

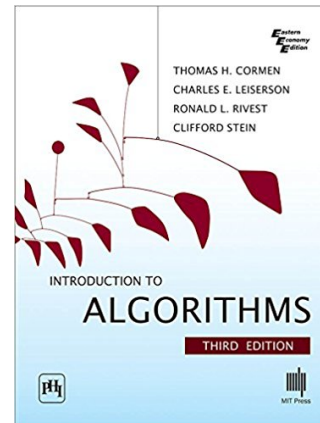
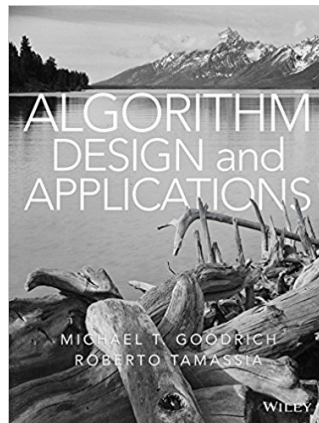
# COMP26120: More on the Complexity of Recursive Programs (2019/20)

Lucas Cordeiro

[lucas.cordeiro@manchester.ac.uk](mailto:lucas.cordeiro@manchester.ac.uk)

# Divide-and-Conquer (Recurrence)

- Textbooks:
  - *Algorithm Design and Applications*, Goodrich, Michael T. and Roberto Tamassia (Chapter 8)
  - *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein (Chapters 2 and 4)



# Intended Learning Outcomes

- Review **exponentials, logarithms, factorials, and sums**
- Solve recurrences using **iteration** method and **master** method
- Describe various examples to analyse **divide-and-conquer** algorithms and how to solve their **recurrences**

# Exponentials

- For all real  $a > 0$ ,  $m$ , and  $n$ , we have the following identities:

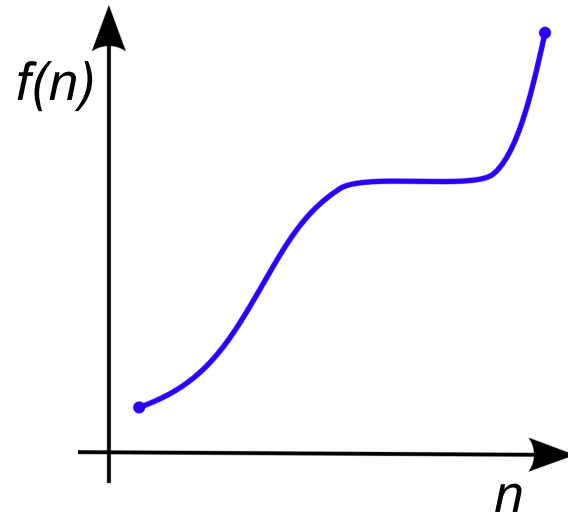
$$a^0 = 1,$$

$$a^1 = a,$$

$$a^{-1} = 1/a,$$

$$(a^m)^n = a^{mn},$$

$$a^m a^n = a^{m+n}.$$



- For all  $n$  and  $a \geq 1$ , the function  $a^n$  is **monotonically increasing** in  $n$

# Logarithms

- For all real  $a > 0$ ,  $b > 0$ ,  $c > 0$ , and  $n$ ,

$$a = b^{\log_b a},$$

$$\log_c(ab) = \log_c a + \log_c b,$$

$$\log_b a^n = n \log_b a,$$

$$\log_b a = \frac{\log_c a}{\log_c b},$$

$$\log_b(1/a) = -\log_b a,$$

$$\log_b a = \frac{1}{\log_a b},$$

$$a^{\log_b c} = c^{\log_b a},$$

in each equation above, logarithm bases are not 1

# Factorials

- The notation  $n!$  is defined for integers  $n \geq 0$  as

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n \cdot (n-1)! & \text{if } n > 0. \end{cases}$$

Thus,  $n! = 1 \cdot 2 \cdot 3 \cdots n$ .

A **weak upper bound** on the factorial function is  $n! \leq n^n$  since each of the  $n$  terms in the factorial product is at most  $n$

- A form of Stirling's approximation

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

# Sums

- Given a sequence of  $a_1, a_2, \dots, a_n$  of numbers, where  $n$  is a **nonnegative integer**, we can write the finite sum  $a_1 + a_2 + \dots + a_n$  as

$$\sum_{k=1}^n a_k .$$

If  $n = 0$ , the value of the summation is 0

- Given an infinite sequence of  $a_1, a_2, \dots, a_n$

$$\lim_{n \rightarrow \infty} \sum_{k=1}^n a_k .$$

If the limit does not exist, the series **diverges**; If the limit exists and is finite, the series **converges**

# Algebraic Regrouping

- **Distributive:** A common factor can be distributed over all the summands

$$\sum_{k \in K} c a_k = c \sum_{k \in K} a_k$$

- **Addition:** Add each pair of summands with the same index

$$\sum_{k \in K} (a_k + b_k) = \sum_{k \in K} a_k + \sum_{k \in K} b_k$$

- **Permutation:** The value of a sum is unchanged by permuting the order of the summands

$$\sum_{k \in K} a_k = \sum_{p(k) \in K} a_{p(k)}$$



# Arithmetic and Geometric Series

- The summation

$$\sum_{k=1}^n k = 1 + 2 + \cdots + n$$

is an **arithmetic series** and has the value

$$\begin{aligned}\sum_{k=1}^n k &= \frac{1}{2}n(n+1) \\ &= \Theta(n^2) .\end{aligned}$$

- For real  $x \neq 1$ , the summation

$$\sum_{k=0}^n x^k = 1 + x + x^2 + \cdots + x^n$$

is a **geometric series** and has the value

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1} .$$

# The Perturbation Method

- The initial step of a **perturbation** is to equate two expressions for  $S_{n+1}$  by taking its first and last terms
- Find  $S_n$  on both sides of the equation
- Isolate  $S_n$  to find the formula for the sum

What makes it interesting is not the theory behind it, but the fact that it works so effectively so often for sums that contain exponentials

# Illustrative Example of the Perturbation Method

- Given our **geometric series**:

$$S_n = \sum_{k=0}^n x^k$$

$$S_{n+1} = S_n + x^{n+1} \quad \text{Obtain the last term}$$

$$S_{n+1} = x^0 + \sum_{k=1}^{n+1} x^k = 1 + \sum_{k=0}^n x^{k+1} = 1 + x \sum_{k=0}^n x^k = 1 + xS_n$$

Obtain the first term

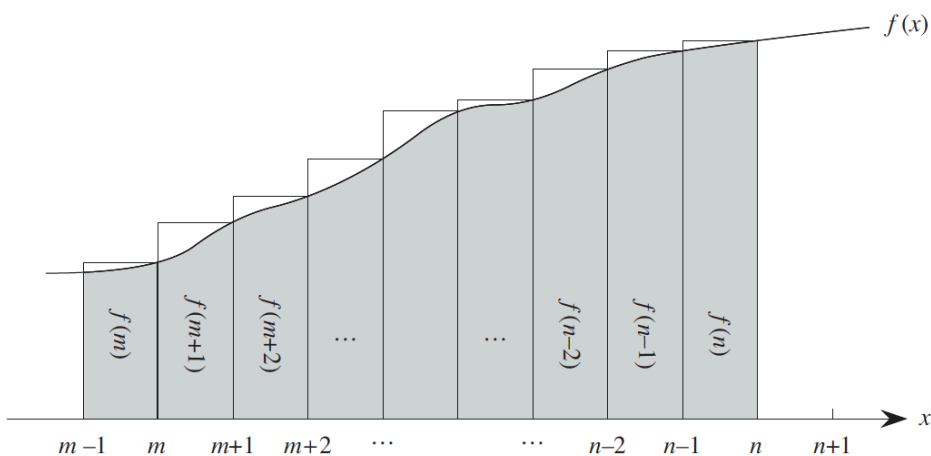
$$1 + xS_n = S_n + x^{n+1}$$

$$S_n = \frac{x^{n+1} - 1}{x - 1}$$

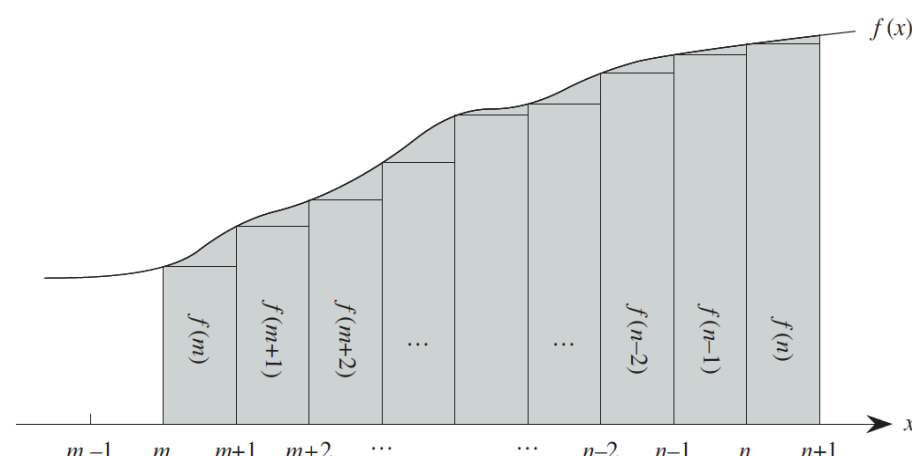
# Approximation by Integrals

- When a summation has the form  $\sum_{k=m}^n f(k)$ , where  $f(k)$  is a **monotonically increasing function**, we can approximate it by integrals

$$\int_{m-1}^n f(x) dx \leq \sum_{k=m}^n f(k) \leq \int_m^{n+1} f(x) dx .$$



$$\int_{m-1}^n f(x) dx \leq \sum_{k=m}^n f(k)$$



$$\sum_{k=m}^n f(k) \leq \int_m^{n+1} f(x) dx$$

# Approximation by Integrals (Cont.)

- When  $f(k)$  is a **monotonically decreasing function**, we can use a similar method to provide the bounds

$$\int_m^{n+1} f(x) dx \leq \sum_{k=m}^n f(k) \leq \int_{m-1}^n f(x) dx .$$

$$\begin{aligned} \sum_{k=1}^n \frac{1}{k} &\geq \int_1^{n+1} \frac{dx}{x} \\ &= \ln(n+1) . \end{aligned}$$

First inequality

$$\begin{aligned} \sum_{k=2}^n \frac{1}{k} &\leq \int_1^n \frac{dx}{x} \\ &= \ln n , \end{aligned}$$

Second inequality

which yields the bound

$$\sum_{k=1}^n \frac{1}{k} \leq \ln n + 1 .$$

# Solving Recurrences

- In addition to the **substitution method**, another option is what the literature calls the “**iteration method**”
  - Expand the recurrence
  - Work some algebra to express as a summation
  - Evaluate the summation
- We will show several examples

# Example 1: Iteration

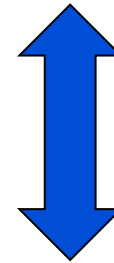
- What is the **complexity** of this recurrence?

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

# Example 1: Iteration

- $s(n) = c + s(n-1)$   
 $= c + c + s(n-2)$   
 $= c + c + c + s(n-3)$   
 $= 3c + s(n-3)$   
...  
 $= kc + s(n-k) = ck + s(n-k)$

$$\begin{aligned} s(n) &= c + s(n-1) \\ s(n-1) &= c + s(n-2) \\ s(n-2) &= c + s(n-3) \end{aligned}$$



$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$



# Example 1: Iteration

- So far for  $n > k$  we have
  - $s(n) = ck + s(n-k)$
- if  $k=n$ ?
  - $s(n) = cn + s(0) = cn$
- Then
  - $s(n) = cn$
  - $s(n) = O(n)$

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

# Example 2: Iteration

- What is the **complexity** of this recurrence?

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

# Example 2: Iteration

- $s(n)$

$$k=1 \quad = n + s(n-1)$$

$$k=2 \quad = n + n-1 + s(n-2)$$

$$k=3 \quad = n + n-1 + n-2 + s(n-3)$$

$$k=4 \quad = n + n-1 + n-2 + n-3 + s(n-4)$$

=

$$= \overset{\vdots}{\dots} \quad \curvearrowright$$
$$= n + n-1 + n-2 + n-3 + \dots + n-(k-1) + s(n-k)$$

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

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

# Example 2: Iteration

- So far for  $n > k$  we have

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

- If  $k=n$  ?

$$\sum_{i=1}^n i + s(0) = \sum_{i=1}^n i + 0 = n \frac{n+1}{2}$$

- Then:

(Arithmetic series)

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

$$s(n) = O(n^2)$$

# Example 3: Iteration

- What is the **complexity** of this recurrence?

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

# Example 3: Iteration

- $T(n) = 2T(n/2) + c$   
     $= 2(2T(n/2/2) + c) + c$   
     $= 2^2T(n/2^2) + 2c + c$   
     $= 2^2(2T(n/2^2/2) + c) + 3c$   
     $= 2^3T(n/2^3) + 4c + 3c$   
     $= 2^3T(n/2^3) + 7c$   
     $= 2^3(2T(n/2^3/2) + c) + 7c$   
     $= 2^4T(n/2^4) + 15c$   
     $= \dots$   
     $= 2^kT(n/2^k) + (2^k - 1)c$

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

# Example 3: Iteration

- So far for  $n > 2^k$  we have
  - $T(n) = 2^k T(n/2^k) + (2^k - 1)c$
- If  $k = \lg n$ ?
  - $T(n) = 2^{\lg n} T(n/2^{\lg n}) + (2^{\lg n} - 1)c$   
 $= n T(n/n) + (n - 1)c$   
 $= n T(1) + (n-1)c$   
 $= nc + (n-1)c = (2n - 1)c$

$$a^{\log_b c} = c^{\log_b a}$$

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

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

# Example 4: Iteration

- What is the **recurrence equation**?

```
MergeSort(A, left, right) {  
    if (left < right) {  
        mid = floor((left + right) / 2);  
        MergeSort(A, left, mid);  
        MergeSort(A, mid+1, right);  
        Merge(A, left, mid, right);  
    }  
}
```

- What is the **complexity** of this recurrence?

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$



# Example 4: Iteration

- $T(n) =$

$$aT(n/b) + cn$$

$$a(aT(n/b/b) + cn/b) + cn$$

$$a^2T(n/b^2) + cna/b + cn$$

$$a^2T(n/b^2) + cn(a/b + 1)$$

$$a^2(aT(n/b^2/b) + cn/b^2) + cn(a/b + 1)$$

$$a^3T(n/b^3) + cn(a^2/b^2) + cn(a/b + 1)$$

$$a^3T(n/b^3) + cn(a^2/b^2 + a/b + 1)$$

...

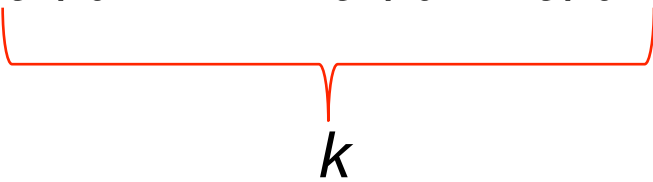
$$a^kT(n/b^k) + cn(a^{k-1}/b^{k-1} + a^{k-2}/b^{k-2} + \dots + a^2/b^2 + a/b + 1)$$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

# Example 4: Iteration

- So we have
  - $T(n) = a^k T(n/b^k) + cn(a^{k-1}/b^{k-1} + \dots + a^2/b^2 + a/b + 1)$
- For  $k = \log_b n$ 
  - $n = b^k$
  - $$\begin{aligned} T(n) &= a^k T(1) + cn(a^{k-1}/b^{k-1} + \dots + a^2/b^2 + a/b + 1) \\ &= a^k c + cn(a^{k-1}/b^{k-1} + \dots + a^2/b^2 + a/b + 1) \\ &= ca^k + cn(a^{k-1}/b^{k-1} + \dots + a^2/b^2 + a/b + 1) \\ &= cn(a^k/n + a^{k-1}/b^{k-1} + \dots + a^2/b^2 + a/b + 1) \\ &= cn(a^k/b^k + a^{k-1}/b^{k-1} + \dots + a^2/b^2 + a/b + 1) \end{aligned}$$

# Example 4: Iteration

- Then, for  $k = \log_b n$ 
  - $T(n) = cn(a^k/b^k + \dots + a^2/b^2 + a/b + 1)$ 
- If  $a = b$ ?
  - $T(n) = cn(k + 1)$   
 $= cn(\log_b n + 1)$   
 $= \Theta(n \log_b n)$

# Example 4: Iteration

- If  $a < b$ ?

$$T(n) = cn(a^k/b^k + \dots + a^2/b^2 + a/b + 1)$$

- Note that:

$$\sum_{n=0}^k x^n = (x^k + x^{k-1} + \dots + x^2 + x + 1) = \frac{x^{k+1} - 1}{x - 1}$$

- We have:

$$\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \dots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \frac{1 - (a/b)^{k+1}}{1 - (a/b)} < \frac{1}{1 - a/b}$$

- Then:

$$T(n) = cn \cdot \Theta(1) = \Theta(n)$$

# Example 4: Iteration

- If  $a > b$ ?

$$T(n) = cn(a^k/b^k + \dots + a^2/b^2 + a/b + 1)$$

$$\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \dots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta\left((a/b)^k\right)$$

- $T(n) = cn \cdot \Theta(a^k / b^k)$   
 $= cn \cdot \Theta(a^{\log_b n} / b^{\log_b n}) = cn \cdot \Theta(a^{\log_b n} / n)$   
since  $a^{\log_b n} = n^{\log_b a}$   
 $= cn \cdot \Theta(n^{\log_b a} / n) = \Theta(cn \cdot n^{\log_b a} / n)$   
 $= \Theta(n^{\log_b a})$

# Example 4: Iteration

- Therefore...

$$T(n) = \begin{cases} \Theta(n) & a < b \\ \Theta(n \log_b n) & a = b \\ \Theta(n^{\log_b a}) & a > b \end{cases}$$

# The Master Theorem

- Given: a **divide and conquer** algorithm
  - An algorithm that divides the problem of size  $n$  into  $a$  subproblems, each of size  $n/b$
  - Let the cost of each stage (i.e., the work to divide the problem  $D(n)$  + combine solved subproblems  $C(n)$ ) be described by the function  $f(n)=D(n)+C(n)$
- The master method provides a “cookbook” method for solving recurrences of the form

# The Master Theorem (cont.)

- if  $T(n) = aT(n/b) + f(n)$  then

$$T(n) = \left\{ \begin{array}{ll} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & \begin{array}{l} f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \text{ AND} \\ af(n/b) \leq cf(n) \text{ for large } n \end{array} \end{array} \right\} \begin{array}{l} \varepsilon > 0 \\ c < 1 \end{array}$$



# The Master Theorem (cont.)

- Used for the solution of this type of recurrences

$$T(n) = aT(n/b) + f(n)$$

- $a \geq 1$ ,  $b > 1$ , and  $f$  asymptotically positive;
- Since  $T(n)$  is the execution time of the algorithm, we can state that
  - Sub-problems of size  $n/b$  are solved recursively each one in time  $T(n/b)$
  - $f(n)$  is the cost to divide the problem and combine its results. In MergeSort we have:

$$T(n) = 2T(n/2) + \Theta(n) \qquad a = 2, b = 2, f(n) = \Theta(n)$$

# Cookbook Method

- **Step 1:** Identify  $a$ ,  $b$  e  $f(n)$
- **Step 2:** Determine  $n^{\log_b a}$
- **Step 3:** Compare  $f(n)$  to  $n^{\log_b a}$  asymptotically
- **Step 4:** According to the case, apply the corresponding rule

# Example 1

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

Step 1:  $a=9, b=3, f(n) = n$

Step 2:  $n^{\log_b a} = n^{\log_3 9} = n^2$

Step 3:  $f(n) = O(n^{\log_3 9 - \varepsilon})$ ,  $O(n)$  where  $\varepsilon=1$

Step 4: case 1 applies:

if  $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$  for some constant  $\varepsilon > 0$ , then  $T(n) = \Theta\left(n^{\log_b a}\right)$

Therefore  $T(n) = \Theta(n^2)$

# Example 2

- $T(n) = T(2n/3) + 1$

Step 1:  $a = 1, b = 3/2, f(n) = 1$

Step 2:  $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Step 3:  $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$

Step 4: case 2 applies:

if  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \log n)$

Therefore  $T(n) = \Theta(\log n)$

# Example 3

- $T(n) = 3T(n/4) + n \log n$

Step 1:  $a = 3, b = 4, f(n) = n \log n$

Step 2:  $n^{\log_b a} = n^{\log_4 3} = n^{0.793}$

Step 3:  $f(n) = \Omega(n^{\log_4 3 + \varepsilon})$ , where  $\varepsilon \approx 0.2$

Step 4: case 3 applies:

*If  $f(n) = \Omega(n \log_b^{a+\varepsilon})$  for some constant  $\varepsilon > 0$ , and if  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$*

## Example 3 (Cont.)

If  $f(n) = \Omega(n \log_b^{a+\varepsilon})$  for some constant  $\varepsilon > 0$ , and if  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$

- For sufficiently large  $n$ , we have that
  - $af(n/b) = 3(n/4) \log(n/4) \leq (3/4) n \log n$
  - for  $c = 3/4$

Therefore  $T(n) = \Theta(n \log n)$

# Example 4 (Merge Sort)

$$T(n) = 2T(n/2) + \Theta(n)$$

$$a = 2, b = 2; n^{\log_b a} = n^{\log_2 2} = n = \Theta(n)$$

Given  $f(n) = \Theta(n)$

We have case 2:

if  $f(n) = \Theta\left(n^{\log_b a}\right)$ , then  $T(n) = \Theta\left(n^{\log_b a} \log n\right)$

which leads to:

$$T(n) = \Theta\left(n^{\log_b a} \log n\right) = \Theta\left(n \log n\right)$$

# Exercises

- What is the **recurrence equation** of these programs?

```
fac(n)
  if n = 1 return 1
  else return n * fac(n-1)
```

```
smallest(list, start, end)
  if start ≥ end-1 return list[start]
  else
    mid = ⌊(start+end)/2⌋
    small1 = smallest(list, start, mid)
    small2 = smallest(list, mid+1, end)
    return smallest of small1 and small2
```



# Exercises

- What is the **recurrence equation** of these programs?

```
index binsearch(number n, index low, index high, const  
keytype S[], keytype x)  
  if low ≤ high then mid = (low + high) / 2  
  if x = S[mid] then return mid  
  elseif x < s[mid] then return binsearch(n, low, mid-1, S, x)  
  else return binsearch(n, mid+1, high, S, x)  
  else return 0  
end binsearch
```

# Exercises

- Use the substitution, iteration or master method to solve the following recurrences:

*a)*  $T(n) = T(n-1) + O(1)$  (Factorial)

*b)*  $T(n) = 2T(n/2) + O(1)$  (Smallest)

*c)*  $T(n) = T(n/2) + \Theta(1)$  (Binary Search)

# Summary

- Many useful algorithms are **recursive** in structure:
  - they call themselves recursively one or more times to deal with closely related sub-problems
- We also analysed recurrences using the following methods:
  - substitution
  - Iteration
  - master
- We have demonstrated the recurrence equation and its complexity for factorial, smallest, and binary search