SIT221: Data Structures and Algorithms

Lecture 1: Algorithm Complexity Analysis

Unit team

- Unit chair
  - Dr Duc Thanh Nguyen
    - Email: duc.nguyen@deakin.edu.au
    - Phone: +61 3 5247 9506
    - Website: ducthanhnguyen.weebly.com

- Lecturers
  - Burwood campus
    - Dr Duc Thanh Nguyen
  - Geelong campus
    - Prof Jemal Abawajy
      - Email: jemal.abawajy@deakin.edu.au
      - Phone: +61 3 5227 1376

Unit team (cont.)

- Tutors
  - Burwood
    - Dr Duc Thanh Nguyen
      - Labs: Tue 8 – 9:50 & 14 – 15:50
      - Email: duc.nguyen@deakin.edu.au
    - Dr Eureka Priyadarshani
      - Labs: Thu 12 – 13:50 & 18 – 19:50
      - Email: eureka.priyadarshani@deakin.edu.au
    - Farzaneh Pakzad
      - Labs: Tue 17 – 18:50
      - Email: far.pakzad@gmail.com
    - Vikrant Patel
      - Labs: Tue 16 – 17:50, Thu 10 – 11:50
      - Email: patelv@deakin.edu.au
  - Geelong
    - Prof Jemal Abawajy
      - Labs: Wed 12 – 13:50 & 14 – 15:50
      - Email: jemal.abawajy@deakin.edu.au

Learning objectives

- Explain the principles and operation of major data structures and algorithms and their influence on the algorithmic complexity of solutions.
- Solve programming problems using major data structures and algorithms in an object-oriented context using the C# programming language.
- Understand the kind of information that is provided in library documentation and be able to write and produce similar documentation for solutions.

What is this unit all about?

- What is the best (most efficient) way to represent/structure your program data?
- What are the key algorithm design paradigms?
- How to assess the efficiency of a given alogrithm?

Unit Materials

- Materials available in CloudDeakin:
  - Unit guide
  - News
  - Workbook – Please use it as your textbook
  - Lecture slides
  - Pracs

Textbooks

- No particular books required
- Recommendation
  - "Visual C#: How to Program" prescribed for the pre-requisite unit has some coverage of data structures and algorithms
  - A textbook is available online: http://www.brpreiss.com/books/opus6/
    - Link in CloudDeakin (Admin Info -> Unit Textbooks)
  - May also be useful to refer to some books covering data structures in other languages
    - Books using C++ or Java are probably the closest

Assessment

- 10% – Weekly practical tasks
  - See CloudDeakin for further information including submission instructions
- 30% – Two programming projects (each is worth 15%)
  - See CloudDeakin for further information including submission instructions
- 60% – Examination
  - To pass you only require an overall mark of 50
- Contact the unit chair if you have **<u>any</u>** concerns about satisfying the requirements of the unit as soon as possible

Plagiarism

- Plagiarism is the copying of another person's ideas or expressions without appropriate acknowledgment and presenting these ideas or forms of expression as your own.
- It includes not only written works such as books or journals but data or images that may be presented in tables, diagrams, designs, plans, photographs, film, music, formulae, web sites and computer programs.
- Plagiarism also includes the use of (or passing off) the work of lecturers or other students as your own.

Plagiarism (cont.)

- Please be aware that if the Faculty Academic Progress and Discipline Committee finds a student has committed an act of academic misconduct it may impose one or more of the following penalties
  - Allocate a zero mark or result or other appropriate mark or result for the assessment task;
  - Allocate a zero mark or result or other appropriate mark or result for the Unit;
  - Suspend from a Unit or a Course for up to 4 Study Periods;
  - Exclude from the University;
  - Pay the cost of investigating the misconduct;
  - Require the Student to refrain from association with specified person/s for purposes of study or assessment;
  - Reprimand and caution the student;
  - Require resubmission of one or more assessment tasks;
  - Require a student to undertake alternative assessment for the Unit on terms determined by the faculty committee;
  - Terminate candidature; Recommend to the vice-chancellor or nominee that the degree not be awarded

For this Unit

- Working with other students
  - You may discuss/collaborate with other students to better understand a problem and to determine an approach to solving the problem.
  - You are not permitted to share your solutions (whether finished or in progress) with other students under any circumstances.
  - Your assignment submission, i.e., code, documentation, etc., must be entirely your own work.
- Referencing sources
  - Any code that has been copied/adapted should be clearly referenced (including any code from assignment questions)
    - Note there should be little need for this anyway, otherwise you are not learning the content well enough to pass

Outline

- Algorithm analysis
- Sorting and searching
- Linked lists
- Stacks and queues
- Hash tables
- Trees
- Graphs
- Advanced algorithms on trees and graphs
- Dynamic programming
- Greedy algorithms
- Unit revisited

Algorithm analysis

Why algorithm analysis?

- Machines do not really dominate the efficiency
  - Example
    - Sort an array of 1 million elements.
    - Two algorithms
      - A => complexity = 2 * (number of elements)^2,
      - B => complexity = 50* number of elements * log(number of elements)
    - Two machines
      - C1 (1 billion instructions per second), runs algorithm A
      - C2 (10 millions instructions per second), runs algorithm B
    - Time to sort 1 million elements on C1 = $2 * (10^6)^2 / 10^9$) = 2000 seconds
    - Time to sort 1 million elements on C2 = $50 * 10^6 * \log(10^6) / 10^7$ = 100 seconds !!!
- How efficient is my algorithm? How to improve?
- Given two algorithms A, B; which one is more efficient?

Still not convinced?!

- Watch Jeffrey Dean talk here: https://www.youtube.com/watch?v=modXC5IWTJI

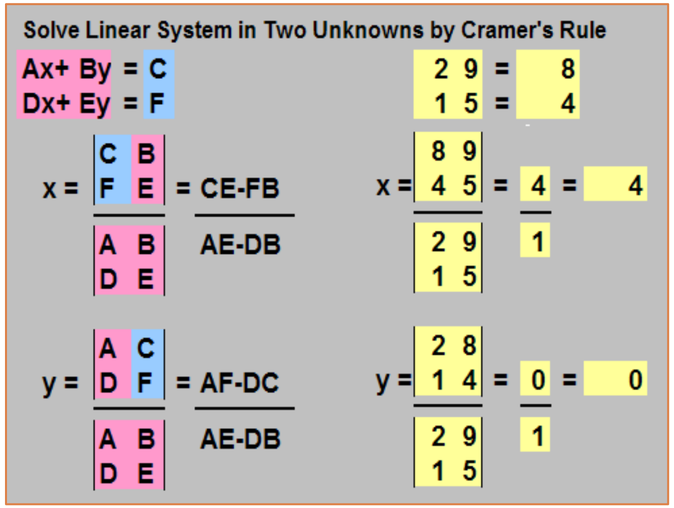
- Who is this guy? http://research.google.com/pubs/jeff.html

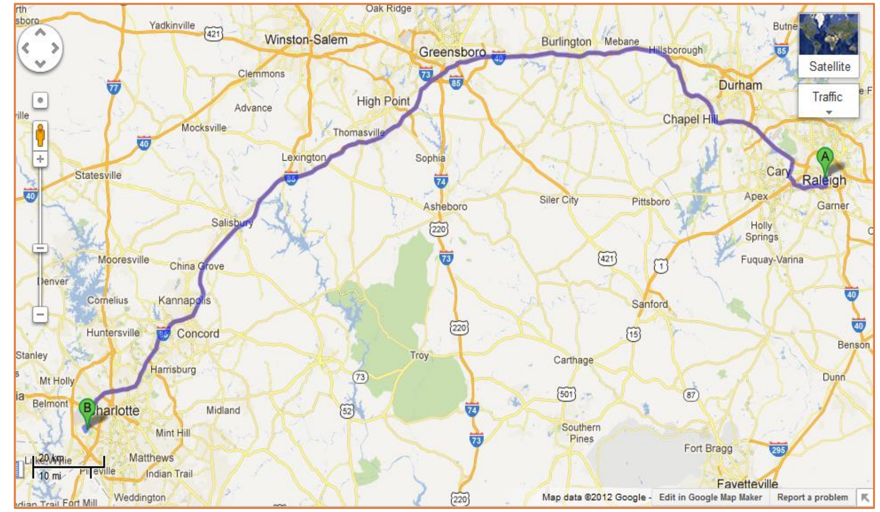
What are we going to cover today?

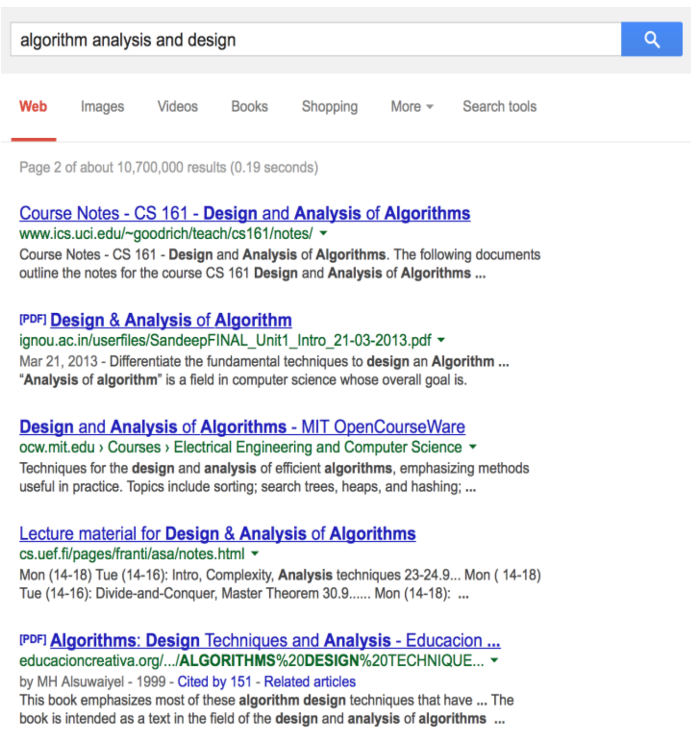
- What does algorithm mean?
- How can I write an algorithm?
- What attributes should I consider?
- How can we analyze algorithms?

Algorithm

A well-defined sequence of computational steps that takes a set of values as input and produces a set of values as output.






algorithm analysis and design

Web Images Videos Books Shopping More Search tools

Page 2 of about 10,700,000 results (0.19 seconds)

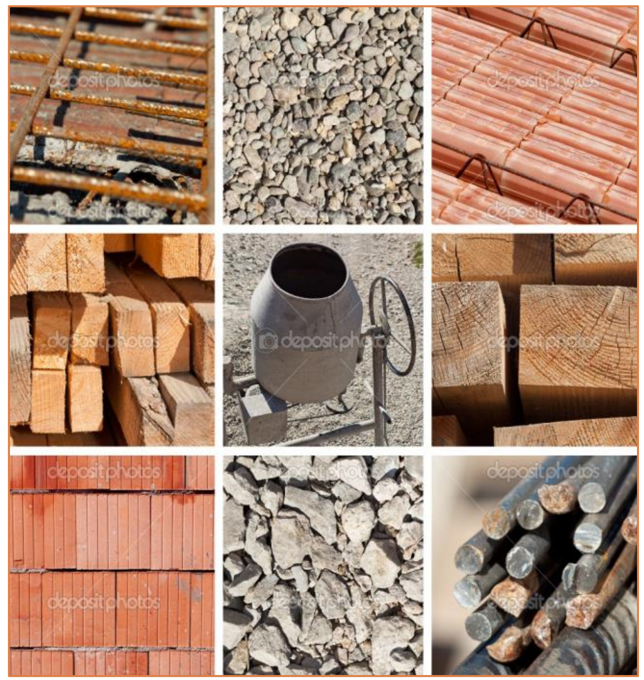
Course Notes - CS 161 - Design and Analysis of Algorithms
www.ics.uci.edu/~goodrich/teach/cs161/notes/
Course Notes - CS 161 - **Design** and **Analysis** of **Algorithms**. The following documents outline the notes for the course CS 161 **Design** and **Analysis** of **Algorithms** ...

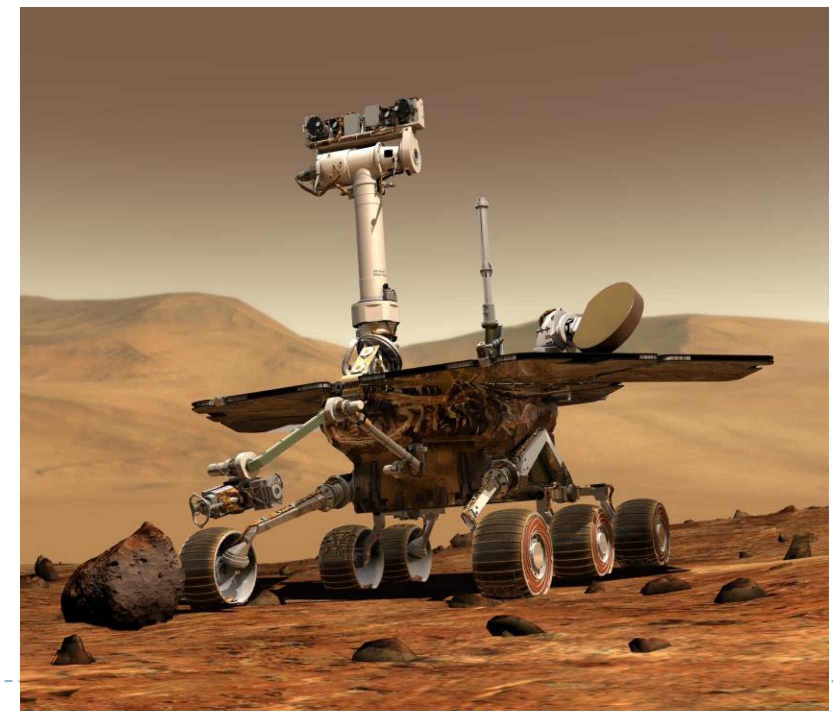
[PDF] Design & Analysis of Algorithm
ignou.ac.in/userfiles/SandeepFINAL_Unit1_Intro_21-03-2013.pdf
Mar 21, 2013 - Differentiate the fundamental techniques to **design** an **Algorithm** ...
"**Analysis** of **algorithm**" is a field in computer science whose overall goal is.

Design and Analysis of Algorithms - MIT OpenCourseWare
ocw.mit.edu › Courses › Electrical Engineering and Computer Science
Techniques for the **design** and **analysis** of efficient **algorithms**, emphasizing methods useful in practice. Topics include sorting; search trees, heaps, and hashing; ...

Lecture material for Design & Analysis of Algorithms
cs.uef.fi/pages/franti/asa/notes.html
Mon (14-18) Tue (14-16): Intro, Complexity, **Analysis** techniques 23-24.9... Mon (14-18) Tue (14-16): Divide-and-Conquer, Master Theorem 30.9...... Mon (14-18): ...

[PDF] Algorithms: Design Techniques and Analysis - Educacion ...
educacioncreativa.org/.../ALGORITHMS%20DESIGN%20TECHNIQUE...
by MH Alsuwaiyel - 1999 - Cited by 151 - Related articles
This book emphasizes most of these **algorithm design** techniques that have ... The book is intended as a text in the field of the **design** and **analysis** of **algorithms** ...





How can I write algorithm?

- Pseudo-code

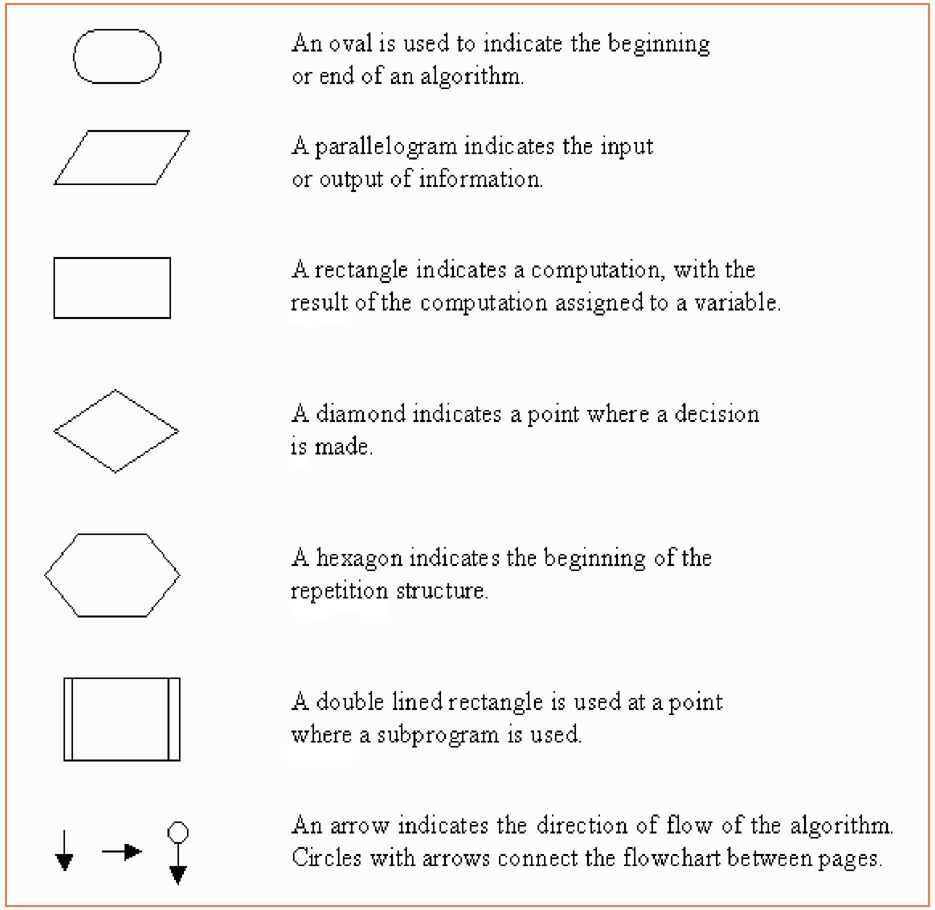
Pseudo-code keywords

START/BEGIN
INPUT, GET
PRINT, DISPLAY, WRITE
WHILE…ENDWHILE
FOR…ENDFOR
DO…UNTIL
IF- THEN…ELSE…ENDIF
CASE…BREAK…ENDCASE
RETURN
STOP/END

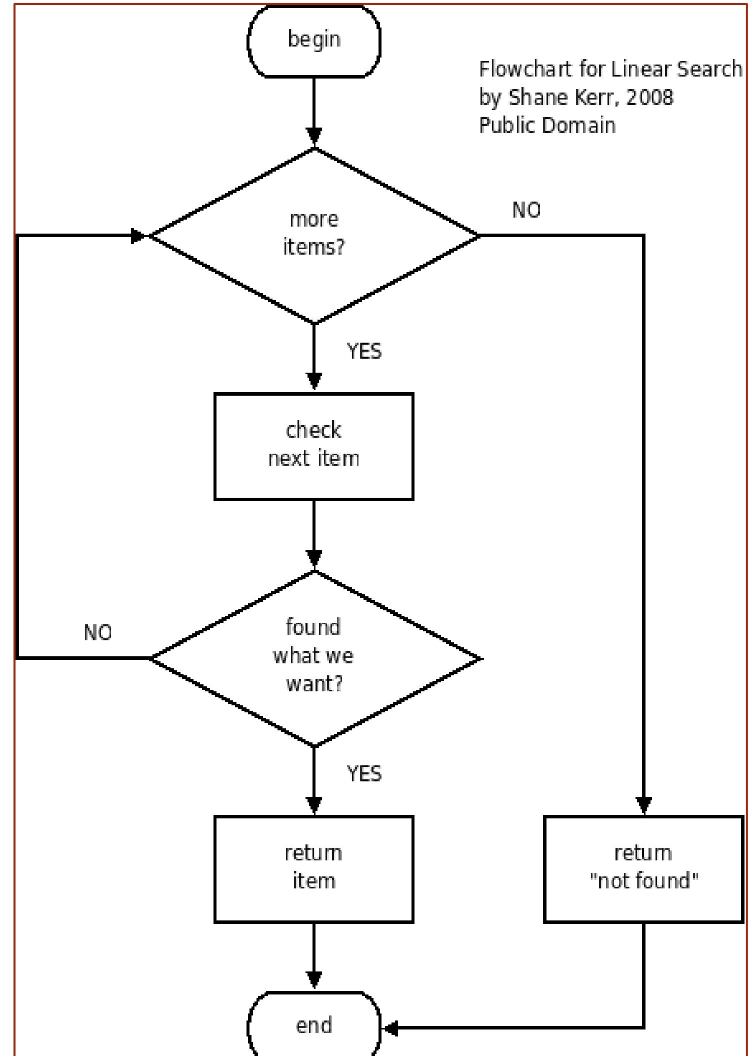
```
BEGIN LINEARSEARCH
      lastindex = ARRAYLENGTH(element)
      foundit = false
      index = 1
      READ targetvalue
      WHILE foundit = false AND index <= lastindex
            IF element(index) = targetvalue THEN
                  foundit = true
            ENDIF
            index = index + 1
      ENDWHILE
      IF foundit = true THEN
            DISPLAY "Target value found.
      ELSE
            DISPLAY "Target value not found"
      ENDIF
END LINEARSEARCH
```

How can I write algorithm (cont.)?

- Flowchart



Flowchart symbols



What is an efficient algorithm?

- Performance/time
- Storage/memory
- Availability
- Reliability
- Security
- Scalability
- …

Running time T(n)

- The running time of a given algorithm is the number of elementary operations to reach a solution.

    - The running time depends on the input size - n
        - Sorting – number of array elements
        - Arithmetic operation – number of bits
        - Graph search – number of vertices and edges

Some mathematical facts

- Some mathematical equalities are:

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

$$\sum_{i=1}^{n} i^2 = 1 + 4 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{n-1} 2^i = 1 + 2 + \ldots + 2^{n-1} = 2^n - 1$$

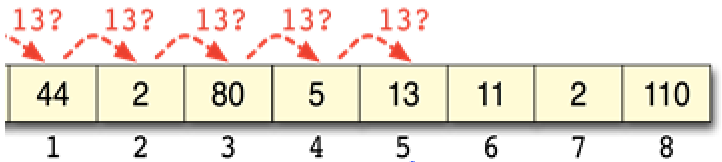
How can I analyse a given algorithm?

- Operation Counting
- Asymptotic Notations
- Substitution Method
- Recurrence Tree
- Master Method

Operation counting

```
FUNCTION LinearSearch(data: array of Integer; n:
Integer; value: Integer) : Integer
BEGIN
   FOR i := 1 to n DO
   BEGIN
        IF data[i] = value THEN
              return i
        END IF
   END FOR
   return 0
END
```

initialization 1 check 1	1 n+1
check 1 return 1	n 1
increment 1 return 1	n 1
T(n) = 1 + (n + 1) + n + 1 + n = 3n + 3	
Best case T(n) = 1 + 1 + 1 + 1 = 4	
Avg case T(n) = 1 + (n + 1)/2 + n/2 + n/2 + 1 = 3n/2 + 5/2	



Best, worst and average case complexity

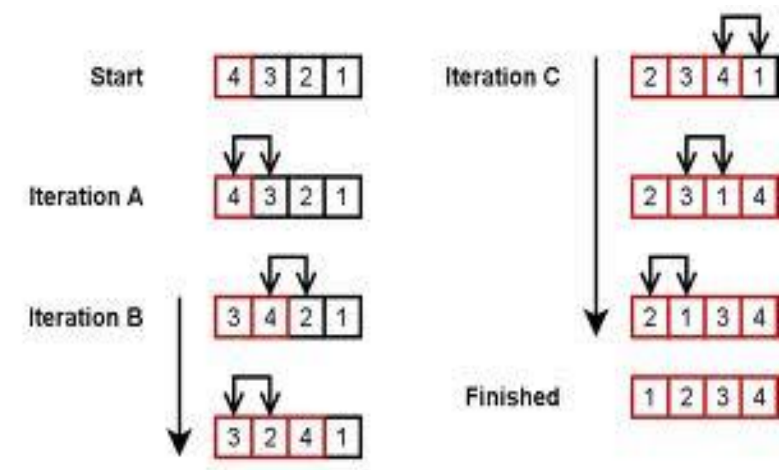
- Running the same algorithm on different inputs, yields different running times, why?
- Back to the linear search algorithm
  - What if the first element matches the value to search for? – Best case running time
  - What if the value does not belong to the data array? – Worst case running time
  - What if the value exists some where in the middle of the data array? – Average case running time

Why should we focus on the worst case?

- When comparing two algorithms, consider the worst-case running time for both…the lower is better.
  - Upper bound
  - Fairly often
  - Average case is usually similar to the worst case

Another example – nested loops

```
FOR j := 2 to n DO
       Temp := A[j]
       i := j -1
       WHILE i>0 and A[i]>Temp DO
              A[i + 1] := A[i]
              i := i-1
       END WHILE
       A[i+1] := Temp
END FOR
```



Another example – nested loops

```
FOR j := 2 to n DO
      Temp := A[j]
      i := j -1
      WHILE i > 0 and A[i] > Temp DO
            A[i + 1] := A[i]
            i := i-1
      END WHILE
      A[i+1] := Temp
END FOR
```

initialization 1	1
check 1	n
assignment 1	n - 1
assignment 1	n - 1
check 2	$\sum_{j=2}^{n} j = \left(\sum_{j=1}^{n} j\right) - 1 = \frac{n(n+1)}{2} - 1$
assignment 1	$\sum_{j=2}^{n} (j-1) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$
decrement 1	$\sum_{j=2}^{n} (j-1) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$
assignment 1	n - 1
increment 1	n - 1

T(n) = 1 + n + n - 1 + n - 1
+ 2 * (n(n+1)/2 - 1) + 2 *
n(n-1)/2 +(n-1)+(n-1)
T(n) = $2n^2$ + 5n - 5

Avg case T(n) = 1 + n + n - 1 + n - 1
+ (n(n+1)/2 - 1)
+ n(n-1)/2 + (n-1) + (n-1)
Avg case T(n) = n^2 + 5n - 4

Best case T(n) = 1 + n + (n - 1) + (n - 1)
+ (n - 1) + (n - 1) + (n - 1)
Best case T(n) = 6n - 4

Which algorithm is more efficient?

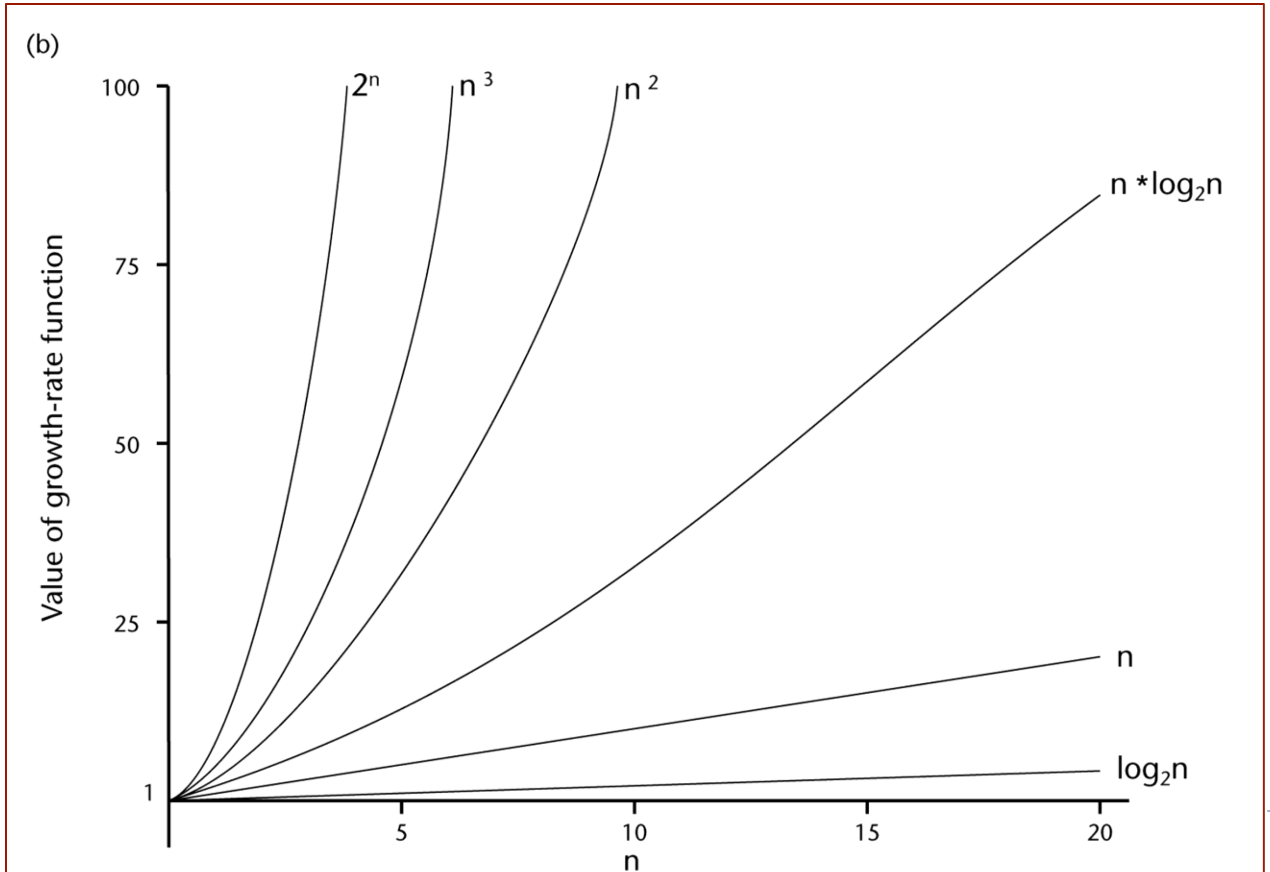
	A	B	C	D
input size	**n^2**	**$1000n^2$**	**$1000n^2 + 10n + 10$**	**5n3**
n = 1	1	$10^3 * 1$	$10^3 * 1 + 20$	5
n = 10	10^2	$10^3 * 10^2$	$10^3 * 10^2 + 110$	$5 * 10^3$
n = 100	10^4	$10^3 * 10^4$	$10^3 * 10^4 + 1010$	$5 * 10^6$
n = 200	$4 * 10^4$	$4 * 10^3 * 10^4$	$10^3 * 10^4 + 2010$	$4 * 10^7$
n = 1000	10^6	$10^3 * 10^6$	$10^3 * 10^6 + 10010$	$5 * 10^9$
n = 10,000	10^8	$10^3 * 10^8$	$10^3 * 10^8 + 100010$	$5 * 10^{12}$
n = 1000,000	10^{12}	$10^3 * 10^{12}$	$10^3 * 10^{12} + 1000010$	$5 * 10^{18}$

* Constant coefficients do not impact efficiency
* lower order terms do not impact efficiency

Order of growth

- Lower-order terms are not significant for large n
- Constants and coefficients are less significant

$$c^n >> n^3 >> n^2 >> n\log(n) >> n >> \log(n) >> c$$



Asymptotic notations

- How does the running time of an algo A increase with very large inputs ? – asymptotic efficiency

- **Big-O:** Asymptotic upper bound of the algorithm running time

  $O(g(n)) = \{ f(n)$: there exist positive constants c and $n0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n0$ $\}$

- **Big-Omega:** Asymptotic lower bound of the algorithm running time

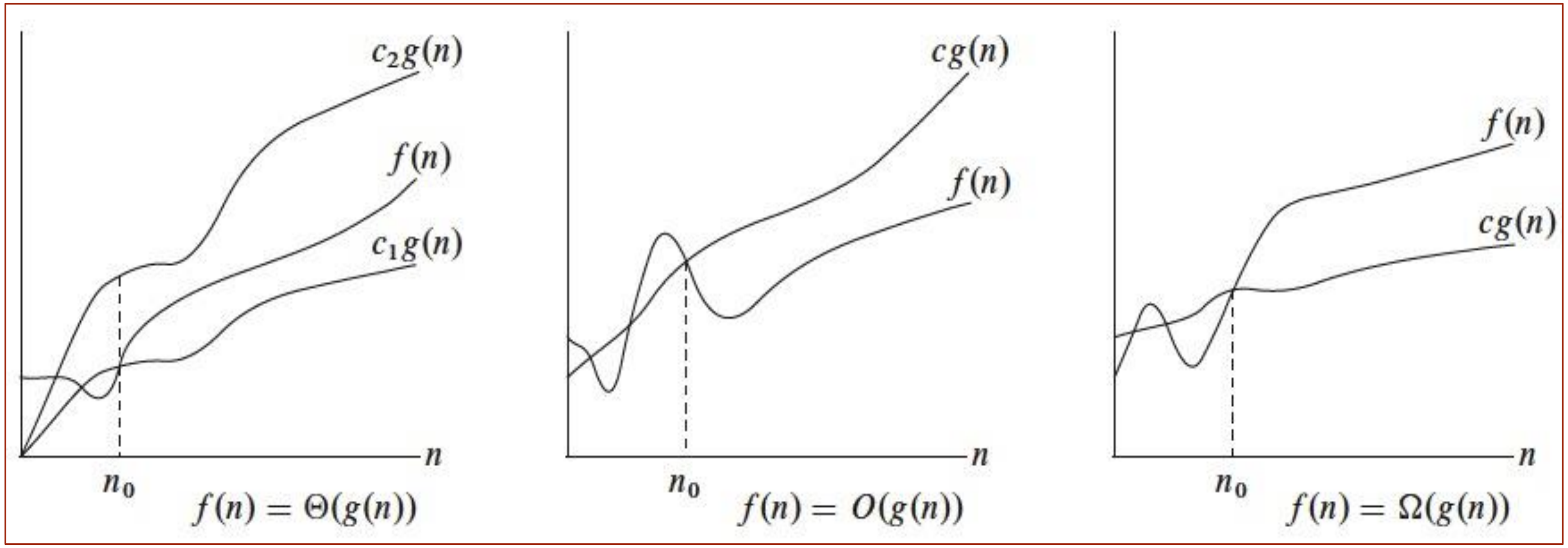
  $\Omega(g(n)) = \{ f(n)$: there exist positive constants c and $n0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n0$ $\}$

- **Big-Theta:** Asymptotically tight bound

  $\theta(g(n)) = \{ f(n)$: there exist positive $c1$, $c2$, and $n0$ such that

  $0 \le c1g(n) \le f(n) \le c2g(n)$ for all $n \ge n0$ $\}$

Asymptotic notations



Example 1

`int sum = 0;`
`for (int n = N; n > 0; n /= 2)`
`   for (int i = 0; i < n; i++)`
`      sum++;`
	N + N/2 + N/4 + … Linear
	O(N)

`int sum = 0;`
`for (int i = 1; i < N; i *= 2)`
`   for (int j = 0; j < N; j++)`
`      sum++;`
	N + N + N + … Linear-ithmic
	$O(N\log_2(N))$

`int sum = 0;`
`for (int i = 1; i < N; i *= 2)`
`   for(int j = 0; j < i; j++)`
`      sum++;`
	1 + 2 + N/4 + N/2 + N … Linear
	O(N)

Example 2

- Given, $f(n) = n^2/2 - 3n$; show that $f(n) = \Theta(n^2)$

$$c1n^2 \le (n^2/2 - 3n) \le c2n^2 \quad \div n^2$$

$$c1 \le (½ - 3/n) \le c2 \quad | \quad n, c1, c2 > 0$$

RHS to hold => $1/2 - 3/n = c2$

=> $c2 = ½$ for $n > 0$

LHS to hold => $1/2 - 3/n = c1$

=> $1/2 > 3/n$ => $n > 6$

=> $c1 = 1/2 - 3/7 =$ **1/14**

Example 3

- Show that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$

Given that $T(n) = O(g(n)) \Rightarrow c2 * g(n) > T(n)$

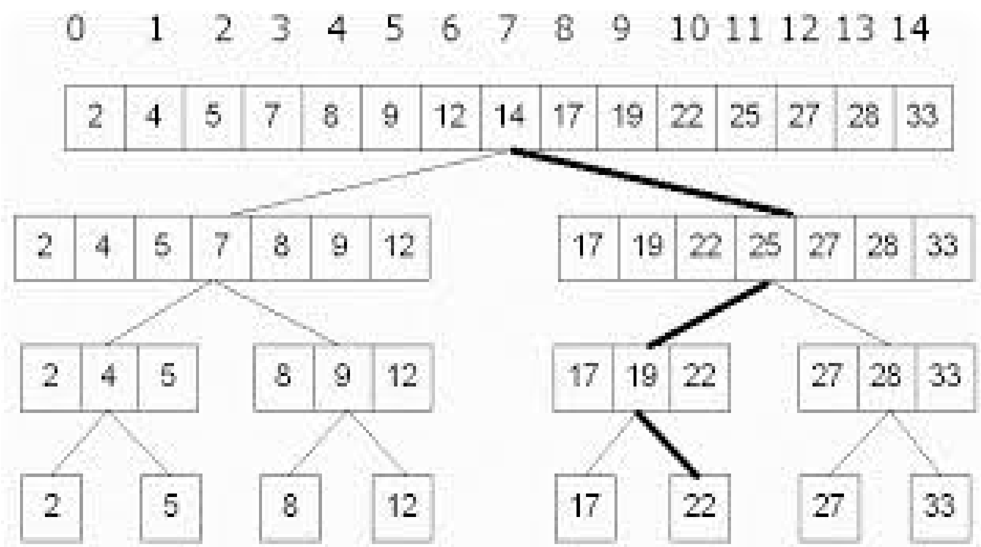
Given that $T(n) = \Omega(g(n)) \Rightarrow c1 * g(n) < T(n)$

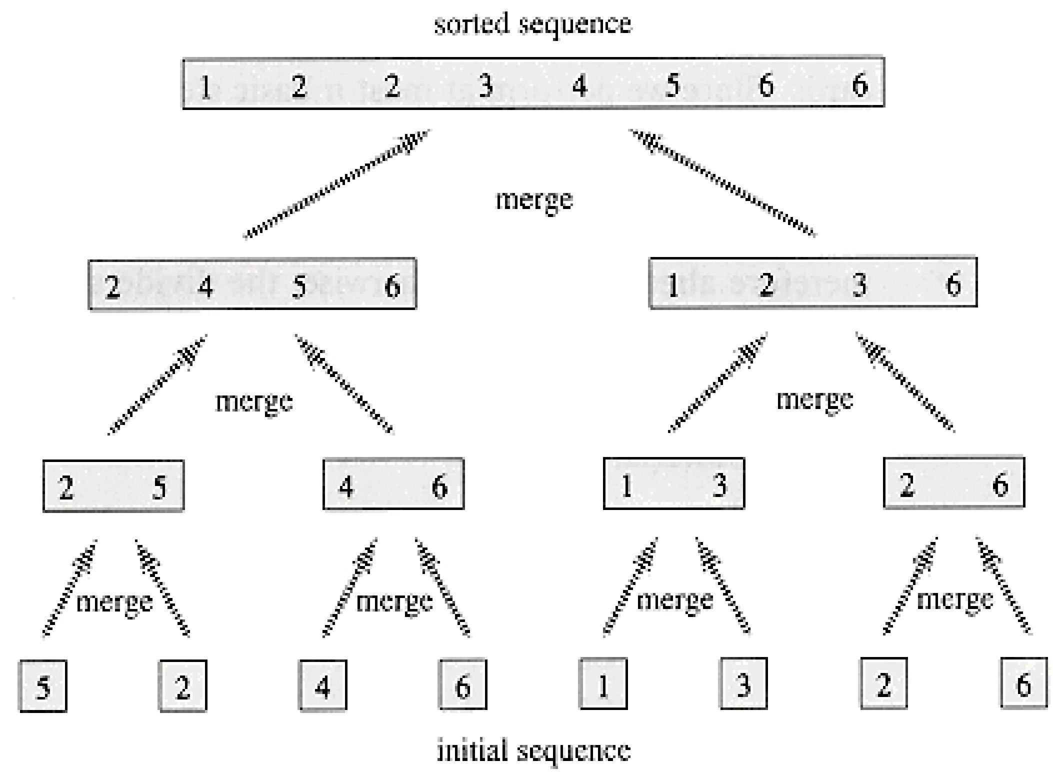
Thus, $c1 * g(n) < T(n) < c2 * g(n)$

Thus, $T(n) = \Theta(g(n))$

Divide and Conquer

- Divide the initial problem (sorting, searching, counting, any computation) into smaller INDEPENDENT sub-problems, solve these sub-problems recursively, and then combine their solution to solve the initial problem.





Recurrence Formula – Substitution - Example(1)

```
Long powerA(long x, long n)
{
        if (n==0) return 1;
        if (n==1) return x;
        else return x * powerA(x, n–1);
}
```

```
T(0)            =       1
T(1)            =       1
T(n)            =       T(n - 1) + 1
                =       T(n - 2) + 1 + 1
                =       T(n - 2) + 2

T(n)            =       T(n – k) + k
@ n – k = 1, k = n  - 1
T(n)            =       T(1) + (n – 1)
T(n)            =       1 + n - 1
T(n) in O(n)
```

Recurrence Formula – Substitution - Example(2)

```
long power(long x, long n)
{
        if (n==0) return 1;
        if (n==1) return x;
        if ((n % 2) == 0)
                return power(x*x, n/2);
        else
                return power(x*x, n/2) * x;
}
```

$$
\begin{aligned}
T(0) &= C1\\
T(1) &= C2\\
T(n) &= T(n/2) + C3\\
&= T(n/4) + C3 + C3\\
&= T(n/4) + 2C3\\
&= T(n/8) + 3C3\\
\\
T(n) &= T(n/2^i) + i * C3
\end{aligned}
$$

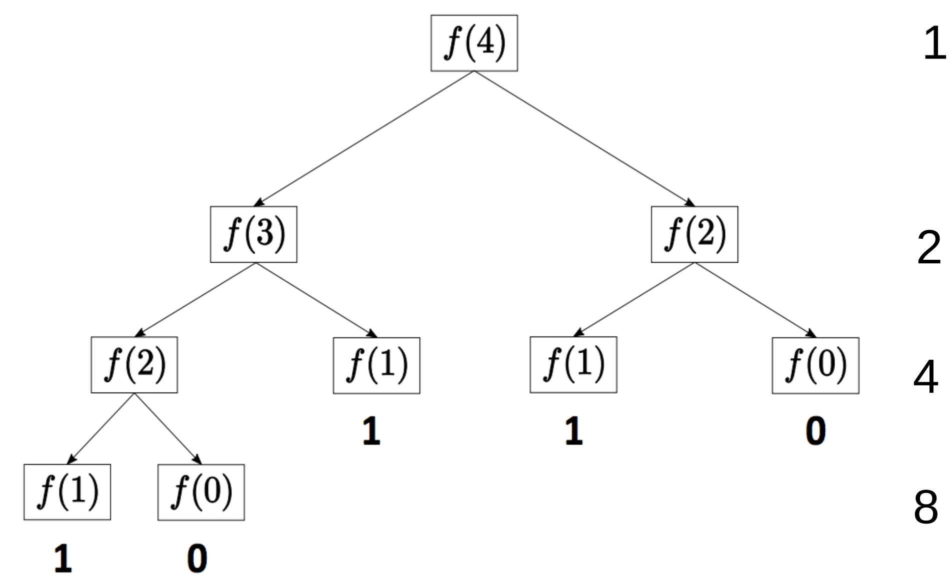
$@n/2^i = 1,\ n = 2^i,\ i = \log_2(n)$

$$
\begin{aligned}
T(n) &= T(1) + \log_2(n) * C3\\
T(n) &= C2 + \log_2(n) * C3
\end{aligned}
$$

$T(n)$ in $O(\mathbf{\log_2(n)})$

Recurrence Formula – Substitution - Fibonacci sequence

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- The next number is found by adding up the two numbers before it.
- $T(n) = T(n-1) + T(n-2) + C$
- $T(n) = [T(n-2) + T(n-3) + C] + T(n-2) + C$
- $T(n) = 2T(n-2) + T(n-3) + 2C$
- $T(n) = 3T(n-3) + 2T(n-4) + 4C$
- $T(n)$ is $O(2^n)$

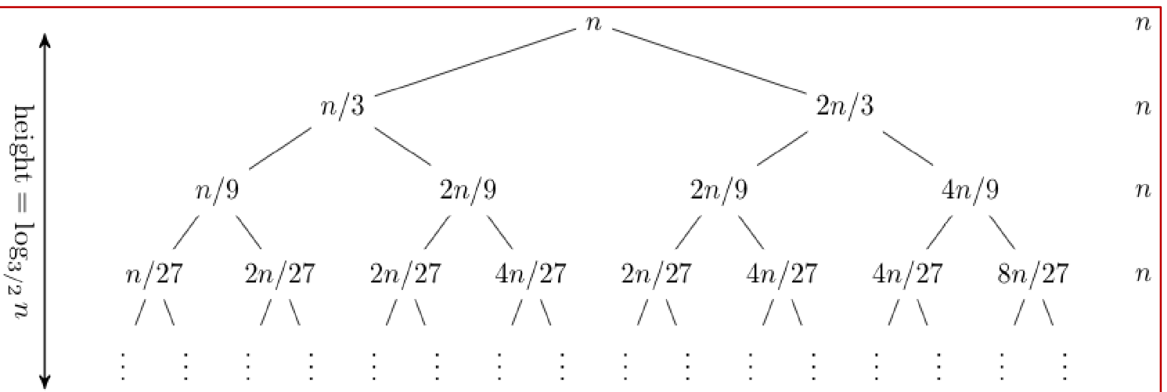


$n =$	0	1	2	3	4	5	6	7
$x_n =$	0	1	1	2	3	5	8	13

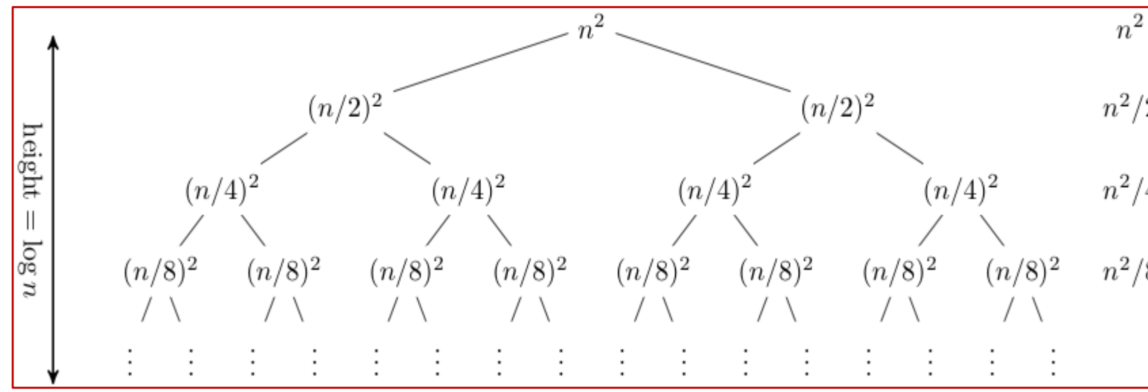
Recurrence Formula - Recurrence Tree Method

Visually analyse algorithm complexity…

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



$T(n) = 2T(n/2) + n^2$



T(n) = n + n + n + … (how many times?)

O(nlogn)

$T(n) = n^2 + n^2/2 + n^2/4 + n^2/8 + \ldots$

$O(n^2)$

Recurrence Formula – Master Method

a => how many sub-problems we have each time
b => how do we divide the problem input

Case1: If the work done at leaves is more, then leaves are the dominant part, and our result becomes the work done at leaves.

Case2: If work done at leaves and root is asymptotically same, then result becomes height multiplied by work done at any level

Case3: If work done at root is asymptotically more, then our result becomes work done at root.

Can you apply this on the examples in the previous slide?
What cases did we have there?

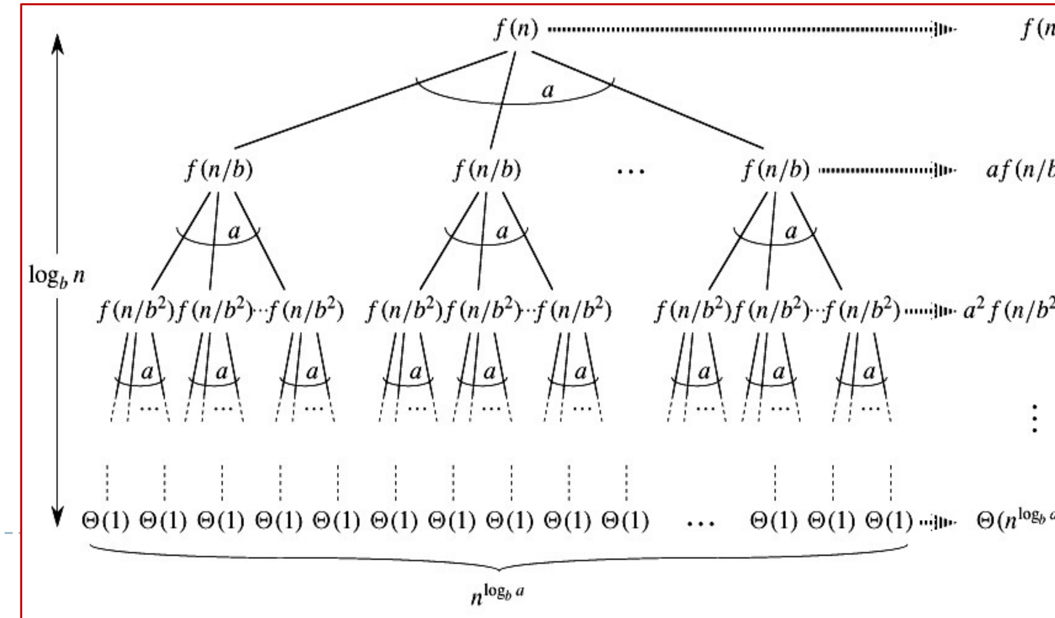
```
T(n) = aT(n/b) + f(n) where a >= 1 and b > 1
```

There are following three cases:

1. If $f(n) = \Theta(n^c)$ where $c < \mathrm{Log}_b a$ then $T(n) = \Theta(n^{\mathrm{Log}_b a})$

2. If $f(n) = \Theta(n^c)$ where $c = \mathrm{Log}_b a$ then $T(n) = \Theta(n^c \mathrm{Log}\ n)$

3. If $f(n) = \Theta(n^c)$ where $c > \mathrm{Log}_b a$ then $T(n) = \Theta(f(n))$



More examples

- We will revisit this next week
  - Merge sort
  - Binary search

Practical – Week1

- Revision on OOP
- Generics
- Disk I/O
- Basic complexity analysis examples – from your code