CS302 – Analysis and Design of Algorithms

Algorithm Analysis

Content
Bubble Sort
Analyzing Bubble Sort
Time Complexity Analysis of Loops
Exercises

Bubble Sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
- Steps:
  1. Start at the beginning of the list– this the pass number.
  2. Compare the last value in the list with the next one down.
  3. If the second value is bigger, swap the positions of the two values.
  4. Move one step down the list.
  5. Again, compare this value with the next and swap if the value is bigger.
  6. Keep going until there are no more items to compare.
  7. Go back to the start of the list – start a new pass.

Bubble Sort

8	4	6	9	2	3	1

Bubble Sort

8	4	6	9	2	3	1

$i = 0$ <-- j

Bubble Sort

8	4	6	9	2	3	1

$i = 0$ <-- j

8	4	6	9	2	1	3

$i = 0$ <---------------------------------- j

Bubble Sort

8	4	6	9	2	3	1

$i = 0$ <--- j

8	4	6	9	2	1	3

$i = 0$ <----------------------------------- j

8	4	6	9	1	2	3

$i = 0$ <-------------------------- j

Bubble Sort

8	4	6	9	2	3	1

$i = 0$ <--- j

8	4	6	9	2	1	3

$i = 0$ <----------------------------------- j

8	4	6	9	1	2	3

$i = 0$ <-------------------------- j

8	4	6	1	9	2	3

$i = 0$ <------------------j

Bubble Sort

8	4	6	9	2	3	1

$i = 0$ <-- j

8	4	6	9	2	1	3

$i = 0$ <-------------------------------------- j

8	4	6	9	1	2	3

$i = 0$ <----------------------------- j

8	4	6	1	9	2	3

$i = 0$ <--------------------j

8	4	1	6	9	2	3

$i = 0$ <-----------j

Bubble Sort

8	4	6	9	2	3	1

$i = 0$ <-- j

8	4	6	9	2	1	3

$i = 0$ <---------------------------------- j

8	4	6	9	1	2	3

$i = 0$ <------------------------- j

8	4	6	1	9	2	3

$i = 0$ <------------------j

8	4	1	6	9	2	3

$i = 0$ <----------j

8	1	4	6	9	2	3

$i = 0$ <---j

Bubble Sort

8	4	6	9	2	3	1

$i = 0$ <-- j

8	4	6	9	2	1	3

$i = 0$ <---------------------------------- j

8	4	6	9	1	2	3

$i = 0$ <------------------------- j

8	4	6	1	9	2	3

$i = 0$ <-----------------j

8	4	1	6	9	2	3

$i = 0$ <----------j

8	1	4	6	9	2	3

$i = 0$ <---j

1	8	4	6	9	2	3

$i = 0$ <---j

Bubble Sort

1	8	4	6	9	2	3

$i = 1$

Bubble Sort

1	8	4	6	9	2	3

$i = 1$

1	8	4	6	9	2	3

$i = 1$ <---------------------------------- j

Bubble Sort

1	8	4	6	9	2	3

$i = 1$

1	8	4	6	9	2	3

$i = 1$ <-----------------------------------j

1	8	4	6	9	2	3

$i = 1$ <---------------------------j

Bubble Sort

1	8	4	6	9	2	3

$i = 1$

1	8	4	6	9	2	3

$i = 1$ <----------------------------------j

1	8	4	6	9	2	3

$i = 1$ <--------------------------j

1	8	4	6	2	9	3

$i = 1$ <-----------------j

Bubble Sort

1	8	4	6	9	2	3

$i = 1$

1	8	4	6	9	2	3

$i = 1$ <-------------------------------------j

1	8	4	6	9	2	3

$i = 1$ <----------------------------j

1	8	4	6	2	9	3

$i = 1$ <-------------------j

1	8	4	2	6	9	3

$i = 1$ <-----------j

Bubble Sort

1	8	4	6	9	2	3

$i = 1$

1	8	4	6	9	2	3

$i = 1$ <------------------------------------j

1	8	4	6	9	2	3

$i = 1$ <----------------------------j

1	8	4	6	2	9	3

$i = 1$ <-------------------j

1	8	4	2	6	9	3

$i = 1$ <-----------j

1	8	2	4	6	9	3

$i = 1$ <---j

Bubble Sort

1	8	4	6	9	2	3

$i = 1$

1	8	4	6	9	2	3

$i = 1$ <-----------------------------------j

1	8	4	6	9	2	3

$i = 1$ <---------------------------j

1	8	4	6	2	9	3

$i = 1$ <------------------j

1	8	4	2	6	9	3

$i = 1$ <-----------j

1	8	2	4	6	9	3

$i = 1$ <---j

1	2	8	4	6	9	3

$i = 1$ <---j

Bubble Sort

1	2	8	4	6	9	3

$i = 2$

1	2	3	8	4	6	9

$i = 3$

1	2	3	4	8	6	9

$i = 4$

1	2	3	4	6	8	9

$i = 5$

Bubble Sort

BUBBLESORT(A, n)

for $i = 1$ **to** $n - 1$
    for $j = n$ **downto** $i + 1$
        if $A[j] < A[j - 1]$
            exchange $A[j]$ with $A[j - 1]$

Content
Bubble Sort
Analyzing Bubble Sort
Time Complexity Analysis of Loops
Exercises

Analyzing Bubble Sort

- The outer loop runs at most n times.

```
BUBBLESORT(A, n)
for i = 1 to n − 1                          c₁ × n
    for j = n downto i + 1
        if A[j] < A[j − 1]
            exchange A[j] with A[j − 1]
```

Analyzing Bubble Sort

- The inner loop runs from n to $i + 1$.

```
BUBBLESORT(A, n)
for i = 1 to n − 1
    for j = n downto i + 1
        if A[j] < A[j − 1]
            exchange A[j] with A[j − 1]
```

$c_1 \times n$

$$c_2 \times \sum_{i}^{n} n - i$$

Analyzing Bubble Sort

- The inner loop runs from n to $i + 1$.

BUBBLESORT(A, n)

for $i = 1$ **to** $n - 1$
 for $j = n$ **downto** $i + 1$
  if $A[j] < A[j-1]$
   exchange $A[j]$ with $A[j-1]$

$$c_1 \times n$$

$$c_2 \times \sum_{i}^{n} n - i$$

$$c_3 \times \sum_{i}^{n} n - i - 1$$

$$c_4 \times \sum_{i}^{n} n - i - 1$$

Analyzing Bubble Sort

- The inner loop runs from n to $i + 1$.
- The lowest value for i is 1, thus the inner loop runs at most n times.

BUBBLESORT(A, n)

for $i = 1$ **to** $n - 1$ — $c_1 \times n$

  for $j = n$ **downto** $i + 1$ — $c_2 \times \sum_{i}^{n} n - i$

    if $A[j] < A[j-1]$ — $c_3 \times \sum_{i}^{n} n - i - 1$

      exchange $A[j]$ with $A[j-1]$ — $c_4 \times \sum_{i}^{n} n - i - 1$

Analyzing Bubble Sort

- The inner loop runs from n to $i + 1$.
- The lowest value for i is 1, thus the inner loop runs at most n times.

```
BUBBLESORT(A, n)
1  for i = 1 to n − 1
2      for j = n downto i + 1
3          if A[j] < A[j − 1]
4              exchange A[j] with A[j − 1]
```

$c_1 \times n$

$c_2 \times \sum_{i}^{n} n - i$

$c_3 \times \sum_{i}^{n} n - i - 1$

$c_4 \times \sum_{i}^{n} n - i - 1$

$$c_1 n + c_2 \sum_{i}^{n} n - i + c_3 \sum_{i}^{n} n - i - 1 + c_4 \sum_{i}^{n} n - i - 1$$

Analyzing Bubble Sort

- The inner loop runs from n to $i + 1$.
- The lowest value for i is 1, thus the inner loop runs at most n times.

BUBBLESORT(A, n)

```
for i = 1 to n − 1
    for j = n downto i + 1
        if A[j] < A[j − 1]
            exchange A[j] with A[j − 1]
```

$c_1 \times n$

$c_2 \times \sum_{i}^{n} n - i$

$c_3 \times \sum_{i}^{n} n - i - 1$

$c_4 \times \sum_{i}^{n} n - i - 1$

$$c_1 n + c_2 \sum_{i}^{n} n - i + c_3 \sum_{i}^{n} n - i - 1 + c_4 \sum_{i}^{n} n - i - 1$$

$$\sum_{i=1}^{n} (n - i) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i$$

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

Analyzing Bubble Sort

- The inner loop runs from n to $i + 1$.
- The lowest value for i is 1, thus the inner loop runs at most n times.

BUBBLESORT(A, n)

```
for i = 1 to n − 1
    for j = n downto i + 1
        if A[j] < A[j − 1]
            exchange A[j] with A[j − 1]
```

$c_1 \times n$

$c_2 \times \sum_{i}^{n} n - i$

$c_3 \times \sum_{i}^{n} n - i - 1$

$c_4 \times \sum_{i}^{n} n - i - 1$

$$T(n) = \Theta(n^2) \text{ for all cases}$$

Content
Bubble Sort
Analyzing Bubble Sort
Time Complexity Analysis of Loops
Exercises

Time Complexity Analysis of Loops

- There are two common loop patterns that appear in our algorithms:
- Single loop:
  - a constant time loop,
  - a loop that runs n times,
  - a loop that grows exponentially,
  - a loop that runs based on a specific condition,
  - a loop that runs with a data structure,
  - consecutive single loops, etc.
- Nested loop: one or more loop inside another loop.

Time Complexity Analysis of Loops

- Single for and while loop running <u>constant times</u>: $O(1)$

```
for (int i = 1; i <= c; i = i + 1)
{
	// some O(1) operation
}
```

```
int i = 1;
while (i <= c)
{
	// some O(1) operation
	i = i + 1;
}
```

Time Complexity Analysis of Loops

- Single for loop running $\underline{n\text{ times}}$ and incrementing or decrementing by a constant: $O(n)$

```
for (int i = 1; i <= n; i = i + c)
{
    // some O(1) operation
}
```

```
for (int i = n; i > 0; i = i - c)
{
    // some O(1) operation
}
```

```
int i = 1;
while (i <= n)
{
    // some O(1) operation
    i = i + c;
}
```

```
int i = n;
while (i > 0)
{
    // some O(1) operation
    i = i - c;
}
```

Time Complexity Analysis of Loops

- Single for and while loop running constant multiple of n times: $O(n)$

```
l = 0, r = n - 1
while (l <= r)
{
        if (some condition)
        {
                // some O(1) operation
                 l = l + 1
        }
        else
        {
                // some O(1) operation
                r = r - 1
        }
        // some O(1) operation
}
```

```
for (int l = 0, r = n - 1; l <= r; )
{
        if (some condition)
        {
                // some O(1) operation
                l = l + 1;
        }
        else
        {
                // some O(1) operation
                r = r - 1;
        }
        // some O(1) operation
}
```

Time Complexity Analysis of Loops

- A single for and while loop incrementing or decrementing by a constant factor: $O(\log n)$

```
int i = 1;
while (i < n)
{
	// some O(1) operation
	i = i * 2;
}
```

```
int i = n;
while (i > 0)
{
	// some O(1) operation
	i = i / 2;
}
```

```
for (int i = 1; i < n; i = i*2)
{
	// some O(1) operation
}
```

```
for (int i = n; i > 0; i = i/2)
{
	// some O(1) operation
}
```

Time Complexity Analysis of Loops

- Single for and while loop incrementing by some constant power: $O(\log(\log n))$

```
int i = 2;
while (i <= n)
{
	// some O(1) operation
	i = pow(i, c);
}
```

```
// Here c is a constant greater than 1
for (int i = 2; i < = n; i = pow(i, c))
{
	// some O(1) operation
}
```

Time Complexity Analysis of Loops

- <u>Consecutive single loops</u>: $O(m + n)$

```
for (int i = o; i < m; i = i + 1)
{
	// some O(1) operation
}

for (int i = 0; i < n; i = i + 1)
{
	// some O(1) operation
}
```

Time Complexity Analysis of Loops

- Two nested for and while loops running n times each: $O(n^2)$

```
for (int i = 0; i < n; i = i + 1)
{
	for (int j = 0; j < n; j = j + 1)
	{
	// some O(1) operation
	}
}
```

```
int i = 0;
while (i < n)
{
	int j = 0;
	while (j < n)
	{
		// some O(1) operation
		j = j + 1;
	}
	i = i + 1;
}
```

Time Complexity Analysis of Loops

- Three nested for and while loops running n times each: $O(n^3)$

```
for (int i = 0; i < m; i = i + 1)
{
        for (int j = 0; j < n; j = j + 1)
        {
                for (int k = 0; k < n; k = k + 1)
                {
                        // some O(1) operation
                }
        }
}
```

```
int i = 0;
while (i < m) {
        int j = 0;
        while (j < n)
        {
                int k = 0;
                while (k < n)
                {
                        // some O(1) operation
                        k = k + 1;
                }
                j = j + 1;
        }
        i = i + 1;
}
```

Content
Bubble Sort
Analyzing Bubble Sort
Time Complexity Analysis of Loops
Exercises

Exercises

Algorithm 16: A(A, p, r)

Input: Array A, parameters p, r
Output: Variable sum

$x \leftarrow 5$;
$sum \leftarrow 0$;
if $x > 10$ **then**
    $sum \leftarrow 10$;
end
else
    $sum \leftarrow 5$;
end

Exercises

Algorithm 16: A(A, p, r)

Input: Array A, parameters p, r
Output: Variable sum

$x \leftarrow 5$; $\longrightarrow$ c_1
$sum \leftarrow 0$; $\longrightarrow$ c_2
if $x > 10$ **then** $\longrightarrow$ c_3
  $sum \leftarrow 10$; $\longrightarrow$ c_4
end
else
  $sum \leftarrow 5$; $\longrightarrow$ c_5
end

Since the algorithm runs independently of any input size or any other variable,

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 = O(1)$$

Exercises

Algorithm 17: A(A, p, r)

Input: Array A, parameters p, r
Output: Variable sum
for $i \leftarrow 1$ **to** n **do**
    $sum \leftarrow i$;
end

Exercises

Algorithm 17: A(A, p, r)

Input: Array A, parameters p, r
Output: Variable sum
for $i \leftarrow 1$ **to** n **do** $\longrightarrow c_1 \times n$
    $sum \leftarrow i$; $\longrightarrow c_2 \times (n-1)$
end

The algorithm runs from 1 … n.
There is one loop, dependent on n.

$$T(n) = c_1 n + c_2(n-1)$$
$$= O(n)$$

Exercises

Algorithm 18: A(A, p, r)

Input: Array A, parameters p, r

Output: Variable sum

```
for i ← 1 to n do
    for j ← 1 to n do
        sum ← i + j ;
    end
end
```

Exercises

Algorithm 18: A(A, p, r)

Input: Array A, parameters p, r

Output: Variable sum

for $i \leftarrow 1$ **to** n **do** $\longrightarrow c_1 \times n$

  for $j \leftarrow 1$ **to** n **do** $\longrightarrow c_2 \times n \times (n-1)$

    $sum \leftarrow i + j$; $\longrightarrow c_3 \times n \times (n-1)$

  end

end

There are two loops:

- The outer loop runs from 1 … n
- The second is independent of the first. it runs from 1 … n.
  Since it is an inner loop, it will be run for $n(n-1)$ times.

$$T(n) = c_1 n + c_2 n(n-1) + c_3 n(n-1) = O(n^2)$$

Exercises

Algorithm 19: A()

Output: Variables i and s

$i \leftarrow 1$;

$s \leftarrow 1$;

while $s \leq n$ **do**

  $i \leftarrow i + 1$;

  $s \leftarrow s + i$;

end

Exercises

Algorithm 19: A()

Output: Variables i and s

$i \leftarrow 1$; $\longrightarrow c_1$

$s \leftarrow 1$; $\longrightarrow c_2$

while $s \leq n$ **do**

  $i \leftarrow i + 1$;

  $s \leftarrow s + i$;

end

$$\sum_{i=1}^{k} i = 1 + 2 + 3 + \dots + k = \frac{k(k+1)}{2}$$

i = 1, 2, 3, 4, 5, 6, … , k

s = 1, 3, 6, 10, 15, 20, … , n

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

$$k^2 + k = 2n$$

$$k^2 \approx 2n$$

$$k = \sqrt{n}$$

$$\therefore T(n) = \sqrt{n}$$

Exercises

Algorithm 20: A()

Input: Integer n
Output: Variable sum
for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** i **do**
        $sum \leftarrow i + j$;
    end
end

Exercises

Algorithm 20: A()

Input: Integer n
Output: Variable sum
for $i \leftarrow 1$ **to** n **do** → $c_1 \times n$
  for $j \leftarrow 1$ **to** i **do** → $c_2 \times \sum_{j=1}^{i} t_i$
    $sum \leftarrow i + j$; → $c_3 \times \sum_{j=1}^{i} t_i - 1$
  end
end

$$T(n) = c_1 n + c_2 \sum_{j=1}^{i} t_i + c_3 \sum_{j=1}^{i} t_i - 1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$

$$\therefore T(n) = O(n^2)$$

Exercises

Algorithm 21: A()

Input: Integer n
Output: Variable sum

for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** i^2 **do**
        for $k \leftarrow 1$ **to** $\frac{n}{2}$ **do**
            $sum \leftarrow i + j + k$;
        end
    end
end

Exercises

Algorithm 21: A()

Input: Integer n

Output: Variable sum

for $i \leftarrow 1$ **to** n **do** $\rightarrow$ $c_1 \times n$

  for $j \leftarrow 1$ **to** i^2 **do** $\rightarrow$ $c_2 \times \sum_{j=1}^{i^2} n - 1$

    for $k \leftarrow 1$ **to** $\frac{n}{2}$ **do** $\rightarrow$ $c_3 \times \sum_{j=1}^{\frac{n}{2}} \sum_{j=1}^{i^2} n - 2$

      $sum \leftarrow i + j + k$; $\rightarrow$ $c_4 \times \frac{n}{2} \sum_{j=1}^{n} i^2$

    end

  end

end

$$T(n) = Total\ Iterations = \sum_{i=1}^{n} \sum_{j=1}^{i^2} \sum_{k=1}^{\frac{n}{2}} 1$$

$$\sum_{k=1}^{\frac{n}{2}} 1 = O(n)$$

$$\sum_{j=1}^{i^2} O(n) = O(n \cdot i^2) = O(n^3)$$

$$\sum_{i=1}^{n} O(n^3) = O(n \cdot n^3) = O(n^4)$$

$$T(n) = O(n^4)$$

Exercises

Algorithm 22: A()

Output: Variable sum

for $i \leftarrow 1$ **to** n **by** $i \leftarrow i \times 2$ **do**

  $sum \leftarrow i$;

end

Exercises

Algorithm 22: A()

Output: Variable *sum*

for $i \leftarrow 1$ **to** n **by** $i \leftarrow i \times 2$ **do** $c_1 \times \log n$

  $sum \leftarrow i$; $c_2 \times \log n - 1$

end

$$
\begin{array}{lccccccc}
i = & 1 & 2 & 4 & 8 & 16 & \cdots & n \\
 & 2^0 & 2^1 & 2^2 & 2^3 & 2^4 & \cdots & 2^k
\end{array}
$$

$$2^k = n$$

$$\lg 2^k = \lg n$$

$$k = \lg n$$

$$T(n) = c_1 \log n + c_2 \log(n-1) = O(\log n)$$

Exercises

Algorithm 23: A()

Output: Variable *sum*

for $i \leftarrow \frac{n}{2}$ **to** n **do**

    for $j \leftarrow 1$ **to** $\frac{n}{2}$ **do**

        for $k \leftarrow 1$ **to** n **by** $k \leftarrow k \times 2$ **do**

            $sum \leftarrow i + j + k$;

        end

    end

end

Exercises

Algorithm 23: A()

Output: Variable *sum*

for $i \leftarrow \frac{n}{2}$ **to** n **do** $c_1 \times n$

  for $j \leftarrow 1$ **to** $\frac{n}{2}$ **do** $c_2 \times n \times (n-1)$

    for $k \leftarrow 1$ **to** n **by** $k \leftarrow k \times 2$ **do** $c_2 \times n \times (n-1) \times \log n$

      $sum \leftarrow i + j + k$;

    end

  end

end

$$T(n) = O(n^2 \log n)$$

Exercises

Algorithm 24: A()

```
Output: Variable n
while n > 1 do
    n ← n/2 ;
end
```

Exercises

Algorithm 24: A()

Output: Variable n
while $n > 1$ **do** $c_1 \times \log n$
  $n \leftarrow \frac{n}{2}$;
end

The loop goes runs:

$$n, \quad \frac{n}{2}, \quad \frac{n}{4}, \quad \frac{n}{8}, \quad \ldots, \quad 1$$

$$T(n) = O(\log n)$$

Exercises

Algorithm 25: Avg(A, n)

Input: Array A, Integer n
Output: Float Average
$Sum \leftarrow 0$;
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $Sum \leftarrow Sum + A[i]$;
end
return Sum/n ;

Exercises

Algorithm 25: Avg(A, n)

Input: Array A, Integer n

Output: Float Average

$Sum \leftarrow 0$; $\longrightarrow c_1$

for $i \leftarrow 0$ **to** $n - 1$ **do** $\longrightarrow c_2 \times n$

  $Sum \leftarrow Sum + A[i]$; $\longrightarrow c_3 \times (n - 1)$

end

return Sum/n ;

$$T(n) = O(n)$$

Exercises

Algorithm 26: LSUM(N)

Input: Integer N
Output: Integer Sum

```
Sum ← 0 ;
for i ← 1 to n − 1 do
    for j ← N to 1 by j ← j/2 do
        Sum ← Sum + j ;
    end
end
return Sum ;
```

Exercises

Algorithm 26: LSUM(N)

Input: Integer N
Output: Integer Sum

$Sum \leftarrow 0$; $\longrightarrow c_1$
for $i \leftarrow 1$ **to** $n-1$ **do** $\longrightarrow c_2 \times n$
  for $j \leftarrow N$ **to** 1 **by** $j \leftarrow j/2$ **do** $\longrightarrow c_3 \times n \times \log n$
    $Sum \leftarrow Sum + j$;
  end
end
return Sum ;

$$T(n) = O(n \log n)$$

Exercises

Algorithm 27: SUM(A, n, m)

```
Input: Array A, Integer n, Integer m
Output: Float Average
Sum ← 0 ;
for i ← 0 to n − 1 do
    for j ← 0 to m − 1 do
        Sum ← Sum + A[i][j] ;
    end
end
return Sum / (n×m) ;
```

Exercises

Algorithm 27: SUM(A, n, m)

Input: Array A, Integer n, Integer m
Output: Float Average

$Sum \leftarrow 0$; $\longrightarrow c_1$
for $i \leftarrow 0$ **to** $n-1$ **do** $\longrightarrow c_2 \times n$
  for $j \leftarrow 0$ **to** $m-1$ **do** $\longrightarrow c_2 \times n \times m$
    $Sum \leftarrow Sum + A[i][j]$;
  end
end
return $\frac{Sum}{n \times m}$;

$$T(n) = O(nm)$$

Exercises

Algorithm 28: CRAZY_SORT(A)

Input: Array A
Output: Sorted Array A
if $|A| > 1$ **then**
    CRAZY_SORT(1st third of array A) ;
    CRAZY_SORT(2nd third of array A) ;
    CRAZY_SORT(3rd third of array A) ;
    CRAZY_MERGE(3 sorted thirds of array A) ;
end

Assume the CRAZY_MERGE takes time $\lg(n)$

Exercises

Algorithm 28: CRAZY_SORT(A)

Input: Array A
Output: Sorted Array A
if $|A| > 1$ **then**
  CRAZY_SORT(1st third of array A) ; $\longrightarrow c_1 \times T(n/3)$
  CRAZY_SORT(2nd third of array A) ; $\longrightarrow c_2 \times T(n/3)$
  CRAZY_SORT(3rd third of array A) ; $\longrightarrow c_3 \times T(n/3)$
  CRAZY_MERGE(3 sorted thirds of array A) ;
  $c_4 \times \log(n)$
end

$$T(n) = 3T(n/3) + \lg n$$
$$T(n) = O(n)$$

Exercises

Algorithm 29: Factorial(n)

```
Input: Integer n
Output: Integer Factorial of n
if n = 0 then
    return 1 ;
end
if n = 1 then
    return 1 ;
end
return n × Factorial(n − 1) ;
```

Exercises

Algorithm 29: Factorial(n)

Input: Integer n

Output: Integer Factorial of n

if $n = 0$ **then** → c_1

  return 1 ; → c_2

end

if $n = 1$ **then** → c_3

  return 1 ; → c_4

end

return $n \times$ Factorial($n - 1$) ; $c_5 \times T(n-1)$

$$T(n) = O(n)$$

Exercises

Algorithm 30: dexpo(g, A, p)

```
Input: Integer g, Integer A, Integer p
Output: Result of g^A mod p
if A = 0 then
│  return 1 ;
end
if A is odd then
│  a ← dexpo(g, A − 1, p) ;
│  return (a · g mod p) ;
end
else
│  a ← dexpo(g, A/2, p) ;
│  return (a^2 mod p) ;
end
```

Exercises

Algorithm 30: dexpo(g, A, p)

```
Input: Integer g, Integer A, Integer p
Output: Result of g^A mod p
if A = 0 then                    ⟶ c
  | return 1 ;                   ⟶ c
end
if A is odd then                 ⟶ c
  | a ← dexpo(g, A − 1, p) ;     ⟶ T(n − 1)
  | return (a · g mod p) ;       ⟶ c
end
else
  | a ← dexpo(g, A/2, p) ;       ⟶ T(n/2)
  | return (a^2 mod p) ;         ⟶ c
end
```

$$T(n) = \begin{cases} \theta(1) & if\ n = 0 \\ T(n-1) & if\ n\ is\ odd \\ T(n/2) & if\ n\ is\ even \end{cases}$$

Best Case:

$$\mathrm{T}(n) = T(n/2) + \theta(1)$$

$$\therefore T(n) = O(\lg n)$$

Worst Case:

$$\mathrm{T}(n) = T(n-1) + \theta(1)$$

$$\therefore T(n) = O(n)$$

Further Readings

- Time analysis of common loop patterns in programming: https://medium.com/enjoy-algorithm/analysis-of-loop-in-programming-cc9a644ef8cd