Time Complexity Analysis

What do we mean by the time complexity of a running program? - Total operations performed

Big-O

Example 1:

```
void main()
{
   int x = 5;
   x = x + 3;
   print(x);
}
```

1 + 2 + 1 = 4

Constant

Time → O(1)

Example 2:

```
void main()
{
   int n;
   input(n);

   for (int i = 0; i < n; i++)
      print(i);
}
```

Linear

$\propto n$

O(n)

Time

Time Complexity Analysis

Why care about time complexities? – Consider the below example

- Write an algorithm to sum first n natural numbers.

1 + 2 + 3 + 4 - - - - - n

// Brute Force Approach

```
void main()
{
   int n; // Input from user

   int sum = 0;
   for (int i = 1; i <= n; i++)
      sum += i;

   print(sum);
}
```

time → O(n)

Time Complexity Analysis

Why care about time complexities? – Consider the below example

- Write an algorithm to sum first n natural numbers.

```
// Brute Force Approach

void main()
{
   int n;  // Input from user

   int sum = 0;
   for (int i = 1; i <= n; i++)
      sum += i;

   print(sum);
}
```

time → O(n)

```
// Efficient Approach

void main()
{
   int n;  // Input from user

   int sum = (n * (n + 1)) / 2;

   print(sum);
}
```

time → O(1)

Time Complexity Analysis

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

Time Complexity Analysis

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

time → $O(\log_2 n)$

Example for O(logn)

```
void main()
{
   int n;  // Input from user

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

Few Important Algorithms with O(logn) complexities:

- Binary Search
- Certain Divide & Conquer Algorithms

$2^k = n$

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

$k = \log_2 n$

$\Rightarrow \log_a(a^x) \Rightarrow x$

1, 2, 4, 8, 16 ---------- n

$2^0, 2^1, 2^2, 2^3, 2^4$ ---------- 2^k

k

Time Complexity Analysis

```
void main()
{
   int n;  // Input from user

   for (int i = 1; i <= n; i = i * 5)
      print(i);
}
```

time → $O(\log_5 n)$

```
void main()
{
   int n;  // Input from user   +ve

   for (int i = 0; i <= n; i = i * 7)
      print(i);
}
```

time → $O(\infty)$

Time Complexity Analysis

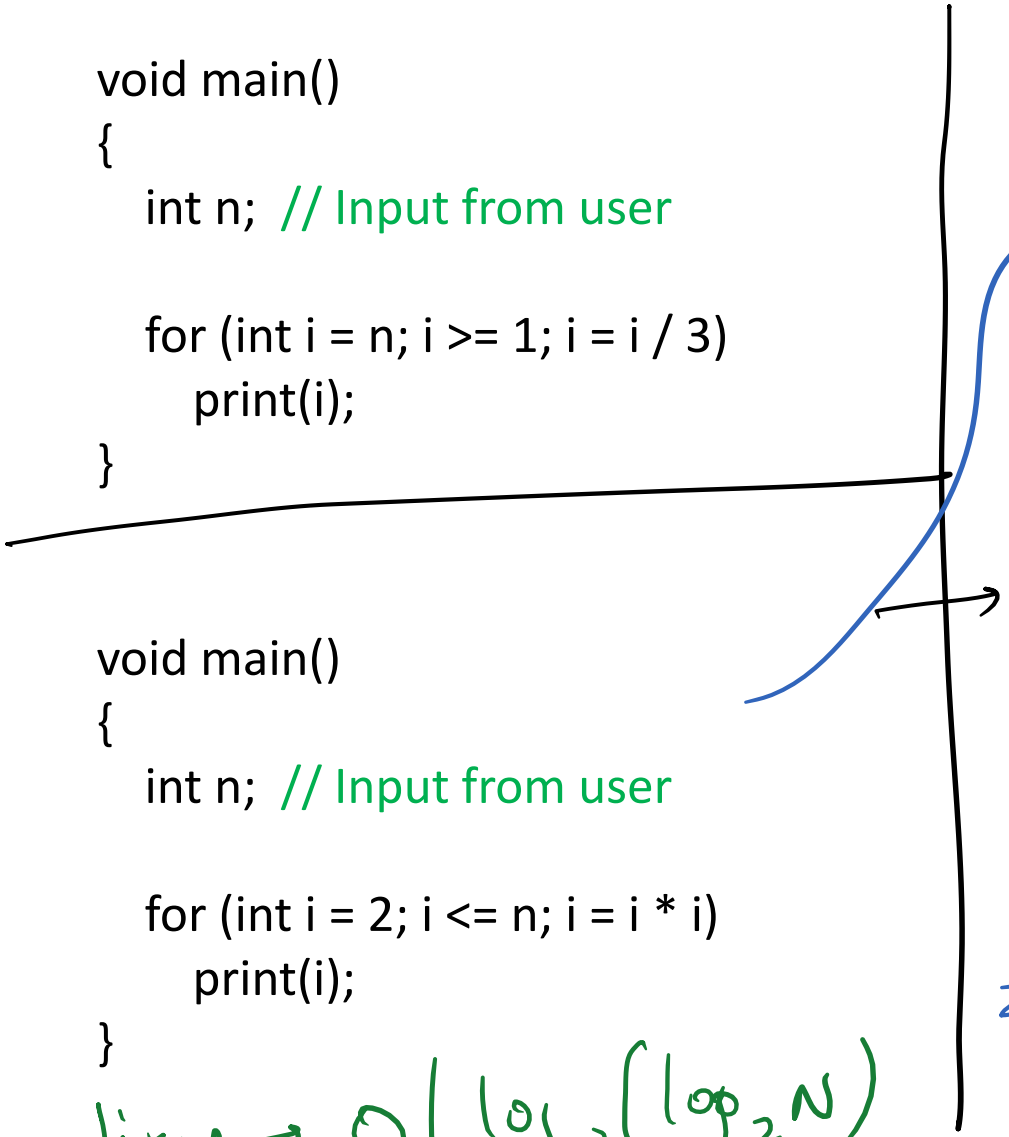
```
void main()
{
   int n;  // Input from user

   for (int i = n; i >= 1; i = i / 3)
      print(i);
}
```

```
void main()
{
   int n;  // Input from user

   for (int i = 2; i <= n; i = i * i)
      print(i);
}
```

time → $O(\log_2(\log_2 N))$



$2, 4, 16, 256 \ldots\ldots\ldots N$

$2^1, 2^2, 2^4, 2^8 \ldots\ldots\ldots N$

$2^{2^0}, 2^{2^1}, 2^{2^2}, 2^{(2^3)} \ldots\ldots\ldots 2^{(2^k)}$

$2^{2^k} = N$

$\Rightarrow \log_2(2^{2^k}) = \log_2 N$

$\boxed{2^k = \log_2 N} \Rightarrow k = \log_2(\log_2 N)$

Time Complexity Analysis

```
void main()
{
   int n;  // Input from user

   int p = 1;

   for (int i = 1; i <= n; i = i * 2)
      p++;

   for (int j = 1; j <= p; j = j * 2)
      print(j)
}
```

time → log (log n)

```
void main()
{
   int n;  // Input from user

   for (int i = n; i >= 1; i = i / 2)
   {
      for (int j = 1; j <= i; j++)
         print (j)
   }
}
```

time → O(n)

Time Complexity Analysis

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

Example for O(√n)

```
void main()
{
   int n;  // Input from user

   for (int i = 1; i <= √n; i++)
      print(i);
}
```

time → $O(\sqrt{n})$

Few Important Algorithms with O(√n) complexities:

- Verify Prime Number

Time Complexity Analysis

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

Example for O(nlogn)

```
void main()
{
   int n;  // Input from user

   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j = j * 2)
         print(i);
}
```

Few Important Algorithms with O(nlogn) complexities:

- Quick Sort
- Merge Sort
- Heap Sort

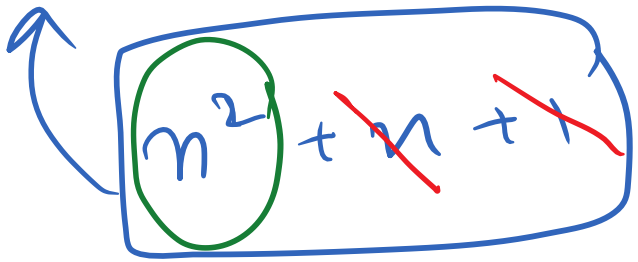
Time Complexity Analysis

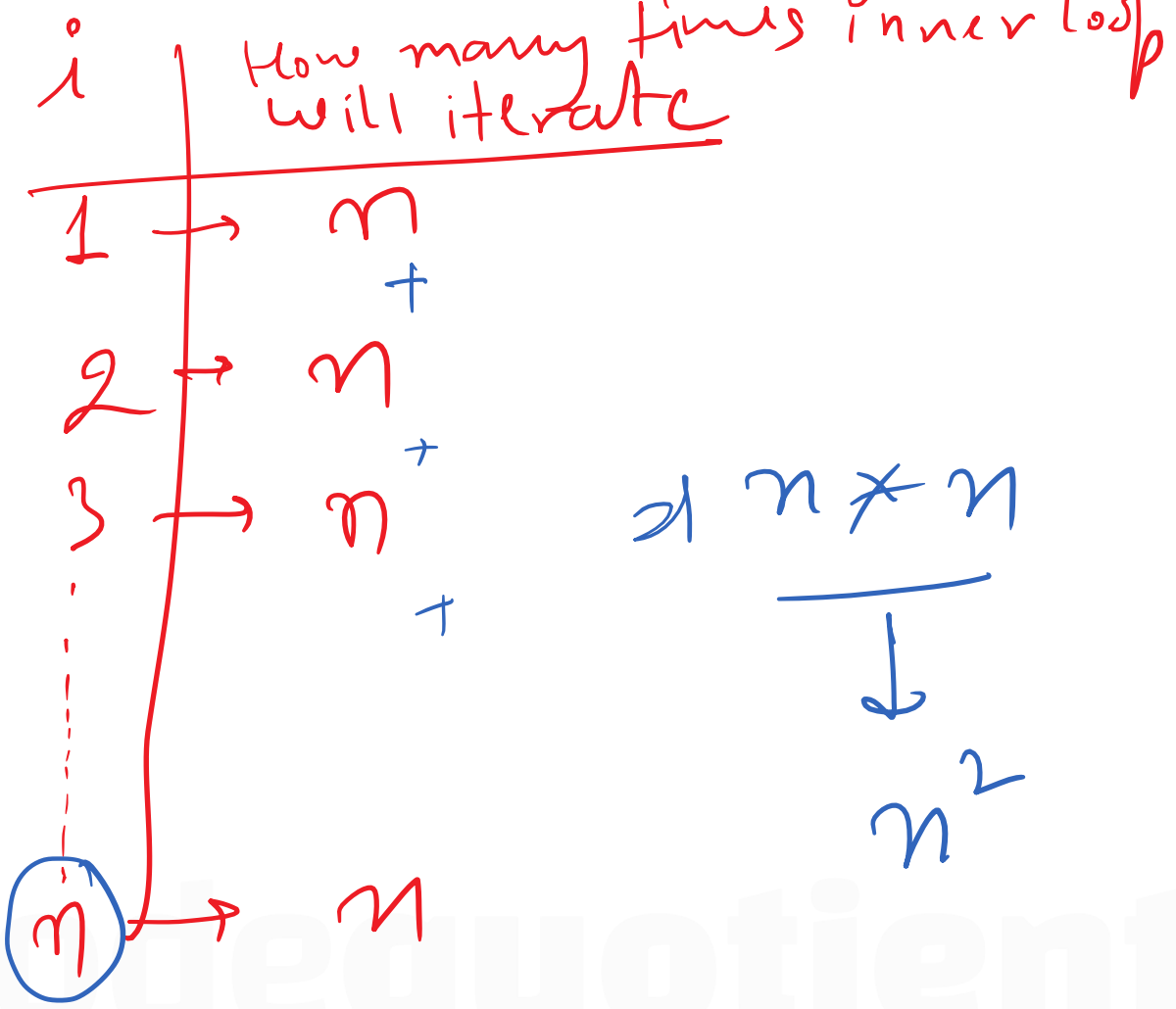
$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

Example for $O(n^2)$

```
void main()
{
   int n;  // Input from user

   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
         print(i);
}
```





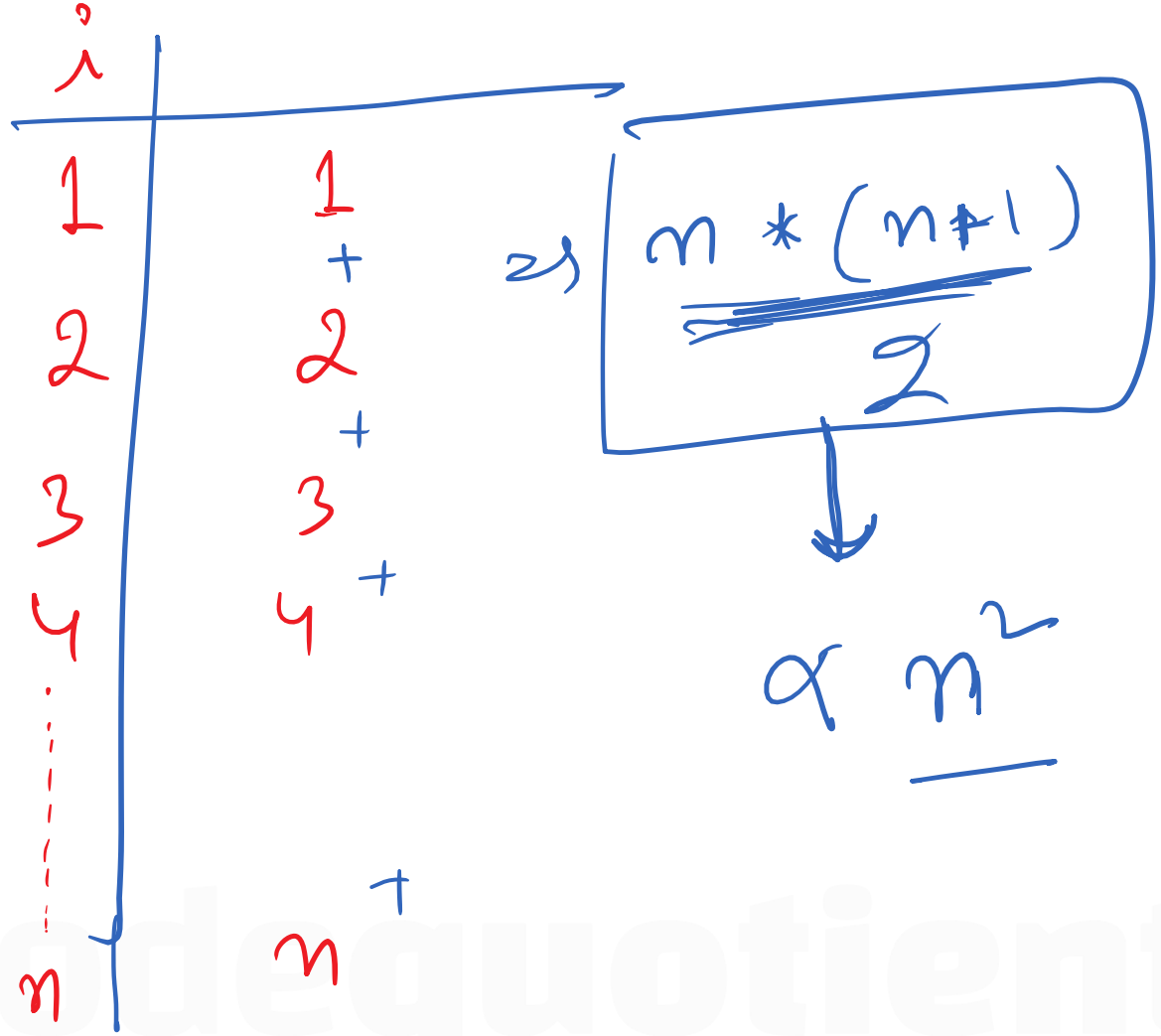
Time Complexity Analysis

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

Example for $O(n^2)$

```
void main()
{
   int n;  // Input from user
   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= i; j++)
         print(i);
}
```

time $\Rightarrow O(n^2)$



Time Complexity Analysis

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

Example for $O(2^n)$

2^n

Backtracking

Given an array with n distinct elements, print all the distict subsets of it.

arr[] = {1, 2, 3}

x x x

2 x 2 x 2 ---- 2^n

{}, {1}, {2}, {3}

{1, 2}, {2, 3}, {1, 3}

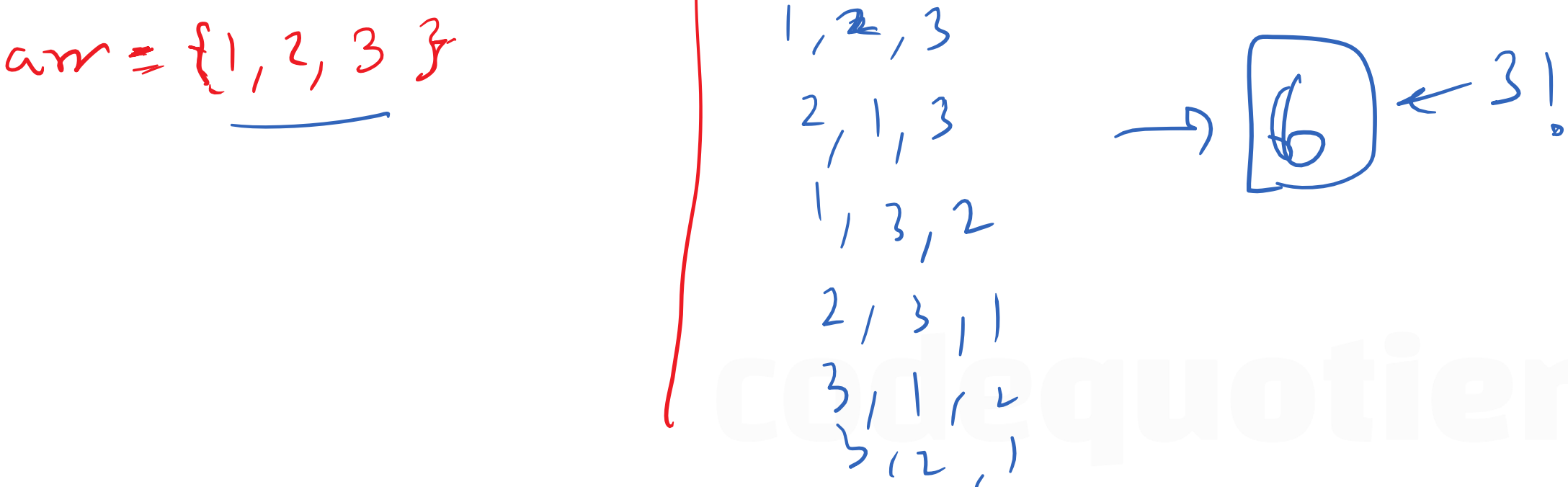
{1, 2, 3}

Time Complexity Analysis

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

Example for O(n!)

Given an array with n distinct elements, print all the permutations of it.



Time Complexity Analysis

Never judge the time complexity by just looking at the number of loops / nested loops.

For Example 1

```
void main()
{
   int n;  // Input from user

   for (int i = 1; i <= n; i++)
      for (int j = i; j <= i; j++)
         print(i);
}
```

time → O(n)

Example 2

```
void main()
{
   int n;  // Input from user

   for (int i = 1; i <= n; i--)
      print(i);
}
```

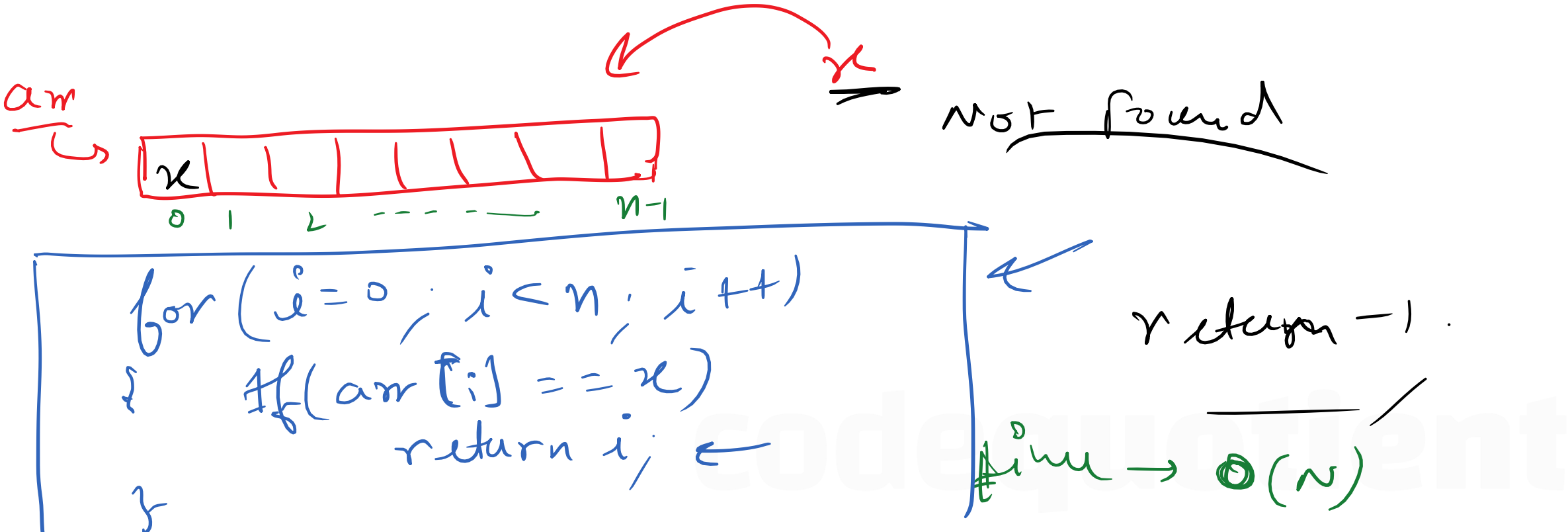
+ve

time → ∞

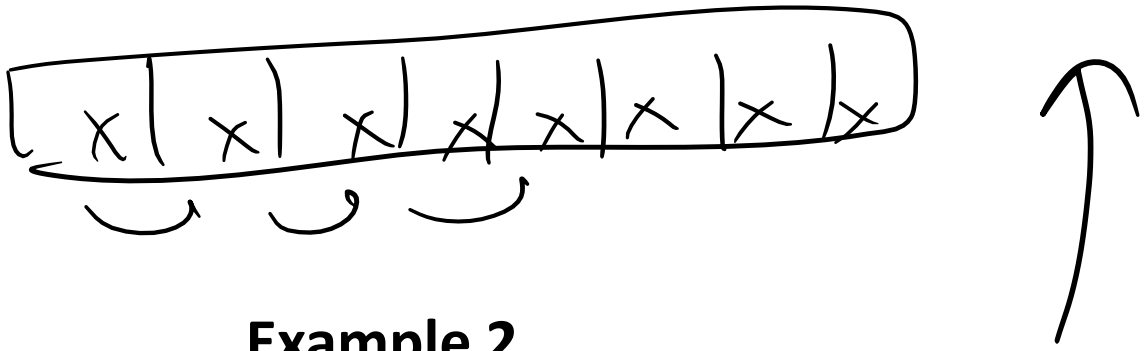
Time Complexity Analysis

General points for finding time complexity:

- For large inputs
- Considering Worst Case scenarios



Space Complexity Analysis



Example 1

```
void main()
{
    int a = 1, b = 2, c = 3;
    int d = a + b + c;
    print(d);
}
```

Space → O(1)

Example 2

```
void main()
{
    int n;          // Input from user
    int arr[n];

    for (int i = 0; i < n; i--)
        // do something
    }
```

Space → O(n)