text{const}$

$= O(n^2 \log_2 n)$

```cpp
void f(int n) {
    for(int i=n/2; i<=n; i++) {
        for(int j=1; j<n; j*=2) {
            for(int k=1; k<n; k*=2) {
                cout << "*";
            }
        }
    }
}
```

n/2

$\overbrace{j=1;\ j<n;\ j*=2}^{\log_2 n}$

$\underbrace{k=1;\ k<n;\ k*=2}_{\log_2 n}$

$= \frac{n}{\cancel{2}} \cdot \log_2 n \cdot \log_2 n \cdot \text{const}$

$= O\left(n(\log n)^2\right)$

```
void f(int n) {
    while(n > 1) {
        n /= 2;
    }
}
```

$n = n/2;$ → const

$n/2^0 = n$

$n/2^1 = n/2$

$n/2^2 = n/4$

$n/2^3 = n/8$

$\vdots$

$n/2^k$

$>1$ ? true — 1

$>1$ ? true — 2

$>1$ ? true — 3

$>1$ ? true — 4

$>1$ ? true — ?

↓ false ⟹ $\underbrace{n/2^k \not> 1}_{K+1}$

$\boxed{K \text{ times}}$

assume $n/2^k = 1$

$2^k = n$

taking $\log_2$ b/s

$\log_2 2^k = \log_2 n$

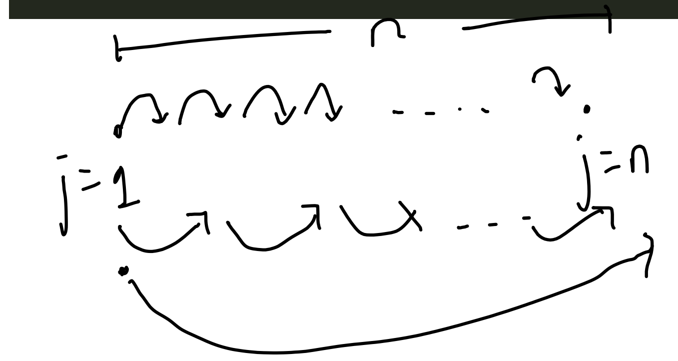
$K = \log_2 n$

$\log_2 n \cdot \text{const}$

$O(\log_2 n)$

```
void f(int n) {
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j+=i) {
            cout << "*";
        }
    }
}
```

$j = j + i$




| i | j | #stars |
| --- | --- | --- |
| 1 | 1 to <=n (j++) | $n$ |
| 2 | 1 to <=n (j=j+2) | $n/2$ |
| 3 | 1 to <=n (j=j+3) | $n/3$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| n | 1 to <=n (j=j+n) | 1 |

$\Sigma$

$$= \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n}$$

$$= n\left(\underbrace{\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}}_{\sim \log_2 n}\right)$$

$$= n \cdot \log_2 n$$

⟹

```
void f(int n) {
    cout << n*10;
}
```

$O(1)$

# Space Complexity Analysis of Iterative Algorithms

16B

```
void f(int n) {
    int a = 0;      4B
    int b = 1;      4B

    if(n == 0 || n == 1) {
        return n;
    }
                 4B
    for(int i=2; i<=n; i++) {
        int c = a+b;
        a = b;        4B
        b = c;
    }

    return b;
}
```

const

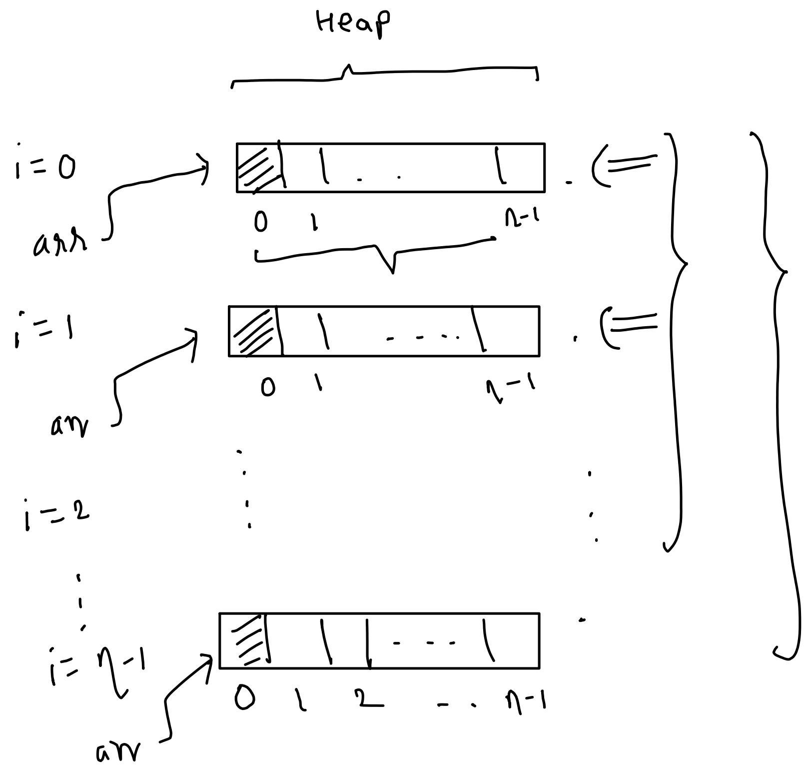
n-1 times

i=2; i<=n; i++

time: O(n)

space: O(1) → auxillary space is indep. of i/p size

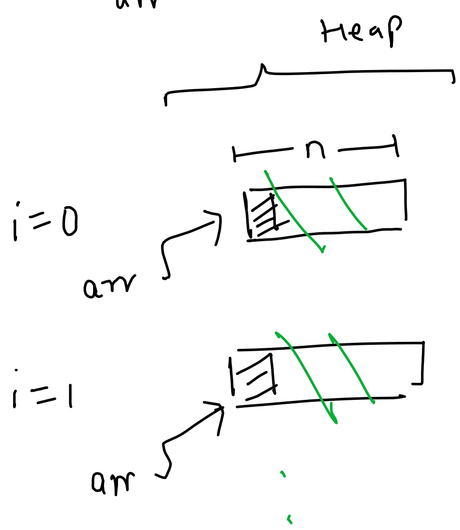
```cpp
void f(int n) {
    for(int i=0; i<=n; i++) {
        int* arr = new int[n];
    }
}
```

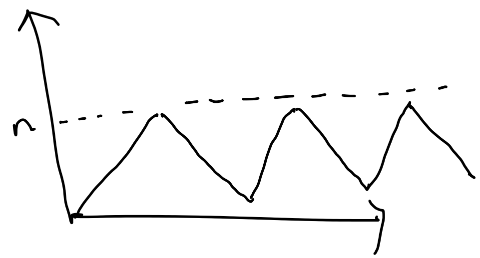



$\underbrace{n \cdot n}_{\downarrow} = O(n^2)$

peak demand of space

```cpp
void f(int n) {
    for(int i=0; i<=n; i++) {
        int* arr = new int[n];
        delete [] arr;
    }
}
```




O(n)