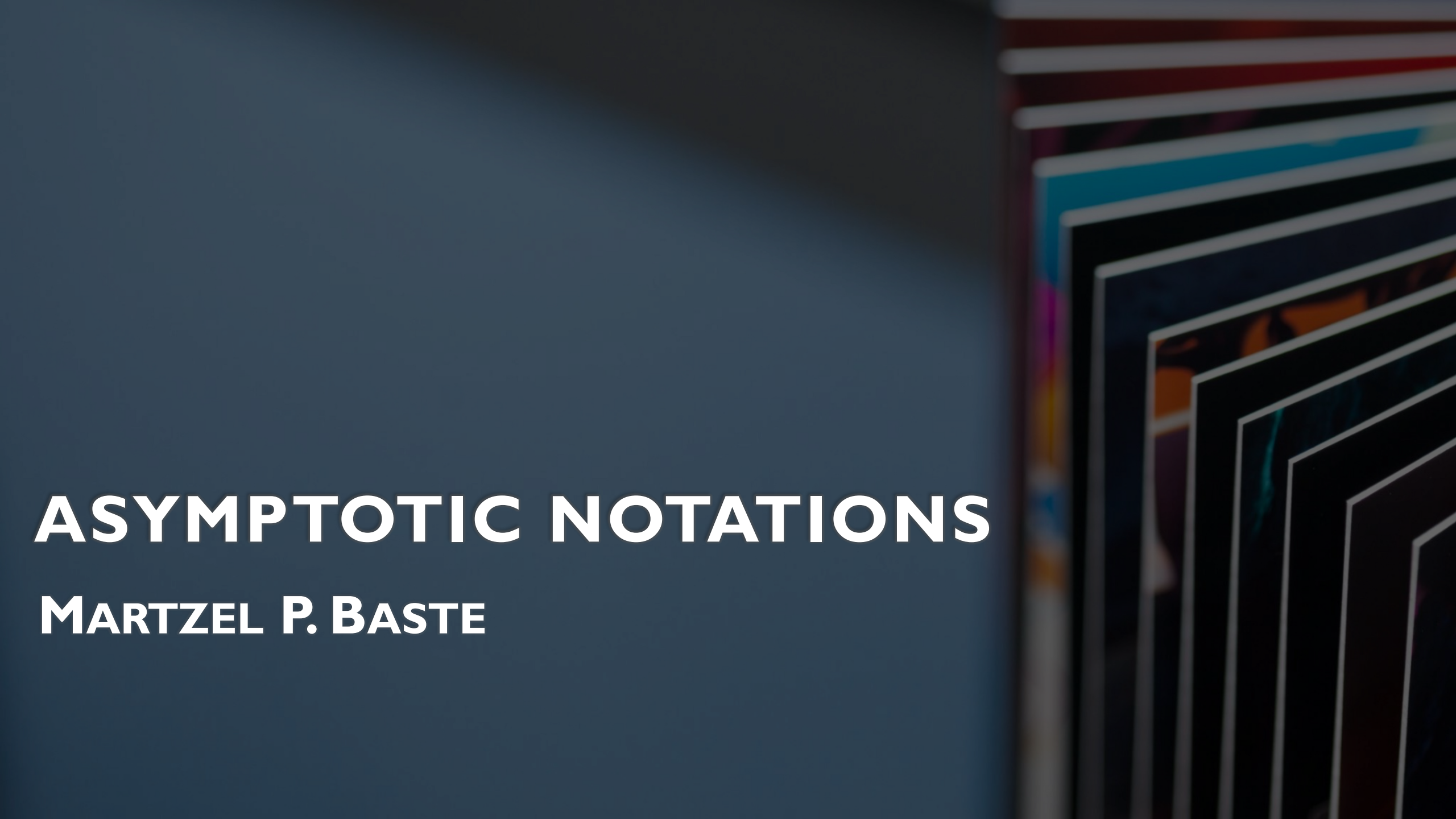
ASYMPTOTIC NOTATIONS

MARTZEL P. BASTE

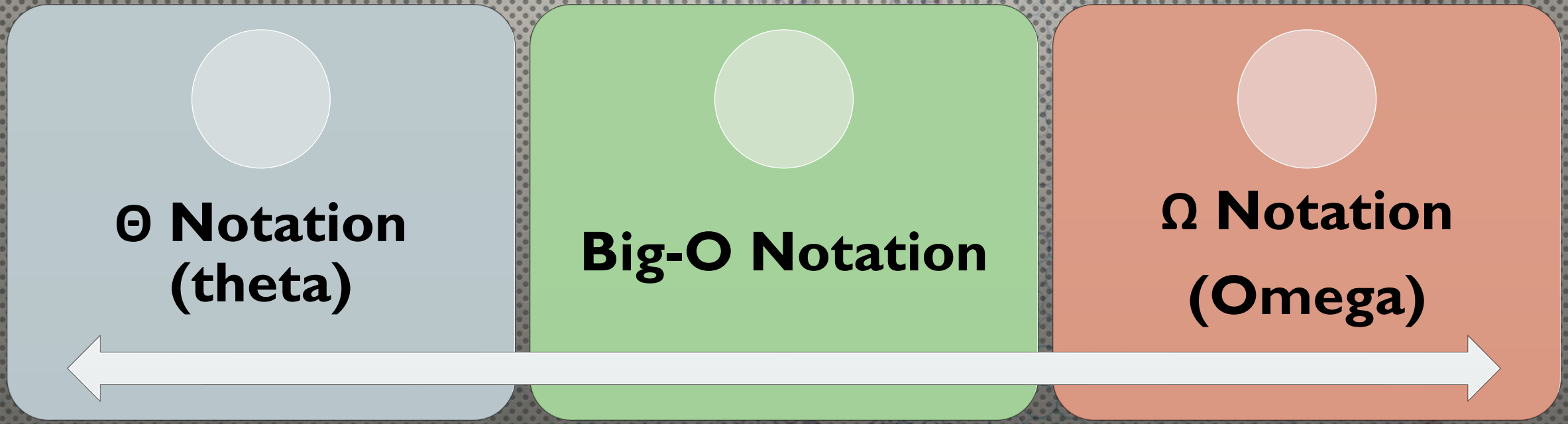


INTENDED LEARNING OUTCOMES

Explore the three Asymptotic Notations

Calculating the complexity based on Asymptotic Notations

ASYMPTOTIC NOTATIONS



```java
public boolean find(int arr[], int max, int val) {
    for (int i = 0; i < max; ++i) {
        if (arr[i] == val)
            return true;
    }
    return false;
}
```

array

size

find

[3, 5, 1, 10, 8]

Best case

find(3)

Average case

$A = (x_1 + x_2 + x_3 \cdots + x_n)/n$

Worst case

find(9)

TYPES OF NOTATIONS FOR TIME COMPLEXITY

Big Oh denotes "*fewer than or the same as*" <expression> iterations.

Maximum (Upper): Worst

Big Omega denotes "*more than or the same as*" <expression> iterations.

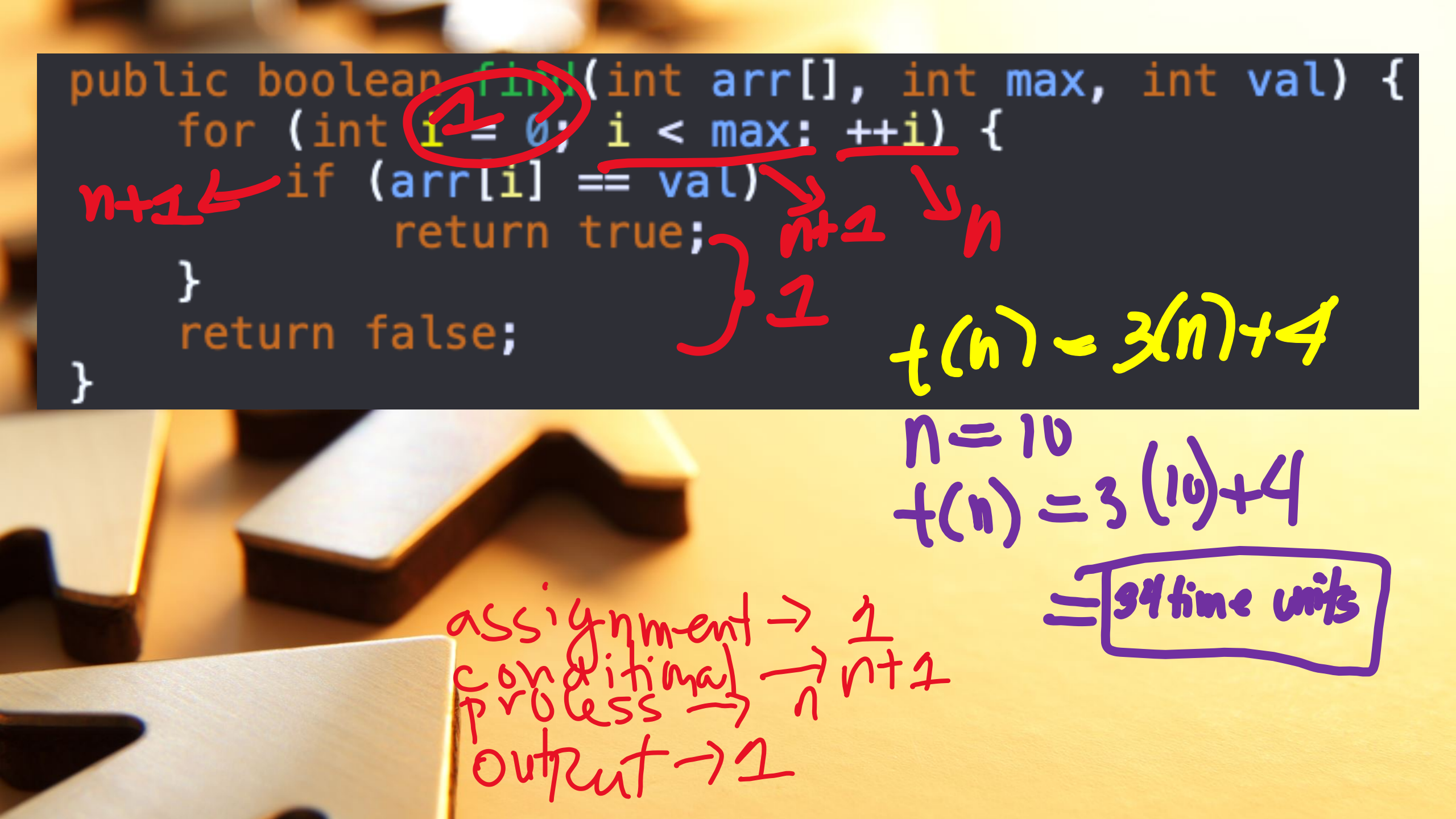
Minimum (Lower): Best

Big Theta denotes "*the same as*" <expression> iterations.

Average Bound

Little Oh denotes "*fewer than*" <expression> iterations.

Little Omega denotes "*more than*" <expression> iterations.


public boolean find(int arr[], int max, int val) {
for (int i = 0; i < max; ++i) {
if (arr[i] == val)
return true;
}
return false;
}
1
n+1
n
n+1
1
t(n) = 3(n)+4
n = 10
t(n) = 3(10)+4
= 34 time units
assignment → 1
conditional → n+1
process → n
output → 1

CALCULATING TIME COMPLEXITY
CONSTANT O(1)

The time complexity of algorithms is most commonly expressed using the **big O notation**. It's an asymptotic notation to represent the time complexity

```
sum=max*(max+1)/2;
```

CALCULATING TIME COMPLEXITY

CONSTANT O(N)

```
int max=new Scanner(System.in).nextInt();
int sum=0;
for(int i=1; i <=max; i++) {
    sum=sum+i;
}
System.out.println("The sum 1 to "+max+" is "+sum);
```

The time complexity for the above algorithm will be **Linear**. The running time of the loop is directly proportional to N. When N doubles, so does the running time.

CALCULATING TIME COMPLEXITY

QUADRATIC $O(N^2)$

```
int N=new Scanner(System.in).nextInt();
for(int a=1;a<=N; a++) {
    for(int b=1; b<=N;b++) {
        System.out.print(b);;
    }
    System.out.println();
}
```

$$t(n)=\sum_{i=1}^{n}[2n+3]+2n+4$$

$$=2\sum_{i=1}^{n}n+\sum_{i=1}^{n}2+2n+4$$

$$=2n^2+4n+4$$

This time, the time complexity for the above code will be **Quadratic**. The running time of the two loops is proportional to the square of N. When N doubles, the running time increases by N * N.

CALCULATING TIME COMPLEXITY - OTHER

- **Logarithmic** Time Complexity
- linear and logarithmic

SUMMARY

- **Best case**. Find at first position: 1 compare
- **Worst case**. Find at last position: N compares
- **Average case**. [(n + 1)/2 compares]
- There are mainly three asymptotic notations:
  - Big-O notation (Worst)
  - Omega notation (Best)
  - Theta notation (Average)
- Complexity and running time are often expressed in "Big O notation," which is used to describe the approximate amount of time an algorithm requires to complete, based the size of its input.
- $O(1), O(N), O(LogN), O(N^2), O(2^N)$ are the most commonly used Big-O

THANK YOU