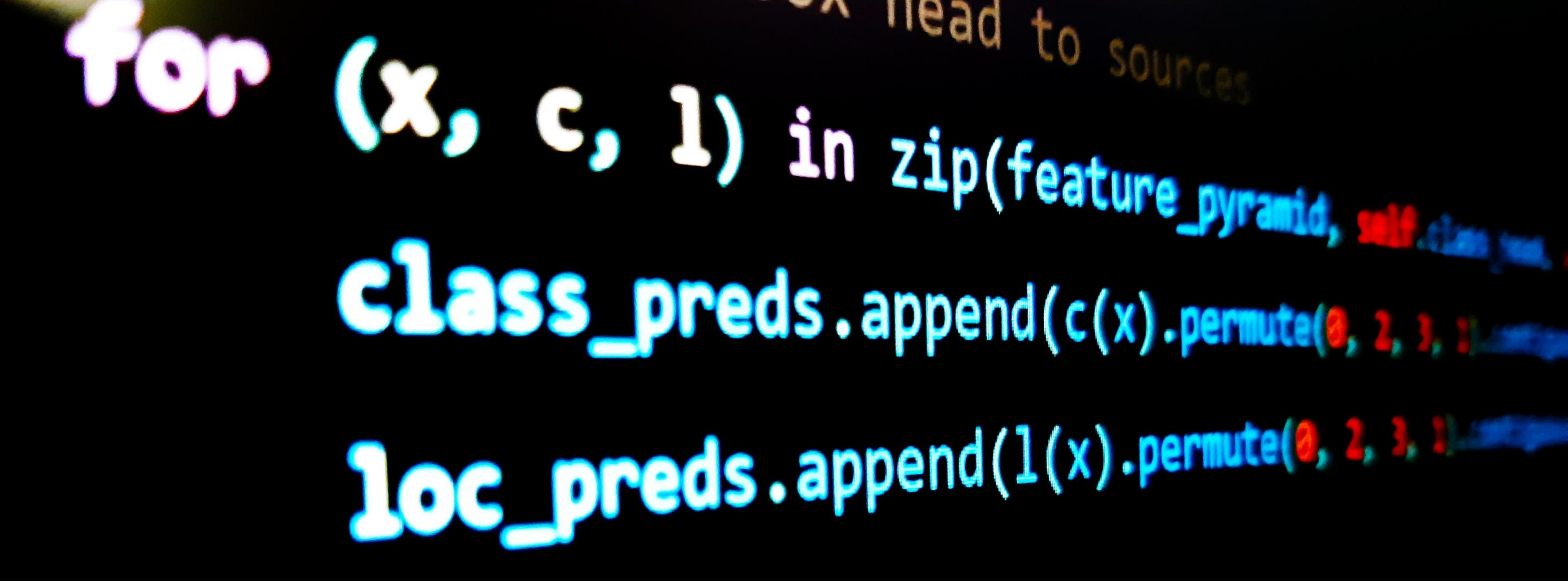


RECURRENCES

CCDSALG T2 AY 2020-2021

RECURRENCE RELATIONS

- Recurrence relations are used to solve for the run-time of a recursive algorithm.

- Recurrence relations will be the mathematical tool that allows us to analyze recursive algorithms.

RECURRENCE RELATIONS

Recursion

A problem-strategy that solves large problems by reducing them to smaller problems of the same form

RECURRENCE RELATIONS

An example is the recursive algorithm for finding the factorial of an input number n. Let us say $n = 4$, thus we want to compute $4! = 4 \times 3 \times 2 \times 1 = 24$

RECURRENCE RELATIONS

Each factorial is related to the factorial of the next smaller integer: $n! = n \times (n - 1)!$

RECURRENCE RELATIONS

Each factorial is related to the factorial of the next smaller integer: $n! = n \times (n-1)!$

$$4! = 4 \ \times 3!$$

RECURRENCE RELATIONS

Each factorial is related to the factorial of the next smaller integer: $n! = n \times (n-1)!$

$$4! = 4 \times \boxed{3!}$$

$$\boxed{3!} = 3 \times 2!$$

RECURRENCE RELATIONS

Each factorial is related to the factorial of the next smaller integer: $n! = n \times (n-1)!$

$$4! = 4 \times \boxed{3!}$$
$$\boxed{3!} = 3 \times \boxed{2!}$$
$$\boxed{2!} = 2 \times 1!$$

RECURRENCE RELATIONS

Each factorial is related to the factorial of the next smaller integer: $n! = n \times (n-1)!$

$$
\begin{aligned}
4! &= 4 \times \boxed{3!} \\
\boxed{3!} &= 3 \times \boxed{2!} \\
\boxed{2!} &= 2 \times \boxed{1!} \\
\boxed{1!} &= 1
\end{aligned}
$$

RECURRENCE RELATIONS

Each factorial is related to the factorial of the next smaller integer: $n! = n \times (n-1)!$

$$4! = 4 \times \boxed{3!}$$

$$\boxed{3!} = 3 \times \boxed{2!}$$

$$\boxed{2!} = 2 \times \boxed{1!}$$

$$\boxed{1!} = 1$$

$$4! = 4 \times 3 \times 2 \times 1$$

RECURRENCE RELATIONS

Mathematically, the factorial function can be defined as:

$$n! = n \times (n-1)! \qquad if\ n > 1$$

$$n! = 1 \qquad if\ n = 1$$

RECURRENCE RELATIONS

Recursive function for finding the factorial of an input n:

```
int factorial (int n) {
   if (n == 1)
      return 1;
   else
      return n * factorial (n - 1);
}
```

Base case

Recursive step

RECURRENCE RELATIONS

Recursive function for finding the factorial of an input n:

```
int factorial (int n) {
   if (n == 1)
      return 1;
   else
      return n * factorial (n - 1);
}
```

$$T(n) = \begin{cases} a, & n = 1 \\ T(n-1) + b, & n > 1 \end{cases}$$

RECURRENCE RELATIONS

What is this function doing?

```
long func(long x, long n) {
  if (n == 0)
    return 1;
  else
    return x * func(x, n - 1);
}
```

RECURRENCE RELATIONS

Power function: Suppose $n\ = 3$ and $\mathrm{k} = 2$, thus the function will compute $2^3 = 2 \times 2 \times 2 = 8$.

RECURRENCE RELATIONS

Note that each factor is related: $k^n = k \times k^{n-1}$

RECURRENCE RELATIONS

Note that each factor is related: $k^n = k \times k^{n-1}$

$$2^3 = 2 \times 2^2$$

RECURRENCE RELATIONS

Note that each factor is related: $k^n = k \times k^{n-1}$

$$2^3 = 2 \times \boxed{2^2}$$

$$2^2 = 2 \times 2^1$$

RECURRENCE RELATIONS

Note that each factor is related: $k^n = k \times k^{n-1}$

$$2^3 = 2 \times \boxed{2^2}$$

$$\boxed{2^2} = 2 \times \boxed{2^1}$$

$$\boxed{2^1} = 2 \times 2^0$$

RECURRENCE RELATIONS

Note that each factor is related: $k^n = k \times k^{n-1}$

$$2^3 = 2 \times \boxed{2^2}$$

$$\boxed{2^2} = 2 \times \boxed{2^1}$$

$$\boxed{2^1} = 2 \times \boxed{2^0}$$

$$\boxed{2^0} = 1$$

RECURRENCE RELATIONS

Note that each factor is related: $k^n = k \times k^{n-1}$

$$2^3 = 2 \times \boxed{2^2}$$

$$\boxed{2^2} = 2 \times \boxed{2^1}$$

$$\boxed{2^1} = 2 \times \boxed{2^0}$$

$$\boxed{2^0} = 1$$

$$2^3 = 2 \times 2 \times 2 \times 1$$

RECURRENCE RELATIONS

Mathematically, the power function can be defined as:

$$k^n = k \times k^{n-1} \qquad if\ n > 0$$

$$k^n = 1 \qquad if\ n = 0$$

RECURRENCE RELATIONS

```
long power(long x, long n) {
  if (n == 0)
    return 1;
  else
    return x * power (x, n - 1);
}
```

Base case

Recursive step

RECURRENCE RELATIONS

```
long power(long x, long n) {
  if (n == 0)
    return 1;
  else
    return x * power (x, n - 1);
}
```

$$T(n) = \begin{cases} a, & n = 0 \\ T(n-1) + b, & n > 0 \end{cases}$$

RECURRENCE RELATIONS

What is this function doing?

```
int func(int n) {
  if (n <= 1)
    return n;
  else
    return func(n – 1) + func(n - 2);
}
```

RECURRENCE RELATIONS

- Fibonacci function: Let us say $n = 4$, thus we want to get the n[th] number in the Fibonacci sequence $([0], 1, 1, 2, 3, \ldots)$, which is 3.
- Note that each number in the sequence is related:
  - $fib(0) = 0$
  - $fib(1) = 1$
  - $fib(2) = fib(2-1) + fib(2-2) = 0 + 1 = 1$
  - $fib(3) = fib(3-1) + fib(3-2) = 1 + 1 = 2$
  - $fib(4) = fib(4-1) + fib(4-2) = 2 + 1 = 3$

RECURRENCE RELATIONS

```
int fib(int n) {
  if (n <= 1)
    return n;
  else
    return fib(n – 1) + fib(n - 2);
}
```

Base case

Recursive step

RECURRENCE RELATIONS

```
int fib(int n) {
  if (n <= 1)
    return n;
  else
    return fib(n – 1) + fib(n - 2);
}
```

$$T(n) = \begin{cases} a, & n \leq 1 \\ T(n-1) + T(n-2) + b, & n > 1 \end{cases}$$

RECURRENCE RELATIONS

- Once the recurrence is defined, remove all of the $T(\ldots)$'s from the right side of the equation
- Determine the closed form and then solve for the number of operations in terms of n.
- Using the number of operations, determine the big-oh run time.

SOLVING RECURRENCES

1. Iteration Method
2. Substitution Method
3. Master's Method

SOLVING RECURRENCES

1. Iteration Method
2. Substitution Method
3. Master's Method

ITERATION METHOD

$$T(n) = \begin{cases} 2, & n = 1 \\ T(n-1) + 3, & n > 1 \end{cases}$$

ITERATION METHOD

$$T(n) = \begin{cases} 2, & n = 1 \\ T(n-1) + 3, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + 3$

ITERATION METHOD

$$T(n) = \begin{cases} 2, & n = 1 \\ T(n-1) + 3, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + 3$

Iteration 2: $T(n) = [T\big((n-1)-1\big) + 3] + 3$

$T(n) = [T(n-2) + 3] + 3$

ITERATION METHOD

$$T(n) = \begin{cases} 2, & n = 1 \\ T(n-1) + 3, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + 3$

Iteration 2: $T(n) = [T\big((n-1) - 1\big) + 3] + 3$

$T(n) = [T(n-2) + 3] + 3$

Iteration 3: $T(n) = \Big[[T\big((n-2) - 1\big) + 3] + 3\Big] + 3$

$T(n) = \big[[T(n-3) + 3] + 3\big] + 3$

ITERATION METHOD

$$T(n) = \begin{cases} 2, & n = 1 \\ T(n-1) + 3, & n > 1 \end{cases}$$

Iteration 3: $T(n) = \Big[\big[T\big((n-2)-1\big) + 3\big] + 3\Big] + 3$

$$T(n) = \big[[T(n-3) + 3] + 3\big] + 3$$

…

Iteration i: $T(n) = T(n-i) + 3i$

ITERATION METHOD

$$T(n) = \begin{cases} 2, & n = 1 \\ T(n-1) + 3, & n > 1 \end{cases}$$

Iteration i: $T(n) = T(n-i) + 3i$

If $n - i = 1$, then $i = n - 1$

- $T(n) = T(n-i) + 3i$
- $T(n) = T\big(n - (n-1)\big) + 3(n-1)$
- $T(n) = T(1) + 3n - 3$
- $T(n) = 2 + 3n - 3$
- $\boldsymbol{T(n) = 3n - 1}$ **Big oh: $\boldsymbol{O(n)}$**

ITERATION METHOD

$$T(n) = \begin{cases} a, & n = 1 \\ T(n-1) + b, & n > 1 \end{cases}$$

ITERATION METHOD

$$T(n) = \begin{cases} a, & n = 1 \\ T(n-1) + b, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + b$

ITERATION METHOD

$$T(n) = \begin{cases} a, & n = 1 \\ T(n-1) + b, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + b$

Iteration 2: $T(n) = [T\big((n-1)-1\big) + b] + b$

$T(n) = [T(n-2) + b] + b$

ITERATION METHOD

$$T(n) = \begin{cases} a, & n = 1 \\ T(n-1) + b, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + b$

Iteration 2: $T(n) = [T\big((n-1) - 1\big) + b] + b$

$$T(n) = [T(n-2) + b] + b$$

Iteration 3: $T(n) = \Big[[T\big((n-2) - 1\big) + b] + b\Big] + b$

$$T(n) = \Big[[T(n-3) + b] + b\Big] + b$$

ITERATION METHOD

$$T(n) = \begin{cases} a, & n = 1 \\ T(n-1) + b, & n > 1 \end{cases}$$

Iteration 3: $T(n) = \Big[\big[T\big((n-2)-1\big)+b\big]+b\Big]+b$

$$T(n) = \big[\big[T(n-3)+b\big]+b\big]+b$$

...

Iteration i: $T(n) = T(n-i) + ib$

ITERATION METHOD

$$T(n) = \begin{cases} a, & n = 1 \\ T(n-1) + b, & n > 1 \end{cases}$$

Iteration i: $T(n) = T(n-i) + ib$

If $n - i = 1$, then $i = n - 1$

- $T(n) = T(n-i) + ib$
- $T(n) = T\big(n - (n-1)\big) + (n-1)b$
- $T(n) = T(1) + bn - b$
- $\boldsymbol{T(n) = a + bn - b}$

Big-oh: $\boldsymbol{O(n)}$

ITERATION METHOD

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2n - 1, & n > 0 \end{cases}$$

ITERATION METHOD

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2n - 1, & n > 0 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + 2n - 1$

ITERATION METHOD

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2n - 1, & n > 0 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + 2n - 1$

Iteration 2: $T(n) = \left[T\big((n-1)-1\big) + 2(n-1) - 1\right] + 2n - 1$

$$T(n) = [T(n-2) + 2n - 3] + 2n - 1$$

$$T(n) = T(n-2) + 4n - 4$$

ITERATION METHOD

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2n - 1, & n > 0 \end{cases}$$

Iteration 1: $T(n) = T(n-1) + 2n - 1$

Iteration 2: $T(n) = \left[T\big((n-1)-1\big) + 2(n-1) - 1\right] + 2n - 1$

$$T(n) = [T(n-2) + 2n - 3] + 2n - 1$$

$$T(n) = T(n-2) + 4n - 4$$

Iteration 3: $T(n) = \left[T\big((n-2)-1\big) + 2(n-2) - 1\right] + 4n - 4$

$$T(n) = T(n-3) + 6n - 9$$

ITERATION METHOD

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2n - 1, & n > 0 \end{cases}$$

Iteration 3: $T(n) = \left[T\left((n-2)-1\right) + 2(n-2) - 1\right] + 4n - 4$

$T(n) = T(n-3) + 6n - 9$

...

Iteration i: $T(n) = T(n-i) + 2in - i^2$

ITERATION METHOD

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2n - 1, & n > 0 \end{cases}$$

Iteration i: $T(n) = T(n-i) + 2in - i^2$

If $n - i = 0$, then $i = n$

- $T(n) = T(n-n) + 2nn - n^2$
- $T(n) = T(0) + 2n^2 - n^2$
- $T(n) = 0 + n^2$
- $\boldsymbol{T(n) = n^2}$

Big-oh: $\boldsymbol{O(n^2)}$

ITERATION METHOD

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

ITERATION METHOD

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T\left(\frac{n}{2}\right) + 1$

ITERATION METHOD

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T\left(\frac{n}{2}\right) + 1$

Iteration 2: $T(n) = \left[T\left(\frac{\frac{n}{2}}{2}\right) + 1\right] + 1 = \left[T\left(\frac{n}{4}\right) + 1\right] + 1$

ITERATION METHOD

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

Iteration 1: $T(n) = T\left(\frac{n}{2}\right) + 1$

Iteration 2: $T(n) = \left[T\left(\frac{\frac{n}{2}}{2}\right) + 1\right] + 1 = \left[T\left(\frac{n}{4}\right) + 1\right] + 1$

Iteration 3: $T(n) = \left[\left[T\left(\frac{\frac{n}{4}}{2}\right) + 1\right] + 1\right] + 1 = \left[\left[T\left(\frac{n}{8}\right) + 1\right] + 1\right] + 1$

ITERATION METHOD

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

Iteration 3: $T(n) = \left[\left[T\left(\frac{n}{8}\right) + 1\right] + 1\right] + 1$

…

Iteration i: $T(n) = T\left(\frac{n}{2^i}\right) + i$

ITERATION METHOD

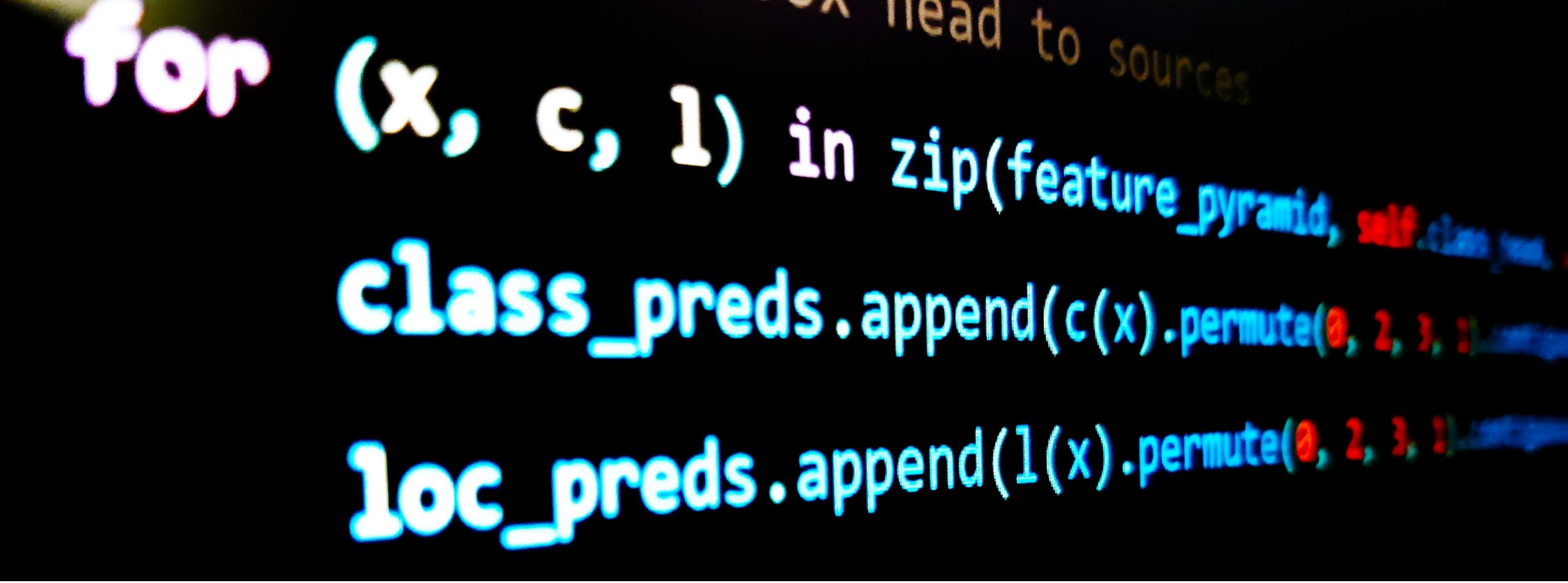
$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

Iteration i: $T(n) = T\left(\frac{n}{2^i}\right) + i$

If $\frac{n}{2^i} = 1$, then $n = 2^i$ and $\log_2 n = i$.

- $T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n$
- $T(n) = T\left(\frac{n}{n}\right) + \log_2 n$
- $T(n) = T(1) + \log_2 n$
- $T(n) = \log_2 n + 1$

Big-Oh: $\boldsymbol{O(\log_2 n)}$



RECURRENCES

CCDSALG T2 AY 2020-2021