CPS 616

Part 1

Dr. Yeganeh Bahoo

Welcome to CPS 616 Algorithms

logistics and introduction

- **Course Q&A:**

  slido.com #987102
- **Instructor:** Dr. Yeganeh Bahoo
- **Lectures:** Mon. 8:10 to 9:00, Wen. 16:10 to 18:00
- **Laboratory:** Labs start in the week of Jan 31. See your schedule.
- **Office Hour:**

  Tuesdays 14:00-15:00
- **Technology learning tools:**

  D2L (http://my.ryerson.ca)

Agenda

- Meet Yeganeh Bahoo: background, experience, and interests (related and unrelated to what we will learn in this course)
- A tour of our Course Outline/Course Management Form
- Expectations
- What this course is about

Prerequisites

CPS 305 and MTH 210

OR

CPS 305 and CPS 420

Office hours

Tuesday 14:00-15:00

Email: bahoo@ryerson.ca with title CPS 616

For office hour book through Google Calendar via D2L

Class rules and announcement

Class announcement: D2L

During class ask your questions in Slido.

More platform for questions: join the discussion forum in D2L and ask questions or participate in discussions.

Note 1: the class will be recorded **if possible** (there will be no promise about the time of upload, or the final quality etc)

Note 2: the slides will be available in D2L (if possible before the class).

Marks and exams

Six labs 30%

Midterm exam 30%

Final exam 40%

Important dates

Midterm: March 5

Last class: April 13

Final exam: TBA

Accommodation support

If you need accommodation support make sure you are in touch with *Academic Accommodation Support Office* on time.

You must register and be in touch with the me if needed before the first graded work is due.

Academic Integrity

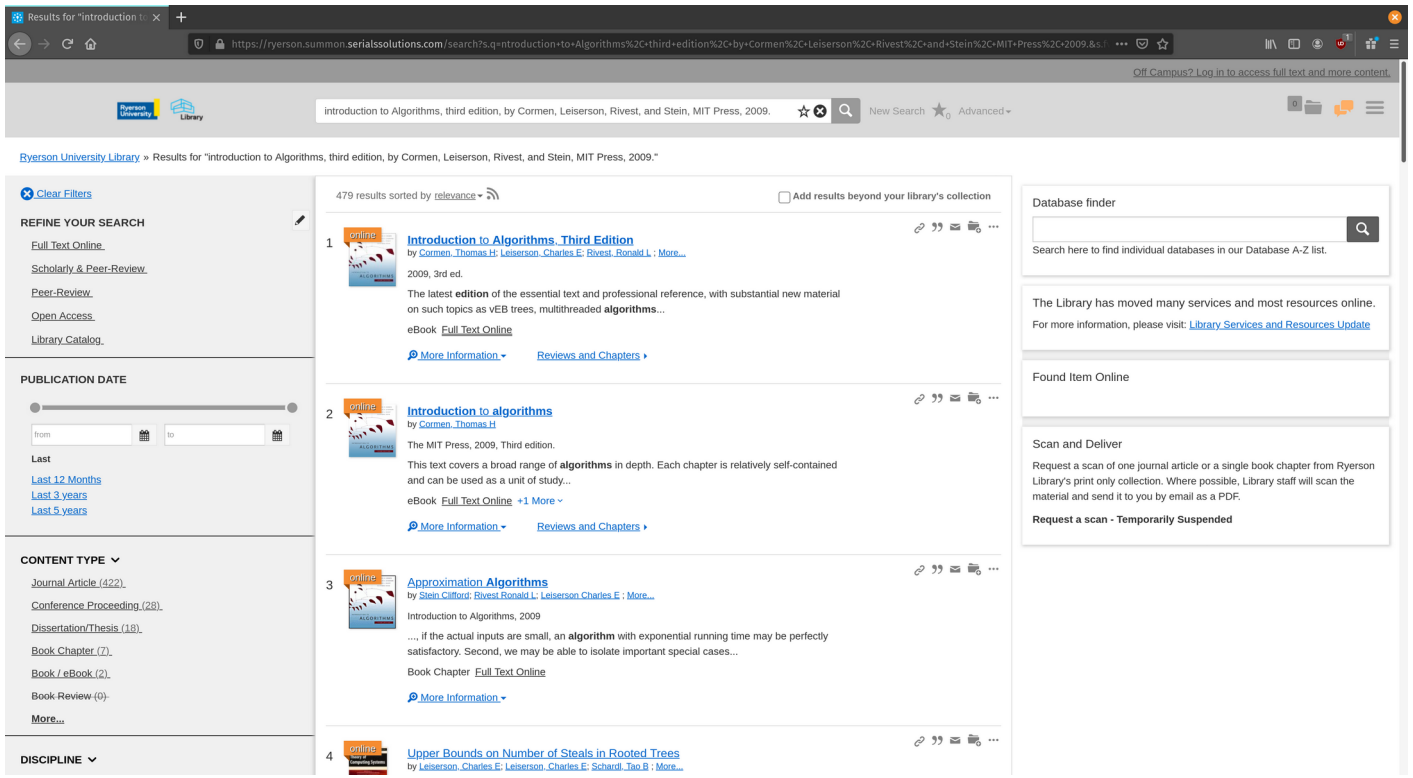
https://www.ryerson.ca/academicintegrity/

Assignments

It is OK to discuss the assignment with other students. However, you must write your solution in our own words (**copy/paste of someone else's is not acceptable**). Also, you must state the name of the people you work with in your assignment submission

Reference

Introduction to Algorithms, third edition, by Cormen, Leiserson, Rivest, and Stein, MIT Press, 2009. [CLRS]



Ryerson Libraray

Other Policies

See course outline provided for the course in D2L.

Expectations

- What I expect of you.
  - Truly prepare for the next lecture
  - Materials you will need in class:
    - paper, pencil, and eraser
    - personal computer (PC or laptop)
  - Deliberate practice, i.e., try difficult exercises
- What do you expect of me?
  - Go to Slido!

Let’s start

Reading

Cormen et al.

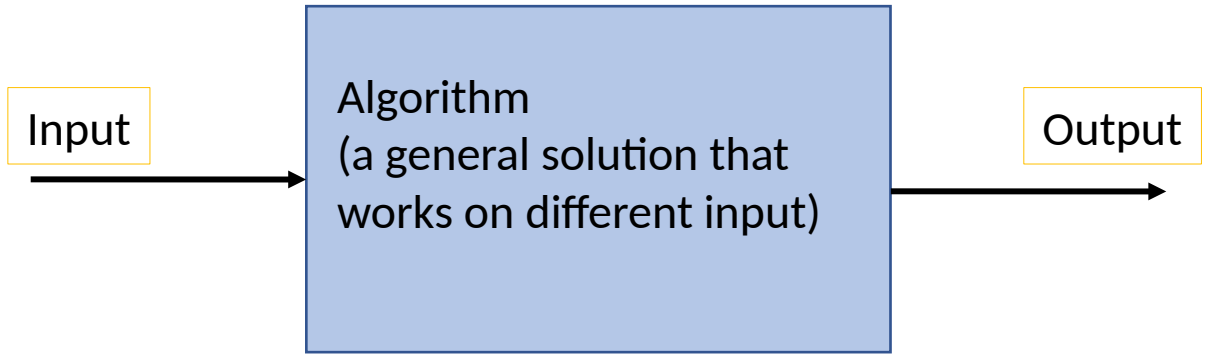
- ✓ Algorithms 1.1
- ✓ Algorithms as a technology 1.2

Algorithm

Input

Output

Finite set of steps



Example

Sorting!

Set 1: 1, 23, 56, 1000, 10

Set 2: -1, 456, 12, 23

What to analyze when an algorithm is proposed for a problem

Correctness

Cost (space, time)

- ✓ How fast it runs
- ✓ How much memory it needs

Optimality

Algorithm analysis

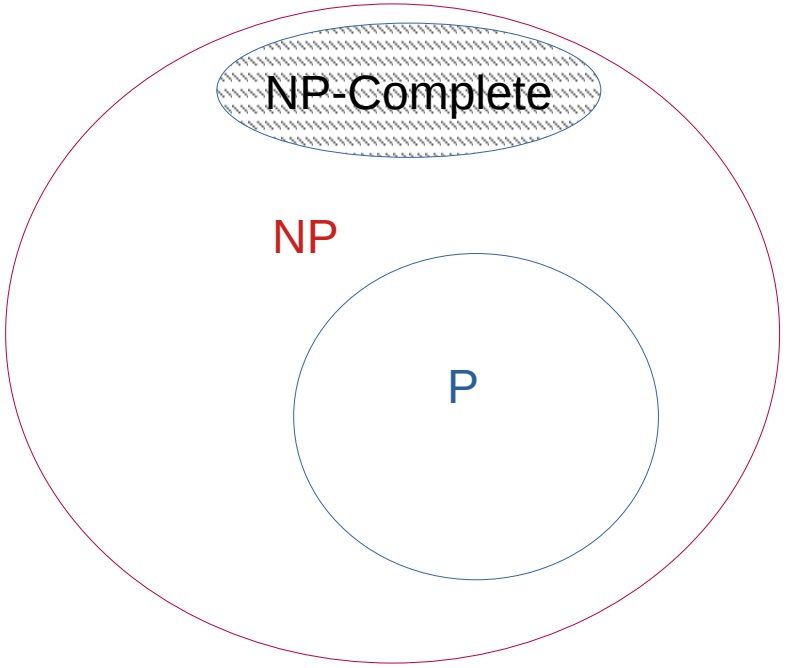
- Cost in terms of input size
- Asymptotic notation (max, min, average cases)
- Cost of recursive algorithms

Algorithmic techniques

- Divide-and-conquer algorithms
- Greedy algorithms
- Dynamic programming
- Randomized algorithms

Complexity theory

- P, NP, and NP-Complete: categories of problems based on the complexity of their algorithms



What to analyze when an algorithm is proposed for a problem

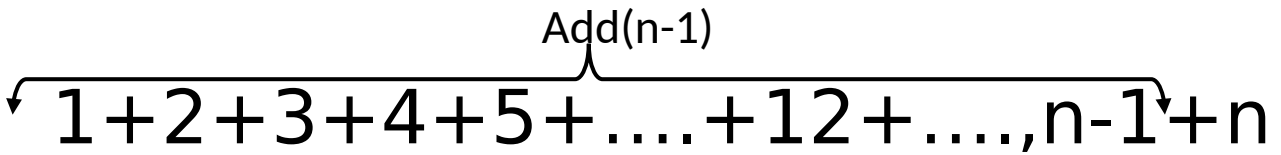
Correctness

Cost (space, time)

- ✓ How fast it runs
- ✓ How much memory it needs

Optimality

Example: addition of n numbers

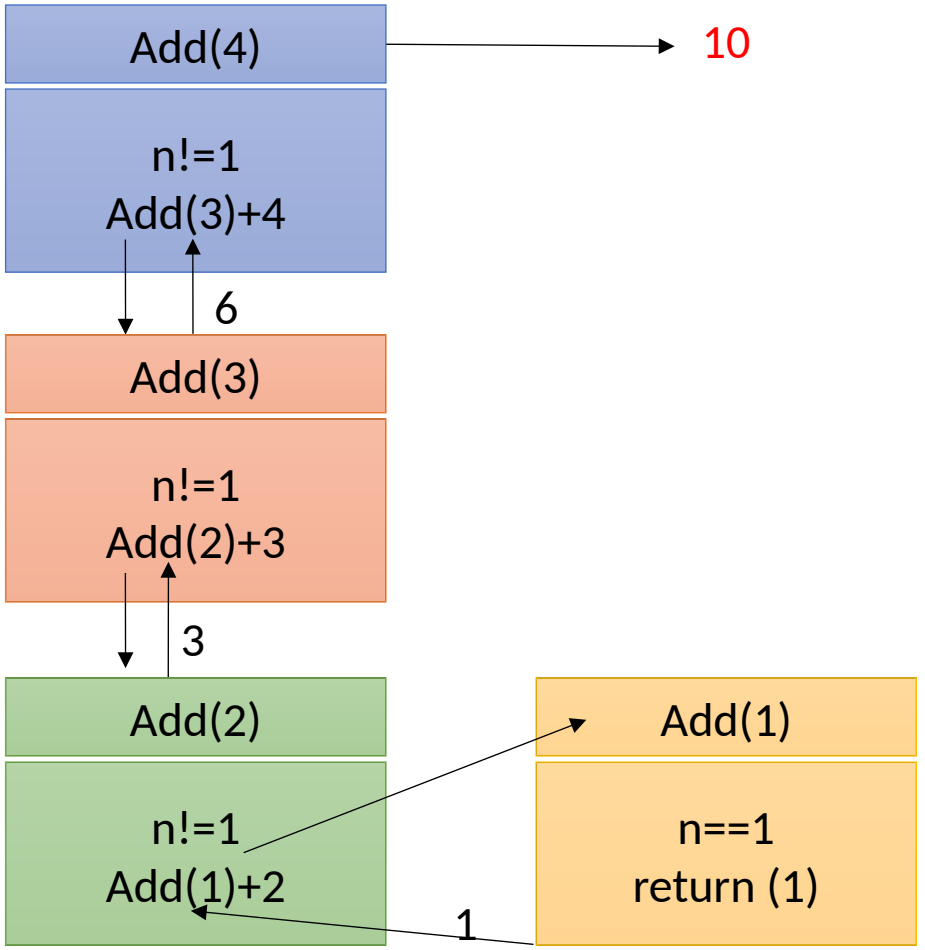


Example: addition of n numbers

```
Add(n)
{
    if n==1:
            return 1;
    Otherwise:
            return Add (n-1) + n;
}
```

Example: addition of n numbers

```
Add(n)
{
    if n==1:
            return 1;
    Otherwise:
            return Add (n-1) + n;
}
```



Example: addition of n numbers

Correctness:

$$1+2+3+4+5+....+12+....+n$$

Cost:

$$T(n) = \begin{cases} O(1) & , n=1 \\ T(n-1) + O(1) & , n!=1 \end{cases}$$

Cost for Add(n)

Cost for Add(n-1)

Local cost

Optimality: can it be done faster?

Example: addition of n numbers

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

Primitive operations

Just having addition, multiplication, and division.

It is faster than previous algorithm, because ....

Primitive operations

- The operations that take O(1) to run.
- The model of computation used to run the algorithm on dictates which operations are primitive.

RAM Word Model: the model used in this course

Data: set of words

Each word is an integer in $\{ -2^{u-1}, \ldots, 2^{u-1} - 1 \}$

Read and write takes O(1) time [one CPU step]

u bits for each word, where u=32 or 64

Float, integer, double and long can be stored in O(1) words in this model!

Primitive operation in RAM model

- Basic integer arithmetic on words: addition, subtraction, multiplication, division
- Logistic operations: AND, OR, XOR, NOT, and bit shift on words
- Boolean operation: <,>,==,!=,>=, =<
- Read or write a word

Non-primitive operation in RAM model

- exponentiation (or radicals)
- logarithms

Example: addition of n number

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

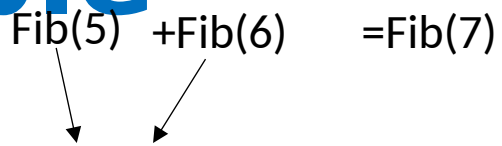
Just having addition, multiplication, and division.

It is faster than previous algorithm, because this is a set of three primitive operations, each take O(1) to run.

Other models of computation

- Turing machine
- real RAM
- cell probe
- ….

Another example



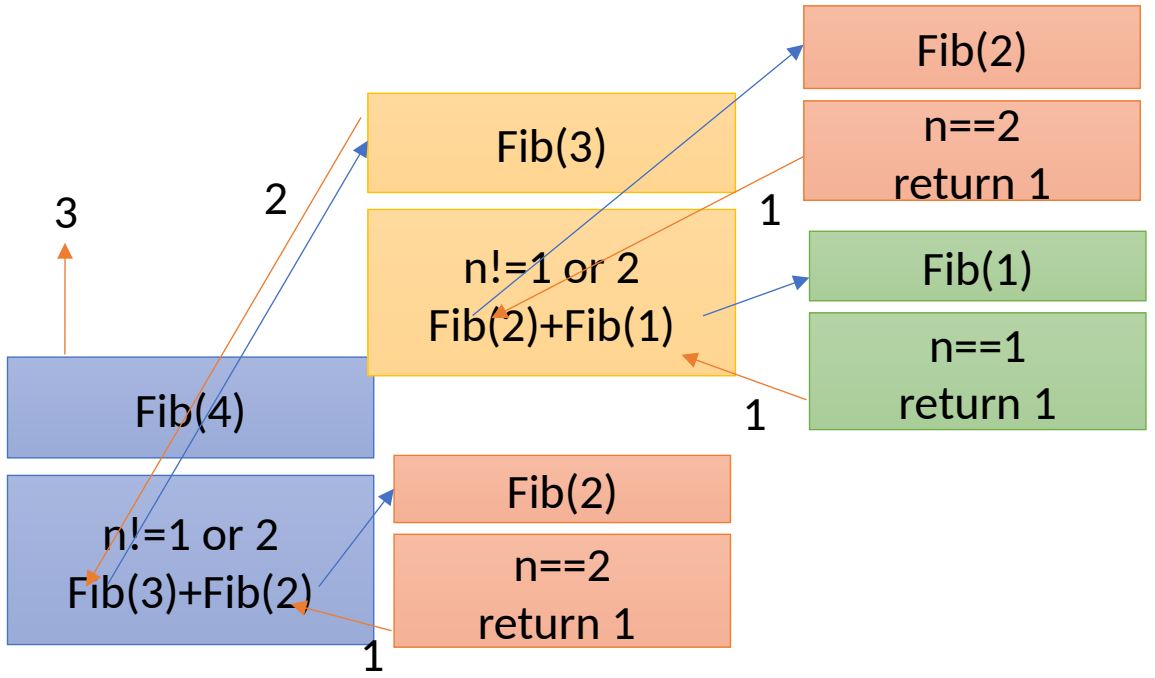
1,1,2,3,5,8,13,……

Fibonacci sequence

$$Fib(n) = \begin{cases} 1 & , n=1 \text{ or } 2 \\ Fib(n-1)+Fib(n-2) & , \text{otherwise} \end{cases}$$

Another example

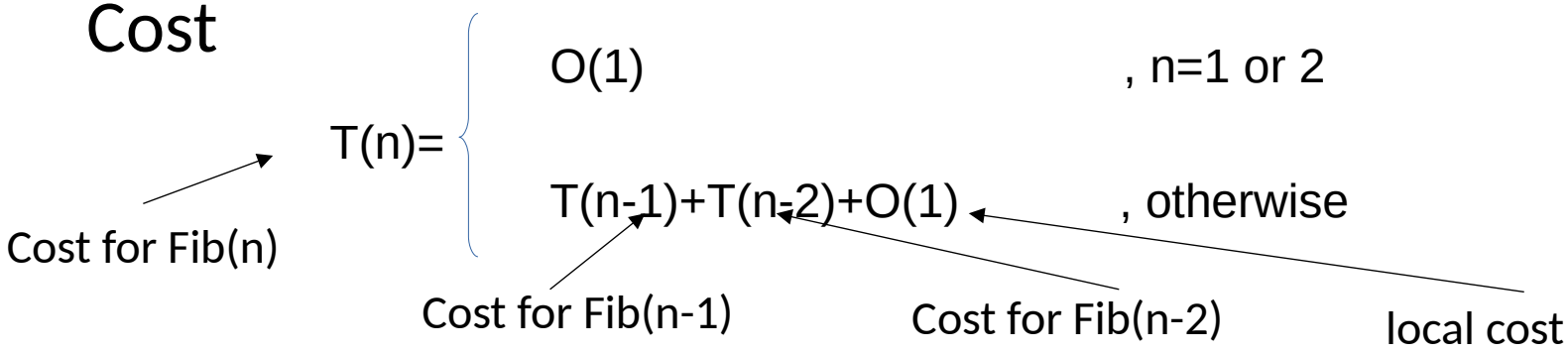
```
Fib(n)
{ if (n==1 || n==2):
        return 1;
otherwise:
        return Fib(n-1)+Fib(n-2)
}
```



Another example

Correctness

Cost



$$T(n) \in O(1.62^n)$$

Optimal ?

Another example

```
Fib(n)
{ int F[n];

   F[0]=1;
   F[1]=1;

for (int i=2; i<n+1;i++)
        F[i]=F(i-1)+F(i-2);

Return F(n-1);
}
```

$$T(n) \in O(n)$$

Another example

$$Fib(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Optimal?

What operations are involved in this equation?

Needed mathematical background

As we saw, we express the time complexity of an algorithm with mathematical equation.

This is because mathematics is a **quantitative** tool to compare algorithms with each other.
This is to express which algorithm is faster or more efficient.

Needed mathematical background

- L’Hôpital’s rule
- Logarithms
- Arithmetic and other polynomial series
- Mathematical induction
- Geometric series
- Recurrence relations
- Combinations

CPS 616

Slide 2

Dr. Yeganeh Bahoo

Outline: Asymptotic Notation

- understanding of big-Oh
- formal definition of big-Oh
- Omega notation
- Theta notation
- little oh
- little omega

Read chapter 2.2 and 3.1 , of the book

Definition of the algorithm analysis

- Time
- Space
- Network traffic
- Time of the developer to program
- Complexity of the code…..

For us and in this course: running time

Comparing two programs for one problem

Alice:
Finding the smallest element in an array
Her algorithm takes 910 millisecond to run when an array of 100 is given
Her system: Linux, 8Gb RAM
Programming language: Python

Bob:
Finding the smallest element in an array
His algorithm takes 1050 millisecond to run when an array of 90 is given
His system: Windows 10, 16Gb RAM
Programming language: C#

Comparing two programs for one problem

Alice:
For the input of size n, the minimum number in the array can be found with f(n) primitive operation:

$$f(n) = 12\, n\, log\, n + 13n + 500$$

Bob:
For the input of size m, the minimum number in the array can be found with g(m) primitive operation:

$$g(m) = 2000\, m$$

Comparing two programs for one problem

Alice:

$$f(n) \in \theta(n \log n)$$

Bob:

$$g(m) \in \theta(m)$$

Expression based on the size of the input

- Running the program on one item is not the same as running the program on 10000000 items.
- Finding the minimum element in the array of size one is different than fining the minimum element in an array of size 10000000.

$$f(n) = 12\ n\ log\ n + 13n + 500$$
$$g(n) = 2000\ n$$

To calculate the cost

1. Find the function that the cost of the algorithm is based on
2. Express it by asymptotic notations

Alice algorithm:

1. $f(n) = 12\ n\ log\ n + 13n + 500$
2. $f(n) \in \theta(n \log n)$

- $f(n) \in O(n\ log\ n)$

Asymptotic notations

- Expressing the time complexity better
- We can compare easier

- O (oh)
- θ (theta)
- Ω (omega)
- o (little oh)
- ω (little omega)

Analyzing the running time of a code

```
int Fun(int [] A)
{
int parameter=A[0], i=0;
while (i<A.length)
   {
     if (A[i]<parameter)
        parameter=A[i];
    i++;
    }
}
```

Primitive operation in RAM Word model

- Basic athematic: Addition, subtraction, multiplication, division
- Logistic operations: AND, OR, XOR, NOT, and bit shift on words
- Boolean operation: $<, >, ==, !=, \geq, \leq$
- Read or write a word

Non-primitive operation in RAM model

- Exponentiation
- Logarithms

Running time

- Code is a set of instructions
- Each instruction takes one clock cycle to run
- Each of these instruction is called a primitive operation

⇒ We must count the number of primitive operation

=

In code: count the number of steps

Step?

- Usually a line of code (usually a line without loop or method call)= a line that perform in O(1) time.

Example:

```
int i;
int j;
j=0;
i=23;
i=i*j;
```

5 steps

Counting step based on line of code

```
int i;
int j;
j=0;
i=23;
i=i*j;
```

5 steps

=

```
int i=23;
int j=0;
i=i*j;
```

3 steps

We want a function to express running time proportional to the number of primitive operations

Suggestions?

Primitive operations instead!

But, counting the step and using asymptotic notation is much easier!

What are we looking for in running time?

- Best-case time
- Worst-case time
- Average case time

Worst case running time means …

The slowest behavior of the algorithm

Gives us upper bound on the time complexity

Example

- Input: an array A of integers
- Output: finding the minimum

```
int min=A[0], i=0;
while(i<A.length)
{
  if (A[i]<min)
     min=A[i];
  i++;
}
```

Part a: 1 step

Part b: 3 steps inside the loop

Loop runs for i=0, …n-1

Example

- Input: an array A of integers
- Output: finding the minimum

```
int min=0, i=0;
while (i<A.length)
{
   if (A[i]<min)
       min=A[i];
   i++;
}
```

$$f(n) = 1 + \sum_{x=0}^{n-1} 3$$

$$f(n) = 1 + 3n$$

Another example

```
void F1( int [] A ) {
int n = A.length;
    for ( int i = 0 ;i< n-1 ; i++ ) {
        for ( int j = i+1 ; j < n ; j++ ) {
            if ( A[i] > A[j] ) {
                int temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            }
    }
}
```

What does this function do?

What is the worst case running time?

Selection sort

```
void F1( int [] A ) {
int n = A.length;
    for ( int i = 0 ;i< n-1 ; i++ ) {
        for ( int j = i+1 ; j < n ; j++ ) {
            if ( A[i] > A[j] ) {
                int temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            }
    }
}
```

$$f(n) = 1 + \sum_{x=0}^{n-2} \sum_{y=x+1}^{n-1} 4$$

$$= 1 + \sum_{x=0}^{n-2} 4(n - x\text{-}1)$$

$$= 1 + \sum_{x=0}^{n-2} 4n - \sum_{x=0}^{n-2} 4x - \sum_{x=0}^{n-2} 4$$

$$= 1 + 4n(n-1) - 4\,(n\text{-}1)(n\text{-}2)/2 - 4(n\text{-}1)$$

$$= 2n^2 - 2n + 1$$

Effect of the method call in the running time

```
void main(int n){
int i=0;
int result=0;
for (int j=1;j<n;j++)
   { i++;
    result=Add(i, result);
}
}
```

```
int Add(int i, int result){
return i+result;
}
```

$$f(n) = 2 + \sum_{x=1}^{n-1} (1 + h(x))$$

$$\in O(n)$$

How to simplify functions

- Ignore the constants
- Ignore low-order functions

- Example:

$$f(n) = n^4 + 3n^3 + n\log n + 500000$$

$$f(n) \in O(n^4)$$

Some examples

$$f(n) = 4n^3 \log n + 12\sqrt{n} \quad \in O(n^3 \log n)$$

$$g(n) = n\sqrt{n} + n \log n \quad \in O(n\sqrt{n})$$

$$h(n) = 2n^4 + \sqrt[4]{n+4} \quad \in O(n^4)$$

Why big-O

- Simple way for expressing the running time!

$$f(n) = \begin{cases} (n\,log\,n)^4 + \sqrt{n} + 12 \;, n > 20 \\ n + \sqrt{\log n} \;, \quad 10 < n < 20 \\ n^7 \log n \;, \quad n < 10 \end{cases}$$

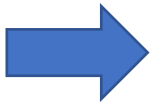
$$f(n) \in O(n^7 \log n)$$

- $f(n) = \begin{cases} 54\,n + 9 & , n = 2k \\ n\sqrt{n} + n\log n + 1 & , \quad n = 2k + 1 \end{cases}$

$\in O(n\sqrt{n})$

Alice:

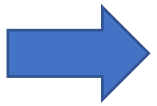
```
void F1( int [] A  ) {
int n = A.length;
    for ( int i = 0 ;i< n-1 ; i++ ) {
        for ( int j = i+1 ; j < n ; j++ ) {
            if ( A[i] > A[j] ) {
                int temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            }
     }
int min=A[0];
for (int i=0;i<n;i++)
    if (A[i]<min)
        min=A[i];
}
```



$f(n) \in O(n^2)$

Bob:

```
void F1( int [] A  ) {
int n = A.length;
int min=A[0];
for (int i=0;i<n;i++)
    if (A[i]<min)
        min=A[i];
}
```



$g(n) \in O(n)$

Running time of mathematical function in CPU when n=1000 if CPU has the power to run one million instruction per second

complexity	name	running time
1	constant	1 microsecond
log n	logarithmic	6.9 microseconds
$\sqrt{n}$		31 microseconds
n	linear	1 millisecond
n log n		6.9 milliseconds
n^2	quadratic	1 second
n^3	cubic	16 minutes
n^4	quartic	11 days
2^n	exponential	$3.4 * 10^{287}$ years
n!	factorial	$1.3 * 10^{2554}$ years

Flashback to cost function: Counting step based on line of code

```
Void main(int A[] ){
for (int n=0;n<A.length;n++){
    int i;
    int j;
    j=0;
    i=23;
    i=i*j;}
}
```

$\in O(n)$

```
void main (int [] A){
for (int m=0;m<A.length;m++){
int i=23;
int j=0;
i=i*j;}
}
```

$\in O(n)$

Flashback to cost function: (additional note) Effect of the loop in the running time

```
for (int n=0;n<A.length;n++)
        n=n*2;
```

$$f(n) = \sum_{x=0}^{n-1} 1 = n \qquad \in O(n)$$

```
for (int n=0;n<20;n++)
        n=n*2;
```

$$g(n) = \sum_{x=0}^{19} 1 = 20 \qquad \in O(1)$$

Function growth

$1 < \log n < n^{1/4} < \sqrt{n} < n < n \log n < n\,log^2 n < n\sqrt{n} < n^2 < n^2 \log n < n^3 < 2^n < n! < n^n$

$f(n) = n$

$n \in O(n)$

$n \in O(n \log n)$

$n \in O(n^4 \log n)$

$n \in O(n!)$

$f(n) = 5n \log n + n^2 + 23$

$\in O(n^2)$

$\in O(n^2 \log n)$

$O(n \log n)?$

Big-O

- Upper bound

$$n \in O(n)$$
$$n \in O(n^4 \log n)$$
$$n \in O(n!)$$

- Highest order term in the function identifies it
- There are asymptotic notation that give more details

$f(n) = O(g(n))?$

- $f(n) \leq g(n)$?

  - We only consider the highest order function
  - We are ignoring constants

Formal definition of big-O

$f(n)$ is $O(g(n))$ if there exist a constant $M > 0$ and $n_0 > 0$ such that

$$f(n) \leq M.g(n), \qquad for\ n > n_0$$

Ignore the constants

Consider the highest order term

Example:

- Example: $n^2 + 13 > 50$ for large values of n

n=7 => 62>50
n=8 => 77>50
....

Example:

Consider $f(n)=4n^3\log(n)$ And $g(n)=n^3\log(n)$

Is $f(n)\in O(g(n))?$

Formal definition of big-O

$f(n)$ *is* $O(g(n))$ if there exist a constant $M > 0$ *and* $n_0 > 0$ such that

$$f(n) \leq M.g(n), \qquad for\ n > n_0$$

Ignore the constants

Consider the highest order term

$$f(n) \in O\big(g(n)\big) \Leftrightarrow \exists M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, f(n) \leq Mg(n)$$

Quantifiers

Universal quantifier (∀)

- For all

$$\forall x > 2, x^3 + 2 > 10$$

Existential quantifier (∃)

- For some

$$\exists x > 0, s.t.\ x^2 = 64$$

$$\exists x, s.t.\ x^2 = 64$$

Quantifier in Calculus

$$\lim_{x \to \infty} f(x) \quad = c$$

$$\forall \epsilon, \exists \delta \; such \; that \; |x - a| < \delta \rightarrow |f(x) - c| < \epsilon$$

Formal definition of big-O

$f(n)$ *is* $O(g(n))$ if there exist a constant $M > 0$ *and* $n_0 > 0$ such that

$$f(n) \leq M.g(n), \qquad for\ n > n_0$$

Ignore the constants

Consider the highest order term

$$f(n) \in O\big(g(n)\big) \Leftrightarrow \exists M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, f(n) \leq Mg(n)$$

Set notation

$$f(n) \in O\big(g(n)\big) \; or \; f(n) = O\big(g(n)\big)$$

This does not mean that $f(n) = g(n)$

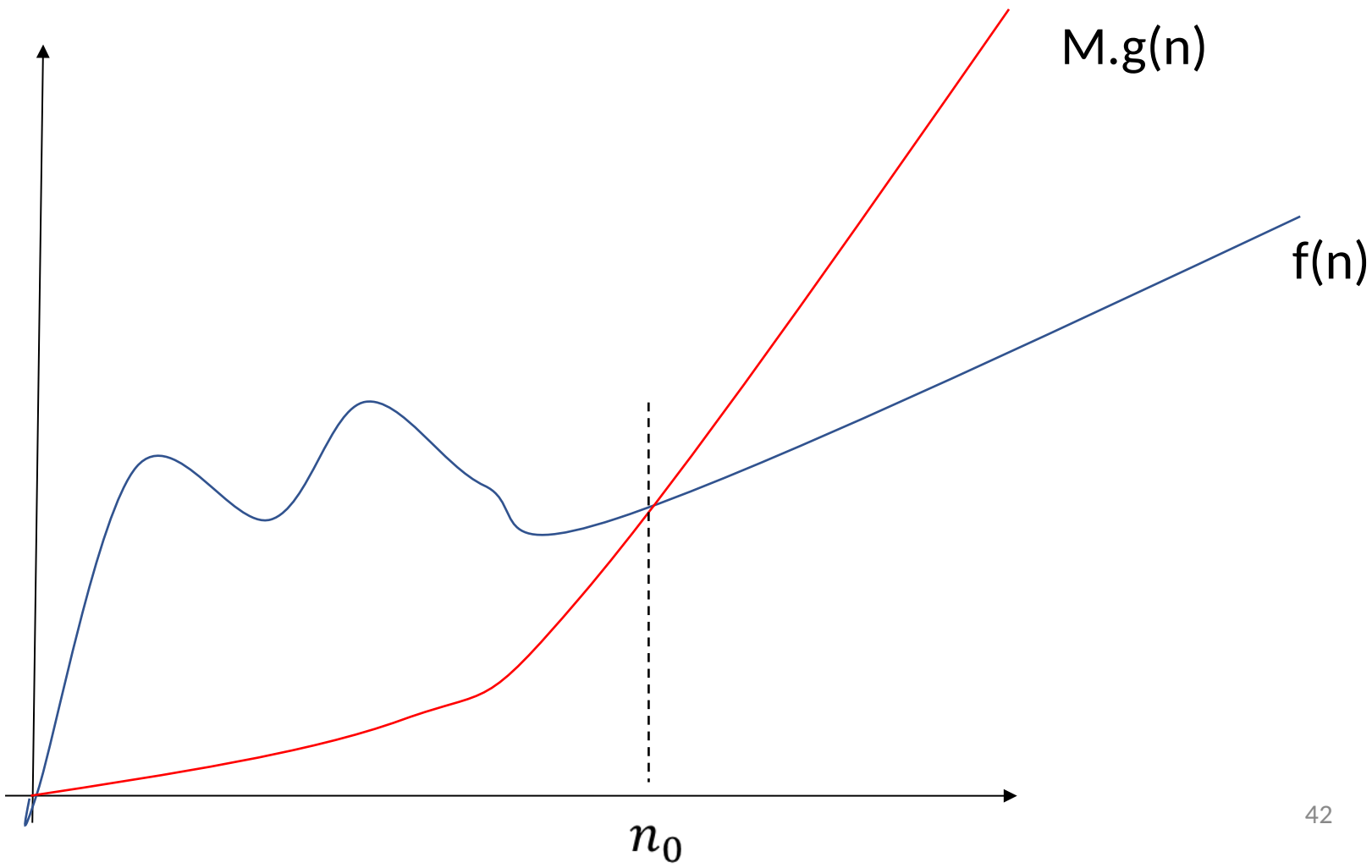
$$n = O\big(n^2\big)$$

$$n^2 = O\big(n^2\big)$$

$$n \neq n^2$$

Formal definition of the mathematical definition of big-O means …

$f(n) \in O(g(n))$



Is $f(n) \in O(g(n))$?

- You need to show the definition holds!

$$f(n) \in O(g(n)) \Leftrightarrow \exists M > 0, \exists n_0 > 0 \; s.t. \; \forall n > n_0, f(n) \leq Mg(n)$$

Example

$$(n^3 + 4) \in O(n^3)?$$

Let M =2, $n_0 = 3$

$$n \geq 3$$
$$n^3 \geq 27$$
$$n^3 \geq 4$$
$$2n^3 \geq 4 + n^3$$
$$M.g(n) \geq f(n)$$

Negating Quantifiers

- Not everyday is snowy is cloudy=There are some days that are sunny!

$$\neg[\forall x\, P(x)] = \exists x\, \neg P(x)$$

$$\neg[\exists x\, P(x)] = \forall x\, \neg P(x)$$

Is $f(n) \notin O(g(n))$?

- Show the negation holds

$$\neg[f(n) \in O(g(n))]$$

$$\neg[\exists M > 0, \exists n_0 > 0 \; s.t. \forall n > n_0, f(n) \leq Mg(n)]$$

$$\forall M > 0, \forall n_0 > 0 \; s.t. \exists n > n_0, f(n) > Mg(n)$$

Example

$$(2n^2 + 3) \notin \mathrm{O(n)}$$

$$\forall M > 0, \forall n_0 > 0 \; s.t. \; \exists n \geq n_0, \; 2n^2 + 3 > Mn$$

$$n = \mathrm{M} + n_0 + 1$$

$$n > 1$$

$$n^2 > n$$

$$n^2 + 3 > n$$

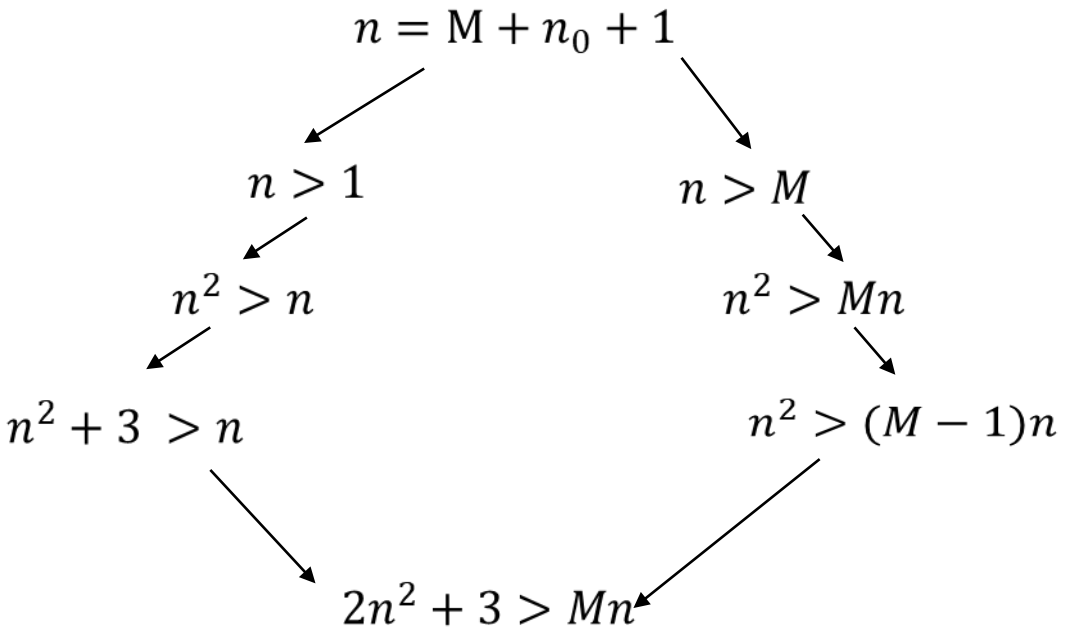
$$n > M$$

$$n^2 > Mn$$

$$n^2 > (M - 1)n$$

$$2n^2 + 3 > Mn$$

Diagram of proof



CPS 616

Slide 3

Dr. Yeganeh Bahoo

Example

$$(2n^2 + 50) \in O(n^2)$$

How do the proof

- Work backward
- To prove $f(n) \in O(g(n))$, it is important to find appropriate value of M and n_0
- To prove that $f(n) \notin O(g(n))$, it is important to find appropriate value of n

Formal definition of Ω = asymptotic lower bound

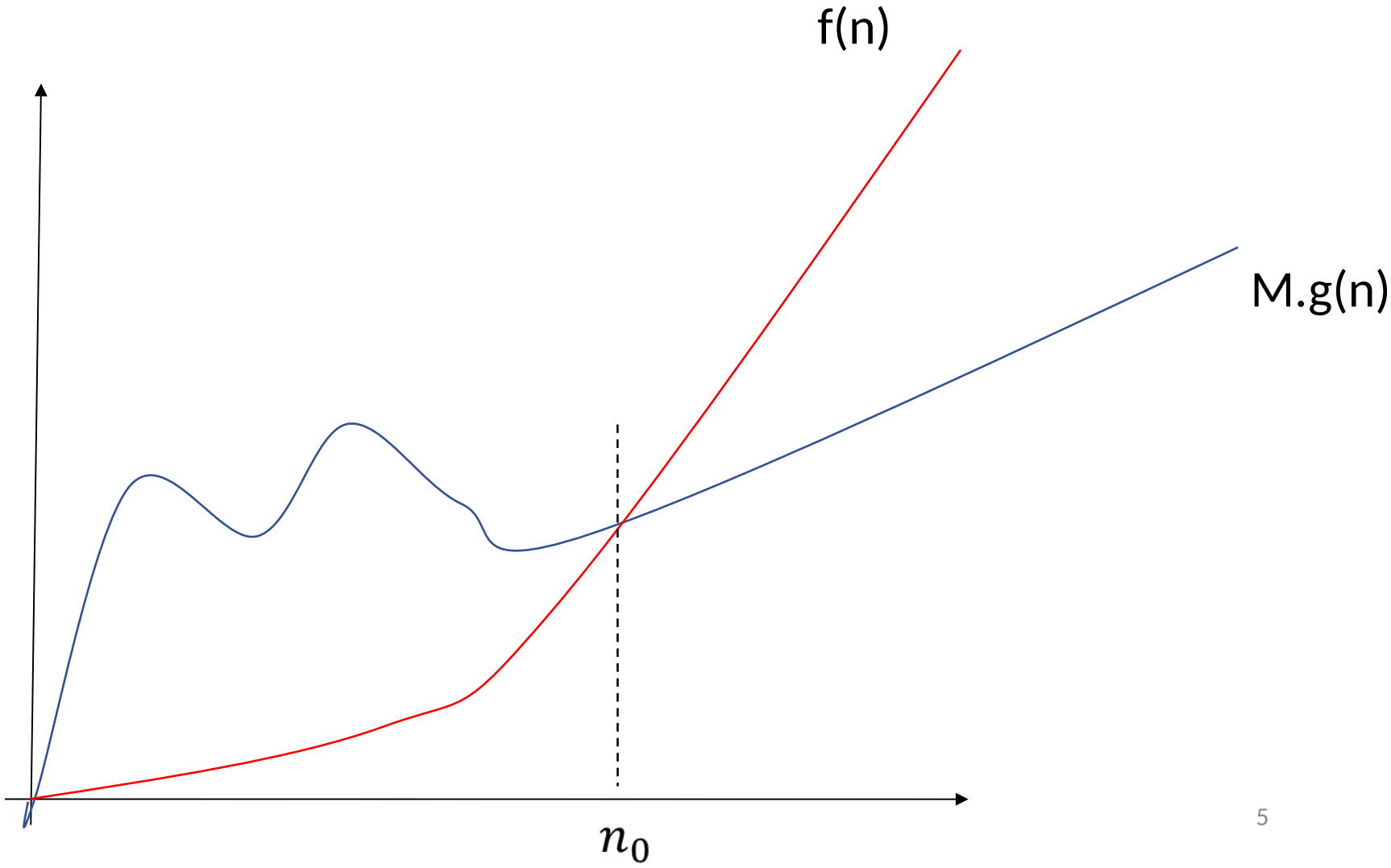
$f(n)$ is $\Omega(g(n))$ if there exist a constant $M > 0$ and $n_0 > 0$ such that

$$f(n) \geq M.g(n), \qquad for\ n > n_0$$

$$f(n) \in \Omega\big(g(n)\big) \Leftrightarrow \exists M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, f(n) \geq Mg(n)$$

Formal definition of the mathematical definition of Ω means …

$f(n) \in \Omega(g(n))$



Omega is a lower bound

$$1 < \log n < n^{1/4} < \sqrt{n} < n < n \log n < n\,log^2 n < n\sqrt{n} < n^2 < n^2 \log n < n^3 < 2^n < n! < n^n$$

$$10n^4 \in \Omega(n^4)$$
$$10n^4 \in \Omega(n^3)$$
$$10n^4 \in \Omega(n^2 \log n)$$
$$10n^4 \notin \Omega(n^5)$$

You can simplify your cost function by:

- Ignoring the constants
- Ignoring the functions with lower order of growth

Your cost function is of Omega of this simplified function, as well as any function with less order of growth

Example

- $f(n) = 8n \log n$
- $g(n) = 4n$

- $f(n) \in \Omega(\mathrm{g(n)})?$

$$\exists M > 0, \exists n_0 > 0 \; s.t. \forall n > n_0, f(n) \geq Mg(n)$$

Example

$M = 2, n_0 = 10$

$$8n \log n \in \Omega(4n)?$$

$$n > 10$$
$$\log n > \log 10$$
$$\log n > 1$$
$$n \log n > n$$
$$8n \log n > 8n$$
$$8n \log n > 2.4n$$
$$f(n) > Mg(n)$$
$$f(n) \geq Mg(n)$$

Theorem

- Given any functions f : $R^+ \to R^+$ and g : $R^+ \to R^+$

$$f(n) \in O\big(g(n)\big) \Leftrightarrow g(n) \in \Omega\big(f(n)\big)$$

$proof: \Rightarrow f(n) \in O\big(g(n)\big)$

$\exists M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, f(n) \leq Mg(n)$

$\exists M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, Mg(n) \geq f(n)$

$\exists M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, g(n) \geq \frac{1}{M} f(n)$

$\exists M' > 0, \exists n_0 > 0\ s.t. \forall n > n_0, g(n) \geq M' f(n), where\ M' = \frac{1}{M}$

$proof: (\Leftarrow)$ Proof is analogous to above

Theta notation

- $f(n) = n^2 \log n + n \log n + 3n + 12$

$$f(n) \in O(n^4)$$
$$f(n) \in O(n^2 \log n)$$

$O(n^2 \log n)$ express f(n) better!

Theta notation

- $f(n) = n^2 \log n + n \log n + 3n + 12$

$$f(n) \in O(n^2 \log n)$$
$$f(n) \in \Omega(n^2 \log n)$$

$$f(n) \in \boldsymbol{\theta}(n^2 \log n)$$

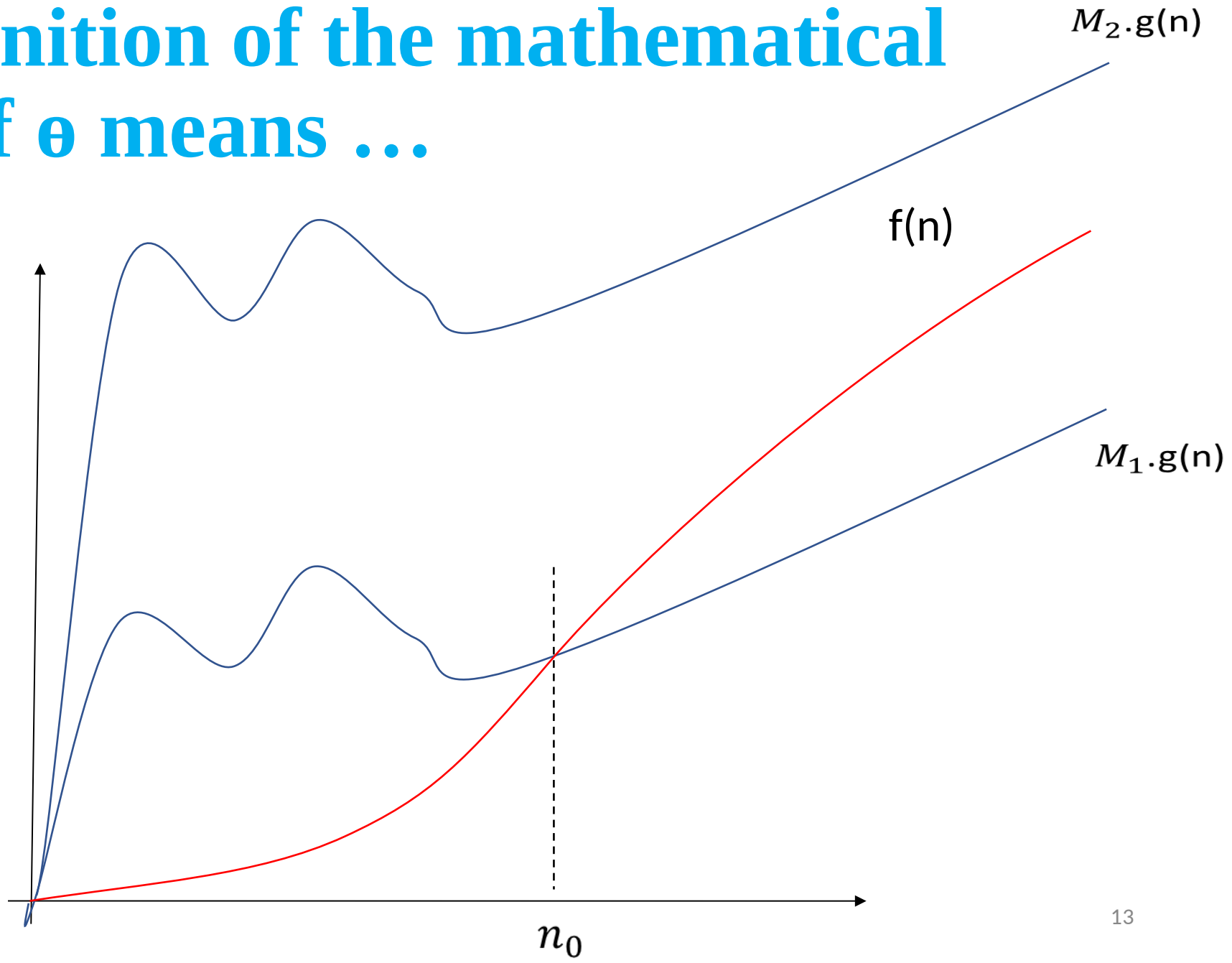
Formal definition of $\boldsymbol{\theta}$ = asymptotic average bound

$f(n)\ is\ \theta(g(n))$ if there exist a constant $M_1 > 0\ and\ M_2 > 0\ and\ n_0 > 0$ such that

$$M_1.g(n) \leq f(n) \leq M_2.g(n), \qquad for\ n > n_0$$

Formal definition of the mathematical definition of ɵ means …

$f(n) \in \theta(g(n))$



Theta is an asymptotic tight bound (upper and lower bound)

- $23n^3 \log n + n^2 + 13n \in \theta(n^3 \log n)$
- $23n^3 \log n + n^2 + 13n \in O(n^3 \log n)$
- $23n^3 \log n + n^2 + 13n \in O(n^4)$
- $23n^3 \log n + n^2 + 13n \in \boldsymbol{\Omega}(n^3 \log n)$
- $23n^3 \log n + n^2 + 13n \in \boldsymbol{\Omega}(n^3)$
- $23n^3 \log n + n^2 + 13n \notin \theta(n^3)$
- $23n^3 \log n + n^2 + 13n \notin \theta(n^4)$

Example

- $f(n) = 3n^3 + 9\,,\ g(n) = n^3,\qquad f(n) \in \theta\big(g(n)\big)?$

$M_1 = 3, M_2 = 4, n_0 = 3$

$$n > 3$$
$$n^3 > 27$$
$$n^3 > 9$$
$$3n^3 + 9 < 3n^3 + n^3 = 4n^3$$
$$3n^3 < 3n^3 + 9$$

$$M_1.g(n) \le f(n) \le M_2.g(n)$$

Theorem

$$f(n) \in \theta\big(g(n)\big) \Leftrightarrow [f(n) \in O\big(g(n)\big)\ and\ f(n) \in \Omega\big(g(n)\big)]$$

Proof: from the definition of O, Ω and θ

Little oh notation

- $f(n) \in O(n^3)$
- $f(n) \notin \theta(n^3)$

Ex.

$$n^{1-\varepsilon} \epsilon O(n), but\ n^{1-\varepsilon} \notin \theta(n)$$
$$n^{2-\varepsilon} \epsilon O(n^2), but\ n^{2-\varepsilon} \notin \theta(n^2)$$

We can say:

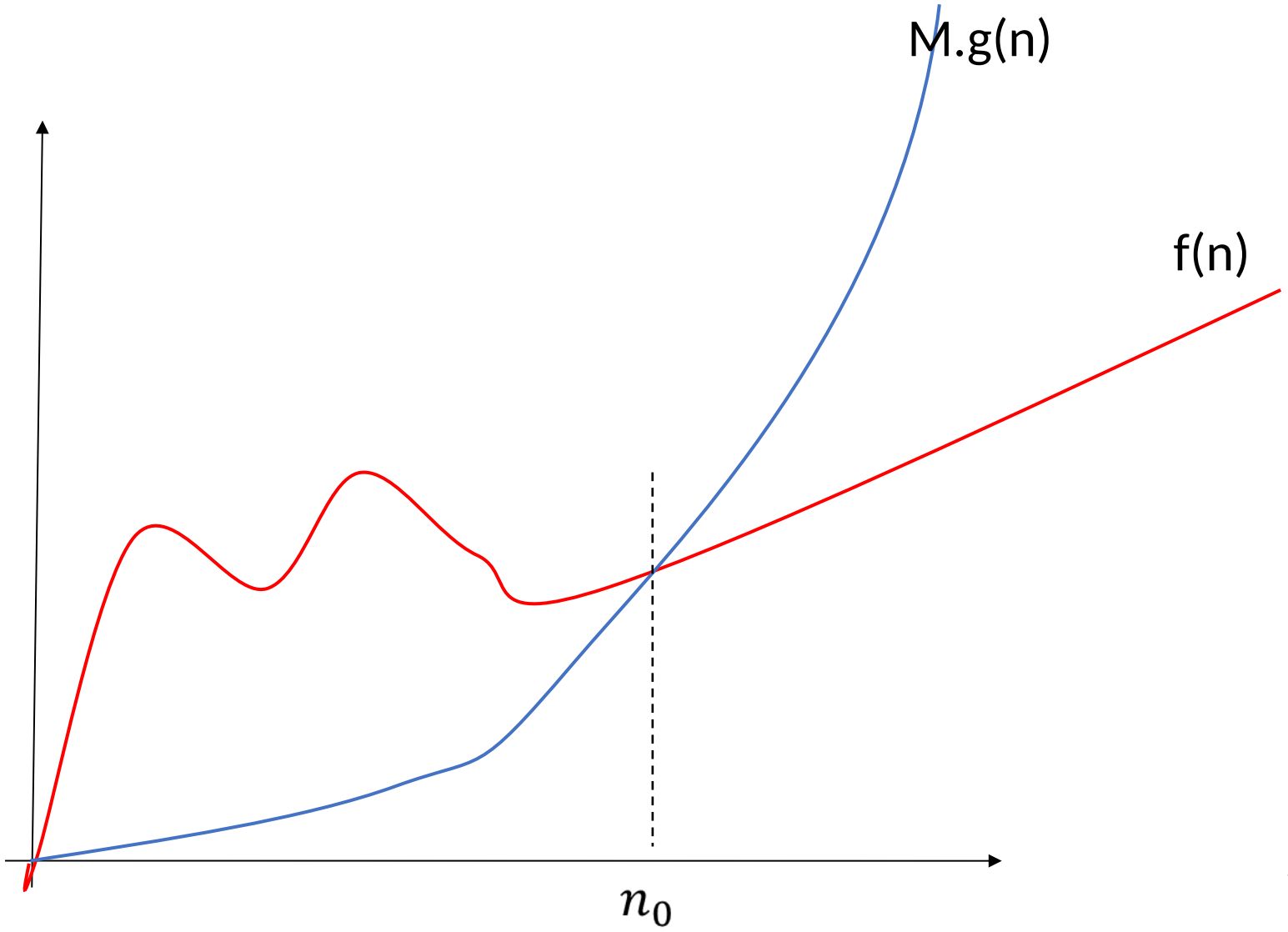
$$n^{1-\varepsilon} \epsilon o(n)$$
$$n^{2-\varepsilon} \epsilon o(n^2)$$

Little oh official notations

- $f(n) \in o(g(n)) \Leftrightarrow \forall M > 0, \exists n_0 > 0 \; s.t. \forall n > n_0, f(n) \leq Mg(n)$

g(n) is upper bound for f(n), BUT they grow with different rate!

Figure



Example

- $f(n) = 2n + 10, g(n) = n^2, \quad f(n) \in o\big(g(n)\big)?$

$$\forall M, n_0 = \max\{\sqrt{\frac{20}{M}}, \frac{4}{M}\}+1$$

$$n > \sqrt{\frac{20}{M}} \qquad n > \frac{4}{M}$$

$$n^2 > \frac{20}{M} \qquad Mn/2 > 2$$

$$Mn^2/2 > 10 \qquad Mn^2/2 > 2n$$

$$Mn^2 > 2n + 10$$

$$Mg(n) \geq f(n)$$

- $f(n) \notin o\big(g(n)\big)$?

$$\neg[\forall M > 0, \exists n_0 > 0 \; s.t. \forall n > n_0, f(n) \leq Mg(n)]$$

$$\exists M > 0, \forall n_0 > 0 \; s.t. \exists n > n_0, f(n) > Mg(n)]$$

Little omega notation

- $f(n) \in \Omega(g(n))$
- $f(n) \notin \theta(g(n))$

Ex.

$$n^{1+\varepsilon} \epsilon \Omega(n), but\ n^{1+\varepsilon} \notin \theta(n)$$
$$n^{2+\varepsilon} \epsilon \Omega(n^2), but\ n^{2+\varepsilon} \notin \theta(n^2)$$

We can say:

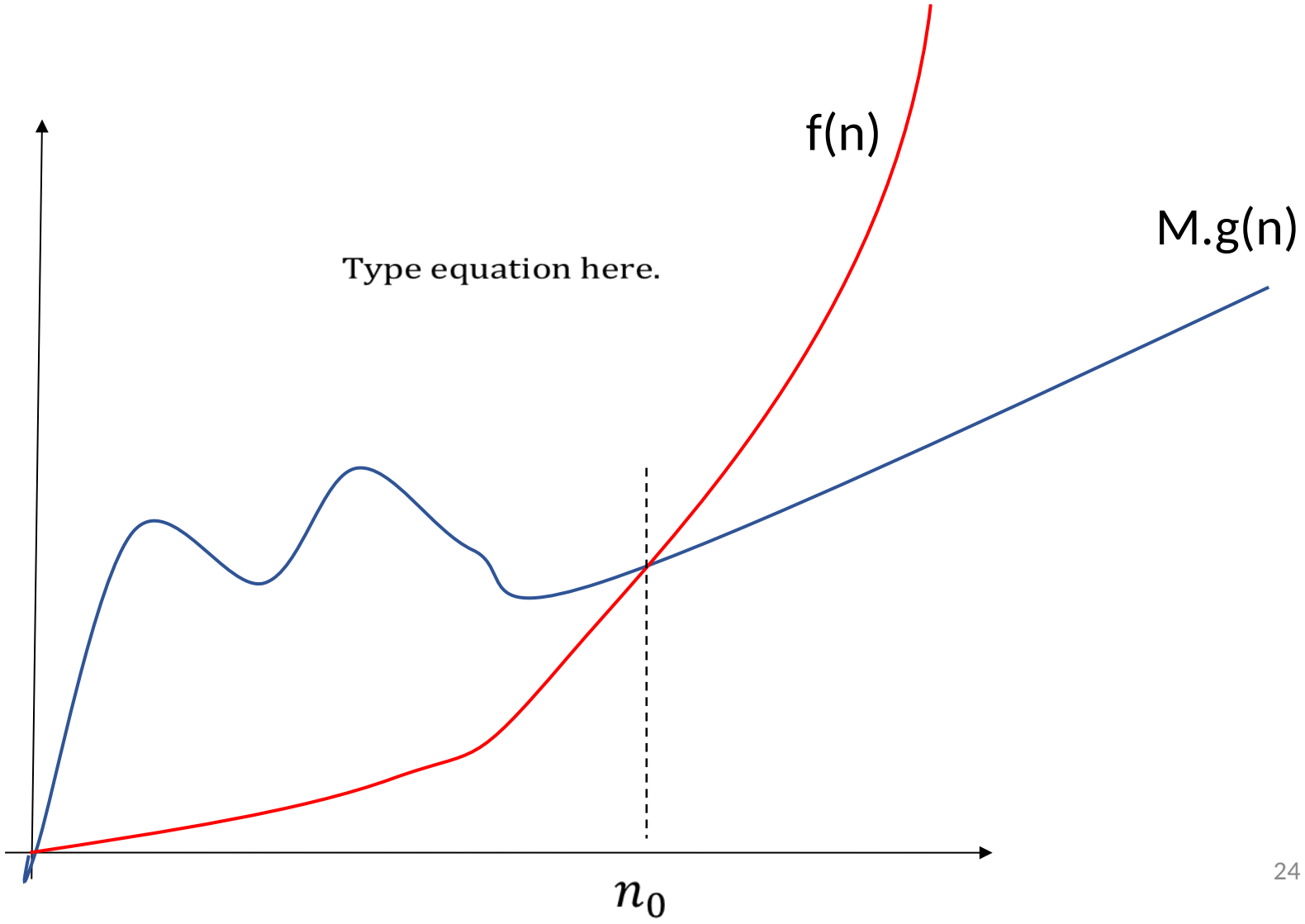
$$n^{1+\varepsilon} \epsilon \omega(n)$$
$$n^{2+\varepsilon} \epsilon \omega(n^2)$$

Little omega notation-formal definition

- $f(n) \in \omega\big(g(n)\big) \Leftrightarrow \forall M > 0, \exists n_0 > 0 \; s.t. \forall n > n_0, f(n) \geq Mg(n)$

g(n) is lower bound for f(n), BUT they grow with different rate!

Figure



Example

- $f(n) = 16n \log n + 4 \log n + 63$

$$f(n) \in O(n \log n)$$
$$f(n) \in \Omega(n \log n)$$
$$f(n) \in \theta(n \log n)$$
$$f(n) \notin o(n \log n)$$
$$f(n) \notin \omega(n \log n)$$

Example

- $f(n) = 16n \log n + 4 \log n + 63$

$$f(n) \in O(n^2 \log n)$$
$$f(n) \notin \Omega(n^2 \log n)$$
$$f(n) \notin \theta(n^2 \log n)$$
$$f(n) \in o(n^2 \log n)$$
$$f(n) \notin \omega(n^2 \log n)$$

Example

- $f(n) = 16n \log n + 4 \log n + 63$

$$f(n) \notin O(\log n)$$
$$f(n) \in \Omega(\log n)$$
$$f(n) \notin \theta(\log n)$$
$$f(n) \notin o(\log n)$$
$$f(n) \in \omega(\log n)$$

Theorem

- Given any functions f : $R^+ \to R^+$ and g : $R^+ \to R^+$

$$f(n) \in o(g(n)) \Leftrightarrow g(n) \in \omega(f(n))$$

$proof: \Rightarrow f(n) \in o(g(n))$

$\forall M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, f(n) \leq Mg(n)$

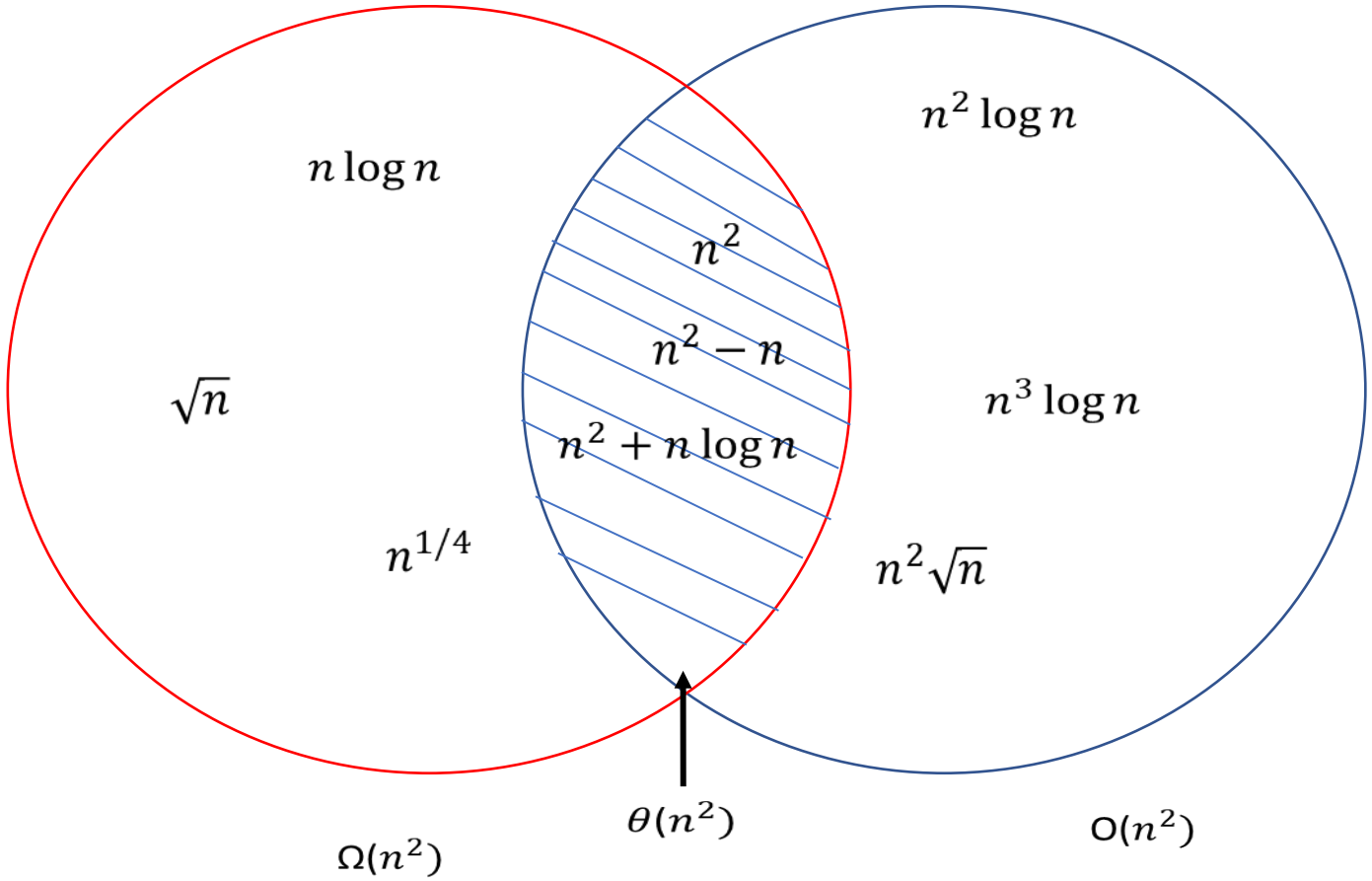
$\forall M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, Mg(n) \geq f(n)$

$\forall M > 0, \exists n_0 > 0\ s.t. \forall n > n_0, g(n) \geq \frac{1}{M} f(n)$

$\forall M' > 0, \exists n_0 > 0\ s.t. \forall n > n_0, g(n) \geq M' f(n), where\ M' = \frac{1}{M}$

$proof: (\Leftarrow)$ Proof is analogous to above

Overview


$n \log n$
$\sqrt{n}$
$n^{1/4}$
$n^2 \log n$
n^2
$n^2 - n$
$n^2 + n \log n$
$n^3 \log n$
$n^2\sqrt{n}$
$\Omega(n^2)$
$\theta(n^2)$
$O(n^2)$

What in mathematics gives us the growth of functions?

- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = c$

For some nice function, lim gives us the answer !

$c=0 \Rightarrow f(n) \in o\big(g(n)\big)$

$c=\infty \Rightarrow f(n) \in \omega\big(g(n)\big)$

$c \in R^{+} \Rightarrow f(n) \in \theta(g(n))$

Examples

- $\lim_{n\to\infty}\left(\frac{12\,n^3+23\,n}{6n^3}\right)$
- $\lim_{n\to\infty}\left(\frac{2\,n^6+12}{6n^5}\right)$
- $\lim_{n\to\infty}\left(\frac{2\,n^2+1}{6n^3}\right)$

Important

- For some functions the limit does not exist!
- Ex. $f(n) = 3\cos n\,, g(n) = 12$
- $\lim_{n\to\infty}(\frac{3\cos n}{12})$
- What to do?

  We must use the definition!

Summery

Determining the cost of an algorithm:

1. Find the cost function
2. Express it by asymptotic notations

Asymptotic notations are the tools to help us to compare the time complexity of the algorithms!

CPS 616

Part 4

Dr. Yeganeh Bahoo

Recurrence Relations
Reference:
CLRS

- Chapters:
  - 4.2
  - 4.3
  - 4.4
  - 4.5

Cost function

- Input: an array A of integers
- Output: finding the minimum

```
int min=A[0];
while(i<A.length)
{
   if (A[i]<min)
       min=A[i]
   i++;
}
```

$$f(n) = 1 + \sum_{x=0}^{n-1} 3$$

$$f(n) = 1 + 3n$$

Cost function

```
void main(int n){
int i=0;
int result=0;
for (int j=1;j<n;j++)
   { i++;
    result=Add(i, result);
}
}
```

```
int add(int I, int result){
return i+result;
}
```

$$f(n) = 2 + \sum_{x=1}^{n-1} (1 + h(x))$$

$$\in O(n)$$

Example: addition of n numbers

$$\overbrace{1+2+3+4+5+\ldots.+12+\ldots.,n-1}^{\text{Add(n-1)}}+n$$

$$Add(n) = \begin{cases} 1 & ,n = 1 \\ Add(n-1) + n & ,n! = 1 \end{cases}$$

Example: addition of n numbers

```
Add(n)
{
    if n==1:                    A
            return 1;
    Otherwise:
            return Add (n-1) + n;      B
}
```

Example: addition of n numbers

Cost:

$$f(n) = \begin{cases} 1 & ,n = 1 \quad \text{A} \\ f(n-1) + 1 & ,n! = 1 \quad \text{B} \end{cases}$$

Another example

- $Fac(n) = \begin{cases} 1 & ,n = 1 \\ Fac(n-1).n & ,n! = 1 \end{cases}$

Example: factorial of n numbers

```
Fac(n)
{
    If n==1:                    } A
            return 1;
    Otherwise:
            return Fac (n-1) . n;   } B
}
```

Example: factorial of n numbers

Cost:

$$f(n) = \begin{cases} 1 & ,n = 1 \quad \text{A} \\ f(n-1) + 1 & ,n! = 1 \quad \text{B} \end{cases}$$

But

- We need to have the time f(n) in terms of n, NOT the function f!

We need to solve the recurrence relations!

Finding the Cost of Recursive Algorithm

- Find a recurrence relation
- Solve the recurrence relation
- Correctness proof by induction

$$Add(n) = \begin{cases} 1 & , n = 1 \\ Add(n-1) + n & , n! = 1 \end{cases}$$

$$f(n) = O(n)$$

Solve the recurrence relation

- Substitution method
- Recurrence tree
- Master Theorem

Substitution method

- To unwind f(n)!

Example: addition of n numbers

- $f(n) = f(n-1) + 1$, $f(1)=1$
- $= \underbrace{f(n-2) + 1}_{f(n-1)} + 1$
- $= \underbrace{f(n-3) + 1}_{f(n-2)} + 1 + 1$
- $= \underbrace{f(n-4) + 1}_{f(n-3)} + \overbrace{1 + 1 + 1}^{4}$
- $= f(n-i) + \underbrace{1 + 1 + 1 + \cdots + 1}_{i}$
- $= \underbrace{f(1)}_{f(1)} + \underbrace{1 + 1 + 1 + \cdots + 1}_{n-1}$
- $= 1 + n - 1$

General pattern

Base case: $n - i = 1$

Example

- We now believe that $f(n) = n$
- But we need to prove it by induction

Reminder:

- Induction

To prove $\forall k \geq 1\ P(k)$

1. Show the base case $P(1) holds$
2. Show $\forall k \geq 1\ P(k) \rightarrow P(k+1)$

- Strong Induction

To prove $\forall k \geq 1\ P(k)$

1. Show the base case $P(1) holds$
2. Show $\forall k \geq 1\ [P(1)\ and\ P(2) and\ \ldots P(k)] \rightarrow P(k+1)$

Proof

- The base case: for some c $P(c)$
- Hypothesis: for a $k \geq c$ $P(k)$
- Proof: $P(k+1)$ holds
- Formal conclusion: $\forall n \geq c$ $P(n)$

Example: addition function

- $\forall n \geq 1\ f(n) = n$
- Base case: $f(1) = 1$
- Hypothesis: for a $k \geq 1\ f(k) = k$
- Proof: show $f(k+1) = k+1$

By the recursion function: $f(k+1) = f(k) + 1$

By hypothesis $f(k) = k \Rightarrow f(k+1) = k+1$

Conclusion: The base holds and the inductive hypothesis concludes the inductive step. So, we have shown $f(n) = n$

Another example

```
void DecToBin (int n)
{
  if (n==1)
       print (n);
  else
      { DecToBin(n/2);
         print (n%2);}
}
```

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

Solving the recurrence function

- $T(n) = T\left(\frac{n}{2}\right) + 1$
- $= T\left(\frac{n}{2^2}\right) + 1 + 1$
- $= T\left(\frac{n}{2^3}\right) + 1 + 1 + 1$
- $= T\left(\frac{n}{2^i}\right) \underbrace{+ 1 + \cdots + 1}_{i}$
- $= T(1) + \log_2 n$
- $= 1 + \log_2 n$

Base case: $\frac{n}{2^i} = 1$

$i = \log_2(n)$

Strong induction

- $\forall n \geq 1\ f(n) = 1 + \log_2 n$

- Base case: $f(1) = 1 + \log_2 1$

  - For a k>1 ,

    $for\ i \in 1,2,\ldots,k, f(k) = 1 + \log_2(k)$

- Proof: show $f(k+1) = 1 + \log_2 (k+1)$

Induction

- Proof: show $f(k+1) = 1 + \log_2(k+1)$
- By the recursion function: $f(k+1) = f\left(\frac{k+1}{2}\right) + 1$

By hypothesis $f\left(\frac{k+1}{2}\right) = 1 + \log_2(k+1)/2$

$$\Rightarrow f(k+1) = \log_2(k+1)/2 + 1 + 1 = \log_2(k+1)/2 + 2\log_2 2 = \log_2 \frac{k+1}{2}.2^2 = \log_2 2(k+1) = \log_2 2 + \log_2(k+1)$$

1

Conclusion: The base holds and the inductive hypothesis concludes the inductive step.

Solving the recurrence function (more example)

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T\left(\frac{n}{4}\right) + 1 & otherwise \end{cases}$$

- $T(n) = 3T\left(\frac{n}{4}\right) + 1$
- $= 3\left(3T\left(\frac{n}{4^2}\right) + 1\right) + 1 = 3^2T\left(\frac{n}{4^2}\right) + 3 + 1$
- $= 3^2\left(3T\left(\frac{n}{4^3}\right) + 1\right) + 3 + 1 = 3^3T\left(\frac{n}{4^3}\right) + 3^2 + 3 + 1$
- $= 3^iT\left(\frac{n}{4^i}\right) + 3^{i-1} + \cdots + 3 + 1$ General pattern

- $= 3^iT(1) + 3^{i-1} + \cdots + 3 + 1$ Base case: $n = 4^i \Rightarrow i = \log_4 n$
- $= 3^{\log_4 n} + \sum_{a=0}^{\log_4 n - 1} 3^a$
- $= \sum_{a=0}^{\log_4 n} 3^a$

Solving the recurrence function (more example)

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T\left(\frac{n}{4}\right) + 1 & otherwise \end{cases}$$

- $T(n) = 3T\left(\frac{n}{4}\right) + 1$
- $= 3\left(3T\left(\frac{n}{4^2}\right) + 1\right) + 1 = 3^2T\left(\frac{n}{4^2}\right) + 3 + 1$
- $= 3^2\left(3T\left(\frac{n}{4^3}\right) + 1\right) + 3 + 1 = 3^3T\left(\frac{n}{4^3}\right) + 3^2 + 3 + 1$
- $= 3^iT\left(\frac{n}{4^i}\right) + 3^{i-1} + \cdots + 3 + 1$

General pattern

- $= 3^iT(1) + 3^{i-1} + \cdots + 3 + 1$

Base case: $n = 4^i \Rightarrow i = \log_4 n$

- $= 3^{\log_4 n} + \sum_{a=0}^{\log_4 n - 1} 3^a$
- $= \sum_{a=0}^{\log_4 n} 3^a$
- $= \frac{3^{\log_4 4n} - 1}{2}$

$$S_n = a_0 . \frac{1 - q^n}{1 - q} \Rightarrow 1 . \frac{1 - 3^{(\log_4 n + 1)}}{1 - 3} = \frac{3^{(\log_4 n + 1)} - 1}{2} = \frac{3^{\log_4 4n} - 1}{2}$$

Solving the recurrence function (more example)

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T\left(\frac{n}{4}\right) + 1 & otherwise \end{cases}$$

- $T(n) = 3T\left(\frac{n}{4}\right) + 1$
- $= 3\left(3T\left(\frac{n}{4^2}\right) + 1\right) + 1 = 3^2T\left(\frac{n}{4^2}\right) + 3 + 1$
- $= 3^2\left(3T\left(\frac{n}{4^3}\right) + 1\right) + 3 + 1 = 3^3T\left(\frac{n}{4^3}\right) + 3^2 + 3 + 1$
- $= 3^iT\left(\frac{n}{4^i}\right) + 3^{i-1} + \cdots + 3 + 1$
- General pattern
- $= 3^iT(1) + 3^{i-1} + \cdots + 3 + 1$
- $= 3^{\log_4 n} + \sum_{a=0}^{\log_4 n - 1} 3^a$

Base case: $n = 4^i \Rightarrow i = \log_4 n$

- $= \frac{3^{\log_4 4n} - 1}{2}$

$a^{\log_c b} = b^{\log_c a} \Rightarrow 3^{\log_4 n} = n^{\log_4 3}$

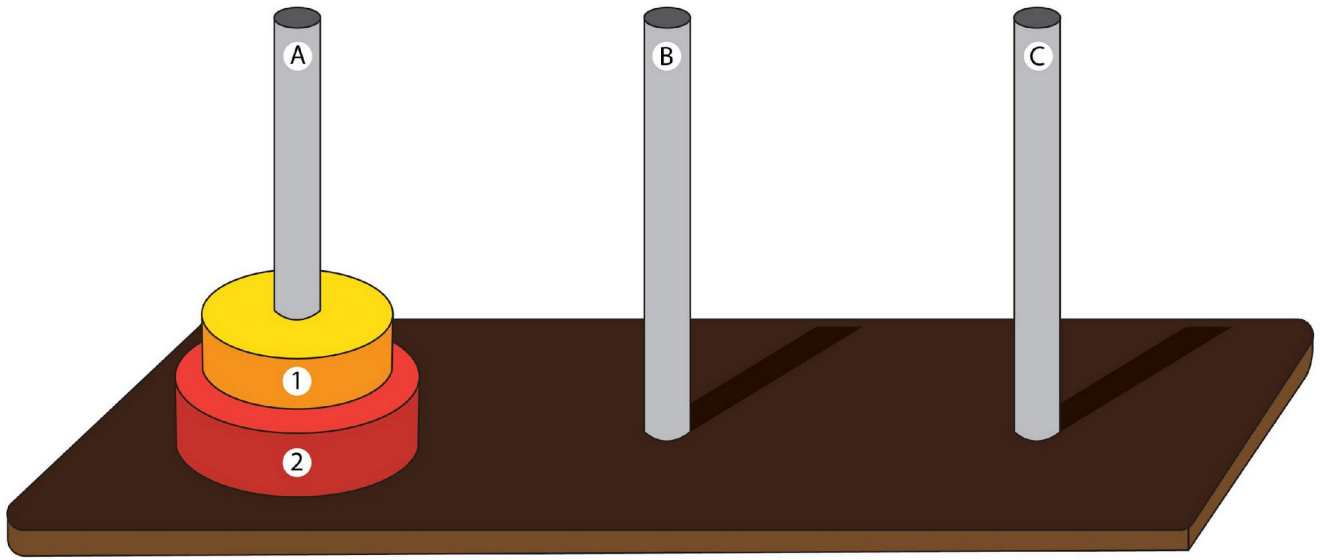
- $= \frac{(4n)^{\log_4 3}}{2} - \frac{1}{2}$

Hanoi tower

Input: n disks with different radius, sorted on the column A

Process: Transfer all disks to column B, with help of column C

Rule: At no time a disk with radius r can be put on top of a disk with radius less than r



CPS616

Part 5

Dr. Yeganeh Bahoo

Cost function of recursive functions

- Input: an array A of integers
- Output: finding the minimum

```
int min=0;
for (int i=0;i<A.length;i++)
{
   if (A[i]<min)
       min=A[i]
   i++;
}
```

$$f(n) = 1 + \sum_{x=0}^{n-1} 3$$

$$f(n) = 1 + 3n$$

Cost function

```
void main(int n){
int i=0;
int result=0;
for (int j=1;j<n;j++)
   { i++;
    result=Add(i, result);
}
}
```

```
int add(int I, int result){
return i+result;
}
```

$$f(n) = 2 + \sum_{x=1}^{n-1} (1 + h(x))$$

$$\in O(n)$$

Example: addition of n number

Add(n-1)

$$1+2+3+4+5+\ldots.+12+\ldots.,n-1+n$$

$$Add(n) = \begin{cases} 1 & ,n = 1 \\ Add(n-1) + n & ,n! = 1 \end{cases}$$

Example: addition of n number

```
Add(n)
{
    If n==1:
            return 1;
    Otherwise:
            return Add (n-1) + n;
}
```

A

B

Example: addition of n number

Cost:

$$f(n) = \begin{cases} 1 & ,n = 1 \quad \text{A} \\ f(n-1) + 1 & ,n! = 1 \quad \text{B} \end{cases}$$

But

- We need to have the time f(n) in terms of n NOT f!

We need to solve the recurrence relations!

Finding the Cost of Recursive Algorithm

- Find a recurrence relation
- Solve the recurrence relation
- Correctness proof by induction

$$f(n) = \begin{cases} 1 & ,n = 1 \\ f(n-1) + n & ,n! = 1 \end{cases}$$

$$f(n) = O(n)$$

Solve the recurrence relation

- Substitution method
- Recurrence tree
- Master Theorem

Substitution method

- To unwind f(n)!

Solving the recurrence function (more example)

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{3}\right) + n & otherwise \end{cases}$$

- $T(n) = 2T\left(\frac{n}{3}\right) + n$
- $= 2\left(2T\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n = 2^2T\left(\frac{n}{3^2}\right) + \frac{2}{3}n + n$
- $= 2^2\left(2T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}\right) + n = 2^3T\left(\frac{n}{3^3}\right) + \frac{2^2}{3^2}n + \frac{2}{3}n + n$
- $= 2^iT\left(\frac{n}{3^i}\right) + n\sum_{a=0}^{i-1}\left(\frac{2}{3}\right)^a$ General pattern
- $= 2^{\log_3 n} + n\sum_{a=0}^{\log_3 n - 1}\left(\frac{2}{3}\right)^a$ Base case: $n = 3^i \Rightarrow i = \log_3 n$

Solving the recurrence function (more example)

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{3}\right) + n & otherwise \end{cases}$$

- $T(n) = 2T\left(\frac{n}{3}\right) + n$
- $= 2\left(2T\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n = 2^2T\left(\frac{n}{3^2}\right) + \frac{2}{3}n + n$
- $= 2^2\left(2T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}\right) + n = 2^3T\left(\frac{n}{3^3}\right) + \frac{2^2}{3^2}n + \frac{2}{3}n + n$
- $= 2^iT\left(\frac{n}{3^i}\right) + n\sum_{a=0}^{i-1}\left(\frac{2}{3}\right)^a$ General pattern
- $= 2^{\log_3^n} + n\sum_{a=0}^{\log_3^n - 1}\left(\frac{2}{3}\right)^a$ Base case: $n = 3^i \Rightarrow i = \log_3^n$
- $= n^{\log_3^2} + \cdots$ $a^{\log_c^b} = b^{\log_c^a} \Rightarrow 2^{\log_3^n} = n^{\log_3^2}$

Solving the recurrence function (more example)

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{3}\right) + n & otherwise \end{cases}$$

- $T(n) = 2T\left(\frac{n}{3}\right) + n$
- $= 2\left(2T\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n = 2^2 T\left(\frac{n}{3^2}\right) + \frac{2}{3}n + n$
- $= 2^2\left(2T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}\right) + n = 2^3 T\left(\frac{n}{3^3}\right) + \frac{2^2}{3^2}n + \frac{2}{3}n + n$
- $= 2^i T\left(\frac{n}{3^i}\right) + n\sum_{a=0}^{i-1}\left(\frac{2}{3}\right)^a$
- $= 2^{\log_3 n} + n\sum_{a=0}^{\log_3 n - 1}\left(\frac{2}{3}\right)^a$
- $= n^{\log_3 2} + \cdots$
- $= n^{\log_3 2} + n.3$

General pattern

Base case: $n = 3^i \Rightarrow i = \log_3 n$

$$q < 1 \Rightarrow S_n = a_0.\frac{1}{1-q} \Rightarrow 1.\frac{1}{1-\frac{2}{3}} = 3$$

Solving the recurrence function (more example)

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

$\log_2 n = \log n$

- $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$
- $= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n = 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n$
- $= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\log\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n$
- $= 2^3T\left(\frac{n}{2^3}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n$

- $= 2^iT\left(\frac{n}{2^i}\right) + \sum_{a=0}^{i-1} n\log n/2^a$ General pattern
- $= 2^{\log_2 n}T(1) + \sum_{a=0}^{\log_2 n - 1} n\log n/2^a$ Base case: $n = 2^i \Rightarrow i = \log_2 n$

Solving the recurrence function (more example)

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

- $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$
- $= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n = 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log \mathrm{n}$
- $= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\log\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log \mathrm{n}$
- $= 2^3T\left(\frac{n}{2^3}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log \mathrm{n}$

- $= 2^iT\left(\frac{n}{2^i}\right) + \sum_{a=0}^{i-1} n\log n/2^a$
- $= 2^{\log_2^n}T(1) + \sum_{a=0}^{\log_2^n - 1} n\log\frac{n}{2^a}$
- $= n^{\log_2^2}.1 + \cdots = n + \cdots$

General pattern

Base case: $n = 2^i \Rightarrow i = \log_2^n$

$a^{\log_c^b} = b^{\log_c^a} \Rightarrow 2^{\log_2^n} = n^{\log_2^2}$

Suppose the base of log is 2

- $\sum_{a=0}^{\log_2 n-1} n \log \frac{n}{2^a} = n \sum_{a=0}^{\log_2 n-1} \log \frac{n}{2^a} = n \sum_{a=0}^{\log_2 n-1} (\log n - \log 2^a)$
- $= n(\sum_{a=0}^{\log_2 n-1} \log n - \sum_{a=0}^{\log_2 n-1} \log 2^a)$
- $= n \log n \sum_{a=0}^{\log_2 n-1} 1 - n \sum_{a=0}^{\log_2 n-1} a$
- $= n \log n . \log n - n \frac{(\log n - 1) \log n}{2}$
- $= \frac{n \log^2 n}{2} + \frac{n \log n}{2}$

$\sum (a - b) = \sum a - \sum b$

$\log a^b = b \log a$

Solving the recurrence function (more example)

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

- $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$
- $= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n = 2^2T\left(\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n$
- $= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\log\frac{n}{2^2}\right) + n\log\frac{n}{2} + n\log n$
- $= 2^3T\left(\frac{n}{2^3}\right) + n\log\frac{n}{2^2} + n\log\frac{n}{2} + n\log n$

- $= 2^iT\left(\frac{n}{2^i}\right) + \sum_{a=0}^{i-1} n\log n/2^a$
- $= 2^{\log_2 n}T(1) + \sum_{a=0}^{\log_2 n - 1} n\log\frac{n}{2^a}$
- $= n^{\log_2 2}.1 + \cdots = n + \cdots$
- $= n + \frac{n\,log^2 n}{2} + \frac{n\log n}{2}$

General pattern

Base case: $n = 2^i \Rightarrow i = \log_2 n$

Solve the recurrence relation

- Substitution method
- Recurrence tree
- Master Theorem

Recurrence tree

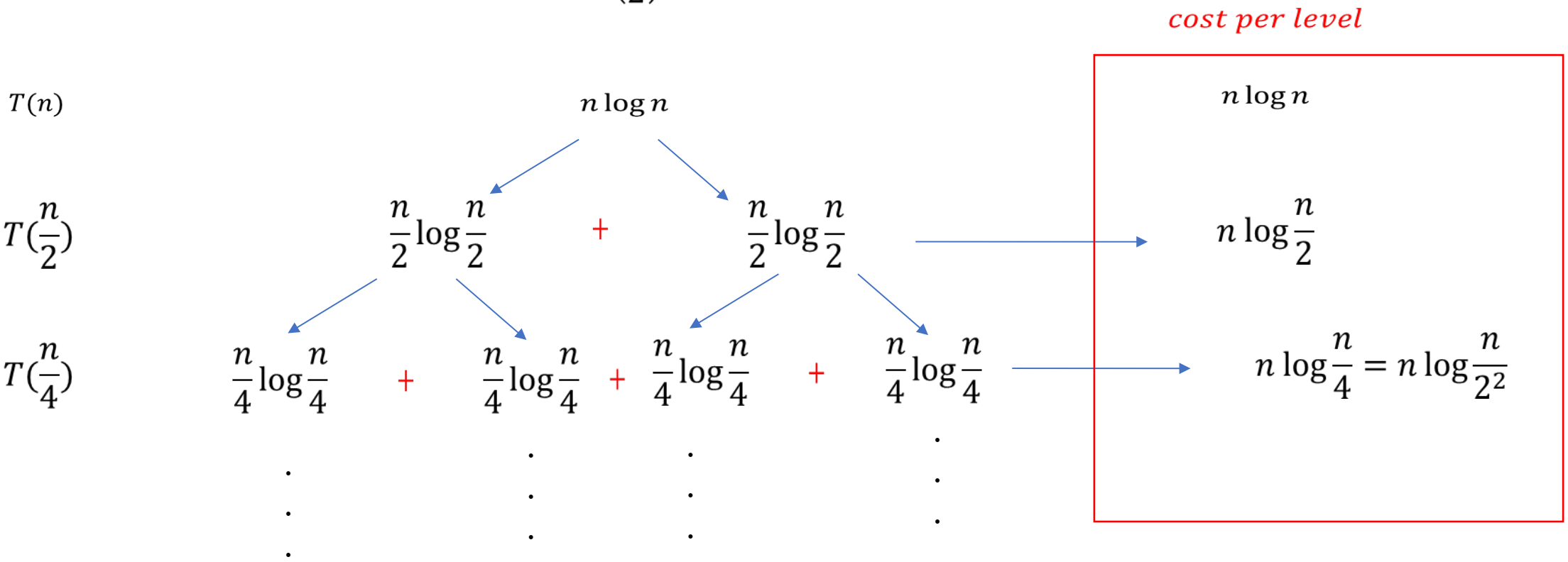
- We can use the recursion tree to unwind the recurrence relation as well by summing the local cost at each level!

Example

Base case: T(1)=1

$$\log_2 n = \log n$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$



$$base\ case: \frac{n}{2^i} = 1 \Rightarrow i = \log_2 n$$

$$\Rightarrow T(n) = \sum_{x=0}^{\log_2 n} n\log\frac{n}{2^x}$$

$$\Rightarrow T(n) = \theta(n\ log^2\ n)$$

$\log_2 n = \log n$

- $\sum_{a=0}^{\log_2 n} n \log \frac{n}{2^a} = n \sum_{a=0}^{\log_2 n} \log \frac{n}{2^a} = n \sum_{a=0}^{\log_2 n} (\log n - \log 2^a)$
- $= n(\sum_{a=0}^{\log_2 n} \log n - \sum_{a=0}^{\log_2 n} \log 2^a)$
- $= n \log n \sum_{a=0}^{\log_2 n} 1 - n \sum_{a=0}^{\log_2 n} a$
- $= n \log_2 n . (\log_2 n + 1) - n \frac{\log_2 n . (\log_2 n + 1)}{2}$
- $= \frac{n Log_2^2 n + n \log_2 n}{2}$

$\sum (a - b) = \sum a - \sum b$

$\log a^b = b \log a$

Recurrence tree

- Reminder: after solving the recurrence relation, we must prove it using induction!

***

$$\Rightarrow T(n) = \theta(n\ log^2\ n)$$

$$\text{First} :\Rightarrow T(n) = O(n\ log^2\ n)$$

$$M = 3, n_0 = 2 \Rightarrow \forall n > 2\ T(n) \leq 3nlog^2 n$$

Proof by induction over n

Base case: $T(2) \leq 3.2\ log^2 2$

RS: $3.2\ log^2 2 = 3.2.1 = 6$

LS: $T(2) = 2T\left(\frac{2}{2}\right) + 2\log 2 = 2.1 + 2 = 4$

$\Rightarrow 4 \leq 6$ $\quad base\ case\ holds$

Assumption:

$\forall k' \in \{2,3, \dots, k-1\}\ T(k') \leq 3.k'log^2\ k'$

***

$$\Rightarrow T(n) = \theta(n\ log^2\ n)$$

$$\text{First} :\Rightarrow T(n) = O(n\ log^2\ n)$$

$$M = 3, n_0 = 2 \Rightarrow \forall n > n_0\ \ T(n) \leq 3nlog^2 n$$

Inductive step:

$$T(k) \leq 3.k\ log^2\ \ k$$

$$T(k) = 2T\left(\frac{k}{2}\right) + k\log k$$

$$\leq 2.3.\frac{k}{2}\ log^2\frac{k}{2} + k\log k$$

$$\leq 3.k\ (\log k\ - \log 2)^2 + k\log k$$

$$\leq 3.k\ (\log k\ - 1)^2 + k\log k$$

$$\leq 3.k\ (log^2 k + 1\ - 2\log k)\quad + k\log k$$

$$\leq 3.k\ .log^2 k + 3.k\ - 5.k\log k$$

X

$$k > 2$$

$$\log k > \log 2$$

$$\log k > 1$$

$$\log k > \frac{3}{5}$$

$$5\log k > 3$$

$$5.\mathrm{k}\log k > 3.k$$

$$0 > 3.k - 5.\mathrm{k}\log k$$

$$3.klog^2 k > 3.k - 5.\mathrm{k}\log k + 3.klog^2 k$$

X

$$T(k) \leq x \leq 3klog^2 k \Rightarrow T(k) \leq 3klog^2\ k$$

***

- Conclusion:

The base case holds and the inductive hypothesis concludes the inductive step. So, we have shown that $T(n) \in O(nlog^2\ n)$

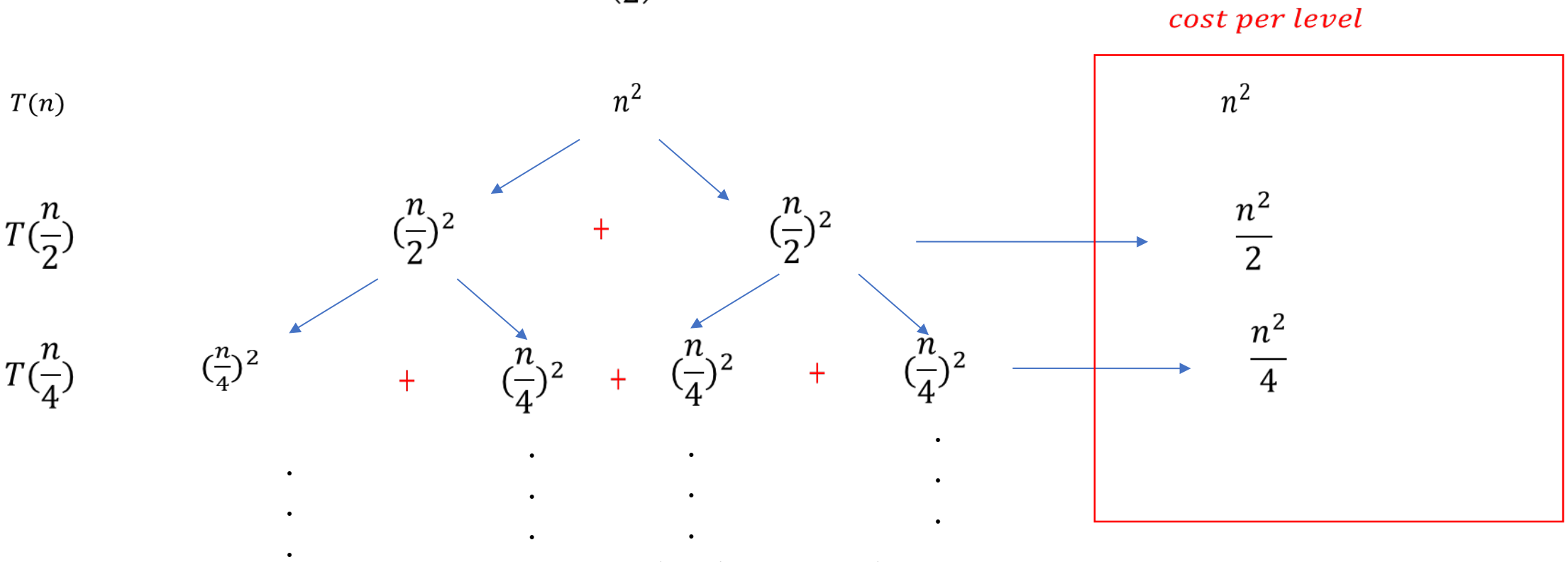
Similar proof can be used to show that $T(n) \in \Omega(nlog^2\ n)$

As a result $T(n) \in \theta(nlog^2\ n)$

Example

Base case: T(1)=1

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



$$base\ case: \frac{n}{2^i} = 1 \Rightarrow i = \log_2 n \Rightarrow T(n) = n^2\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\log_2 n}}\right)$$

$$\Rightarrow T(n) = 2n^2$$

$$q < 1 \Rightarrow S_n = a_0 . \frac{1}{1-q} \Rightarrow 1 . \frac{1}{1-\frac{1}{2}} = 2$$

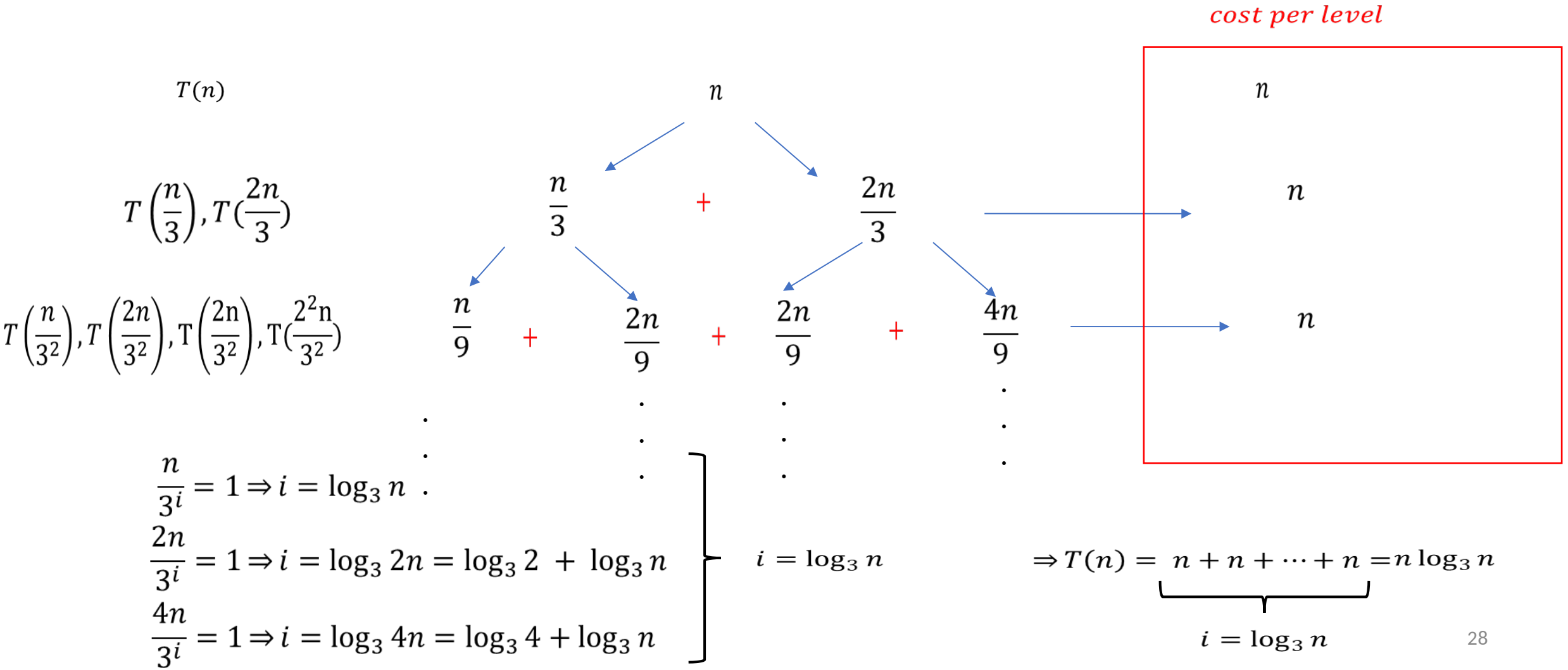
Recurrence tree

- If we are careful about the base cases, the recurrence tree gives the exact solution.
- Typically recurrence tree gives an estimate of the asymptotic notation.

Example

Base case: T(1)=1

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$$



CPS 616

Part 6

Dr. Yeganeh Bahoo

Example: addition of n number

Cost:

$$f(n) = \begin{cases} 1 & ,n = 1 \\ f(n-1) + 1 & ,n! = 1 \end{cases}$$

But

- We need to have the time f(n) in terms of n NOT f!

We need to solve the recurrence relations!

Finding the Cost of Recursive Algorithm

- Find a recurrence relation
- Solve the recurrence relation
- Correctness proof by induction

$$f(n) = \begin{cases} 1 & , n = 1 \\ f(n-1) + 1 & , n! = 1 \end{cases}$$

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

Solve the recurrence relation

- Substitution method
- Recurrence tree
- Master Theorem

Master Theorem

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

$$When\ a \geq 1, b > 1, c \geq 1, d \geq 0$$

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

$$1.\ IF\ f(n) \in O\left(n^{\log_b a - \varepsilon}\right) \Rightarrow T(n) = \theta(n^{\log_b a})$$

$$2.\ IF\ f(n) \in \theta(n^{\log_b a}) \Rightarrow T(n) = \theta(n^{\log_b a}.\log n)$$

$$3.\ IF\ f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right) and\ \exists \kappa > 0, n_0 > 0\ s.t. \forall n > n_0\ \ af(n/b) \leq \kappa f(n) \Rightarrow T(n) \in \theta(f(n))$$

A :Cost of recursive function

B :Local cost

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

1. *The overall cost is dominated by recursive part:* $A > B$
2. *The cost of recurssive and local cost are asymtotically equivalant:* $A = B$
3. *The overall cost is dominated by the local cost* $B > A$

$$\boldsymbol{IF\ f(n) \in O\left(n^{log_b a - \varepsilon}\right) \Rightarrow T(n) = \theta\left(n^{log_b a}\right)}$$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$a = 7, b = 2, \Rightarrow \log_2 7 = 2.807$

$f(n) = n^2 \in O(n^{2.807-\varepsilon})$

$\mathrm{T}(n) \in \theta(n^{\log_b a}) \epsilon \theta(n^{2.807})$

$$\boldsymbol{IF\ f(n) \in \theta(n^{log_b a}) \Rightarrow T(n) = \theta(n^{log_b a}.log\, n)}$$

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

$a = 1, b = 2 \Rightarrow \log_b a = \log_2 1 = 0$

$f(n) = 1 \epsilon \theta\left(n^{\log_b a}\right) = \theta(n^0) = \theta(1)$

$\mathrm{T}(n) \in \theta\left(n^{\log_b a} \log n\right) = \theta\left(n^0 \log n\right) = \theta(\log n)$

$IF\ f(n) \in \Omega\left(n^{log_b\ a+\varepsilon}\right) and\ \exists \kappa > 0, n_0 > 0\ s.t.\ \forall n > n_0\ \ af(n/b) \leq \kappa f(n) \Rightarrow T(n) \in \theta(f(n))$

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

$$a = 2, b = 2 \Rightarrow \log_b a = \log_2 2 = 1$$

$$f(n) = n^2 \epsilon \Omega\left(n^{\log_b a+\varepsilon}\right) = \Omega(n^{1+\varepsilon})$$

$$\mathrm{af}\left(\frac{\mathrm{n}}{\mathrm{b}}\right) = 2\mathrm{f}\left(\frac{\mathrm{n}}{2}\right) = 2.(n/2)^2 = \frac{n^2}{2} \leq \left(\frac{1}{2}\right).f(n)$$

$$T(n) \in \theta\left(f(n)\right) = \theta(n^2)$$

Master Theorem - limits

- It is not always applicable!

$$T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$

$a = 2, b = 2 \Rightarrow \log_b a = \log_2 2 = 1$

$f(n) = n\log n \notin O(n^{1-\varepsilon})$

$f(n) = n\log n \notin \theta(n^1)$

$f(n) = n\log n \notin \Omega(n^{1+\varepsilon})$

Master Theorem - limits

- Recurrence relation are not always in the form represents in Master theorem!

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$T(n) = T\left(\frac{n}{5}\right) + 7T\left(\frac{n}{10}\right) + n$$

This recurrence can be solved exactly with other approaches!

Master Theorem

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

$$When\ a \geq 1, b > 1, c \geq 1, d \geq 0$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

$$a = 4, b = 2, f(n) = \log n$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = \log n \in ?$$

1. $IF\ f(n) \in O\left(n^{\log_b a - \varepsilon}\right) \Rightarrow T(n) = \theta(n^{\log_b a})$

~~2. $IF\ f(n) \in \theta(n^{\log_b a}) \Rightarrow T(n) = \theta(n^{\log_b a} . \log n)$~~

~~3. $IF\ f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right) and\ \exists \kappa > 0, n_0 > 0\ s.t. \forall n > n_0\ \ af(n/b) \leq \kappa f(n) \Rightarrow T(n) \in \theta(f(n))$~~

Extra Examples

$$T(n) = 4T\left(\frac{n}{2}\right) + n\log n$$

Extra Examples

$$T(n) = T(\sqrt{n}) + n \log n$$

Extra Examples

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

Extra Examples

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

Extra Examples

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

***Simpler version of Master Theorem

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

$$When\ a \geq 1, b > 1, c \geq 1, d \geq 0$$

$$if\ af\left(\frac{n}{b}\right) = \kappa f(n)\ \ for\ some\ \kappa > 1 \Rightarrow T(n) \in \theta(n^{\log_b a})$$

$$if\ af\left(\frac{n}{b}\right) = f(n)\ \ \Rightarrow T(n) \in \theta(f(n)\log n)$$

$$if\ af\left(\frac{n}{b}\right) = \kappa f(n)\ \ for\ some\ \kappa < 1 \Rightarrow T(n) \in \theta(f(n))$$

***This applies on some cases that Master Theorem applies

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

$$a = 7, b = 2, f(n) = n^2$$

$$af\left(\frac{n}{b}\right) = 7f\left(\frac{n}{2}\right) = 7.(n/2)^2 = \frac{7n^2}{4} = 7/4.f(n)$$

$$if\ af\left(\frac{n}{b}\right) = \kappa f(n)\ \ for\ some\ \kappa > 1 \Rightarrow T(n) \in \theta(n^{\log_b a})$$

$$\frac{7}{4} > 1 \Rightarrow T(n) \in \theta\left(n^{\log_b a}\right) = \theta(n^{\log_2 7})$$

$***IF\ f(n) \in O\left(n^{log_b\, a-\varepsilon}\right) \Rightarrow T(n) = \theta(n^{log_b\, a})$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$a = 7, b = 2, \Rightarrow \log_2 7 = 2.807$

$f(n) \in n^2 \in O(n^{2.807-\varepsilon})$

$f(n) \in \theta(n^{\log_b a}) \epsilon \theta(n^{2.807})$

***This applies on some cases that Master Theorem applies, but not all

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

$$a = 4, b = 2, f(n) = \log n$$

$$af\left(\frac{n}{b}\right) = 4f\left(\frac{n}{2}\right) = 4.\log n/2 = 4\log n\ - 4\ \log_2 2$$

$$\log a/b = \log a\ - \log b$$

$$\neq \kappa \log n\ for\ any\ \kappa$$

$$\Rightarrow\ simple\ version\ of\ Master\ Theorem\ does\ not\ apply!$$

But what about actual Master Theorem?

***Master Theorem

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

$$When\ a \geq 1, b > 1, c \geq 1, d \geq 0$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

$$a = 4, b = 2, f(n) = \log n$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = \log n \in ?$$

1. $IF\ f(n) \in O\left(n^{\log_b a - \varepsilon}\right) \Rightarrow T(n) = \theta(n^{\log_b a})$

~~2. $IF\ f(n) \in \theta(n^{\log_b a}) \Rightarrow T(n) = \theta(n^{\log_b a} . \log n)$~~

~~3. $IF\ f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right) and\ \exists \kappa > 0, n_0 > 0\ s.t. \forall n > n_0\ \ af(n/b) \leq \kappa f(n) \Rightarrow T(n) \in \theta(f(n))$~~

CPS616

Part 7

Dr. Yeganeh Bahoo

Up to now

- Analysis of the algorithm

a. Cost function

b. Asymptotic notation

c. Finding the cost of recursive algorithms

This let us compare different algorithms by measuring the time complexity of each one!

Algorithmic Techniques: Greedy Algorithm

- Greedy algorithm
- Divide and conquer
- Dynamic programming
- Randomized algorithm

Reference (CLRS)

- 16.1, 16.2, 16.3
- 23.1, 23.2

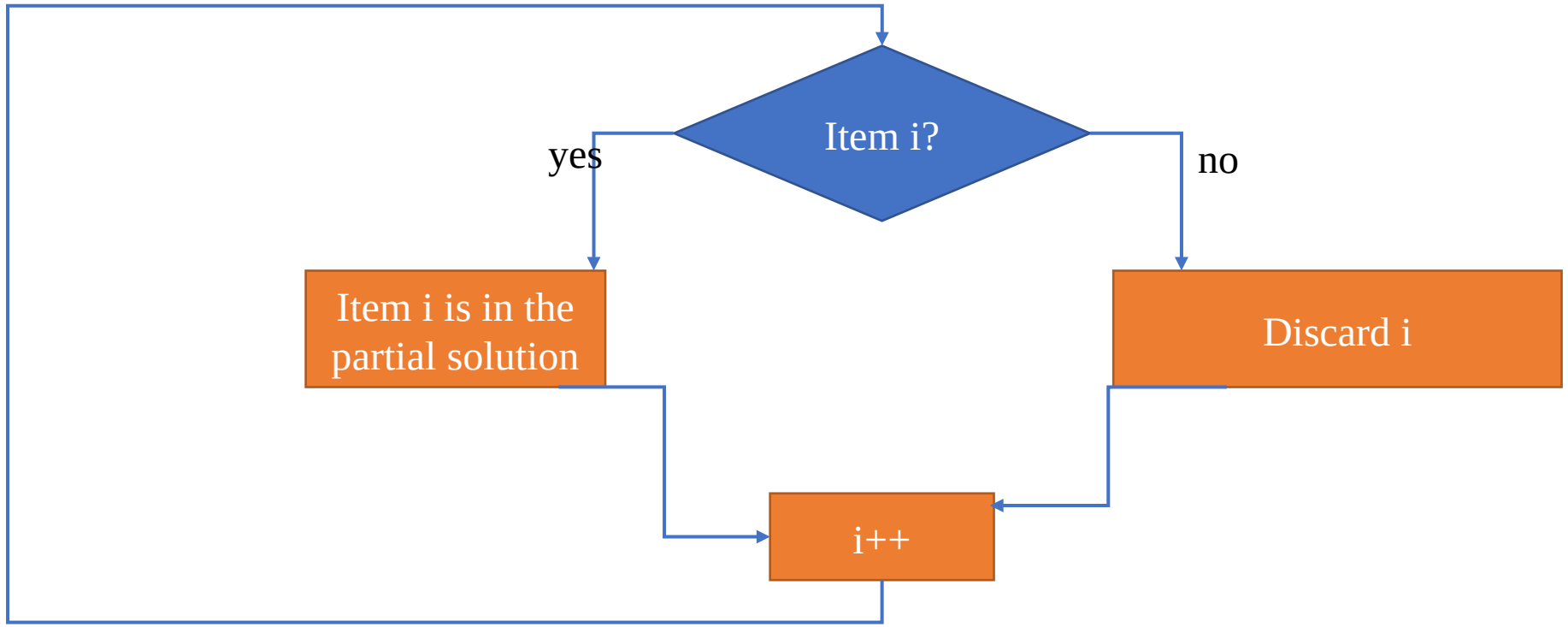
Optimization Problem (definition)

- Finding the best solution for a given problem, in terms of cost or benefit, while there exist several feasible solutions!

Greedy algorithm

- Based on local optimization!
- It selects at each step based on selection function
- There is no backtracking in these algorithms

How greedy algorithm works



0-1 Knapsack Problem

- n items,
- Each item i has the value of c_i and weight of w_i.

- Goal: filling a bag pack with capacity m such that the bag pack has the maximum value

We whether take an item or not, we are not allowed to take an item partially

m=10

Item number	Weight	value	
1	3	$1	
2	8	$2	
3	4	$1	
4	2	$2	
5	5	$2	

Brute force algorithm

- Examine all the feasible sub sets
- Take the optimal solution
- Time complexity: 2^n

Instead of making all possible choice we make a greedy choice

0-1 knapsack problem

Item number	Weight	value	
1	3	$1	
2	8	$2	
3	4	$1	
4	2	$2	
5	5	$2	

Possible greedy strategy:

1. add largest remaining item
2. add most valuable remaining item
3. add densest remaining item

Greedy properties (whether a greedy algorithm can solve your problem)

- **Optimal substructure:** an optimal solution includes optimal sub solutions

- **Greedy choice properties:** choosing a locally optimal choice leads to the globally optimal solutions

Fractional Knapsack Problem

- n items,
- Each item i has the value of c_i and weight of w_i.

- Goal: filling a bag pack with capacity m such that the bag pack has the maximum value

m=10

	weight	value	
1	3	$1	
2	8	$2	
3	4	$1	
4	2	$2	
5	5	$2	

Greedy Strategy -first

- Sort the items based on $\boldsymbol{w_i}$
- Put the item with most $\boldsymbol{w_i}$ in the bag pack completely if the bag has space!
- If the bag does not have enough space put a fraction of item i in bag so that the bag get filled completely.

m=10

	weight	value	
1	3	$1	
2	8	$2	
3	4	$1	
4	2	$2	
5	5	$2	

$$2 + 2\left(\frac{2}{5}\right) = 2.8$$

Greedy Strategy - second

- Sort the items based on c_i
- Put the item with most c_i in the bag pack completely if the bag has space!
- If the bag does not have enough space put a fraction of item i in bag so that the bag get filled completely.

m=10

Sorted list: item 4, item 5, item 2 , item 1, item 3

	weight	value	
1	3	$1	
2	8	$2	
3	4	$1	
4	2	$2	
5	5	$2	

$2 + 2 + 3\left(\frac{2}{8}\right) = 4.75$

Greedy Strategy-third

- Sort the items based on $\boldsymbol{c_i/w_i}$
- Put the item with most $\boldsymbol{c_i/w_i}$ in the bag pack completely if the bag has space!
- If the bag does not have enough space put a fraction of item i in bag so that the bag get filled completely.

m=10

	weight	value	value/weight (c_i/w_i)
1	3	$1	1/3
2	8	$2	¼
3	4	$1	¼
4	2	$2	1
5	5	$2	2/5

$$2(1) + 5(2/5) + 3\left(\frac{1}{3}\right) = 5$$

Why Greedy

- It is fast
- Easy to implement
- Easy to understand

Reminder: applying the greedy algorithm does not always give an optimal solution!

If greedy algorithms does not give the optimal solution…

- If we can prove what is the worst case maximum distance between the result of the algorithm and the optimal solution

  approximation algorithm

- If there is no such a proof

  heuristic approach

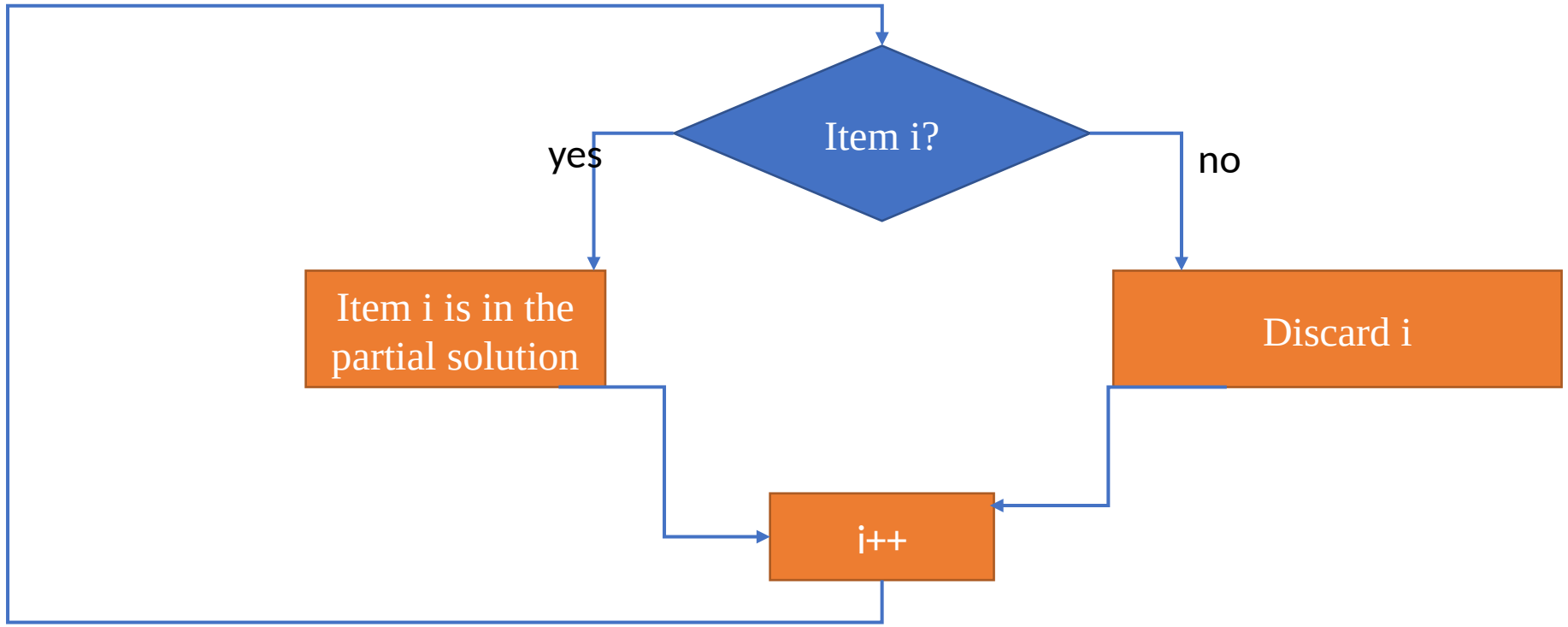
Ex. Activity selection

- There are n tasks using a resource which they cannot be shared.
- Each task i is shown by t_i.
- For each t_i two parameters $s_i \; and \; f_i$ are given as the time a task starts and finishes respectively.

- Goal: choose the most possible activities!

Tasks	Start time	Final time
1	2	13
2	6	10
3	5	7
4	0	6
5	8	11
6	3	5
7	1	4
8	8	12
9	12	14
10	5	9

How greedy algorithm works



tasks	Start time	Final time
1	2	13
2	6	10
3	5	7
4	0	6
5	8	11
6	3	5
7	1	4
8	8	12
9	12	14
10	5	9

Greedy solution

- We want the most tasks that are compatible!

1. Sort the tasks based on their f_i
2. Take the task with minimum f_i
3. Remove tasks which intersect with this task
4. Continue until the entire sorted list is processed

Tasks	Start time	Final time
1	2	13
2	6	10
3	5	7
4	0	6
5	8	11
6	3	5
7	1	4
8	8	12
9	12	14
10	5	9

One processor n tasks

- We have one processor and n tasks.
- Each task has the deadline time of d_i.
- If a task happens after d_i , the fine f_i get applied.
- Goal: scheduling with minimum fine!

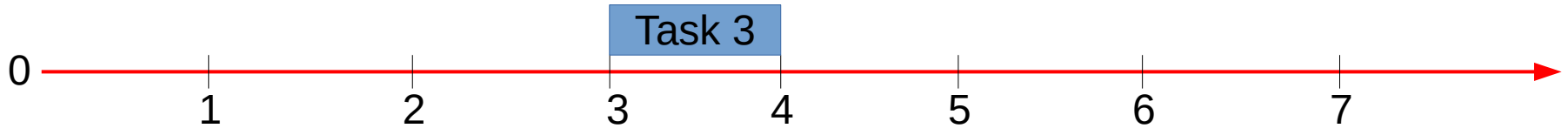
Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50

Greedy strategy

- Sort the task in terms of their fine incrementally.
- Put task i before d_i if there is any interval available before d_i (pick the right most if there is more than one),
- Otherwise we have to schedule i with delay! To do this, we schadual it after we process the rest of the tasks.

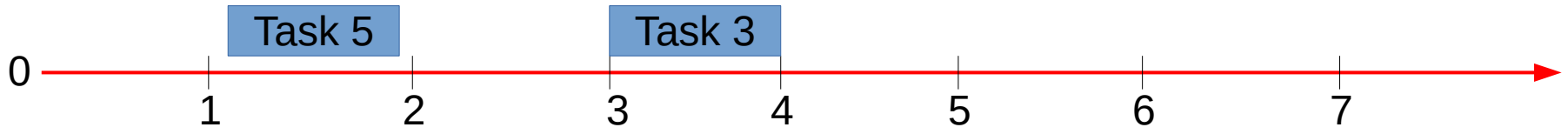
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



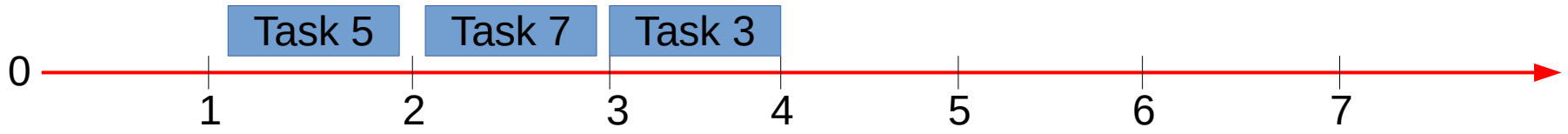
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



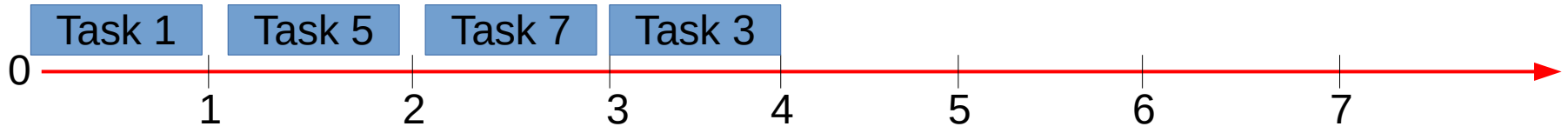
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



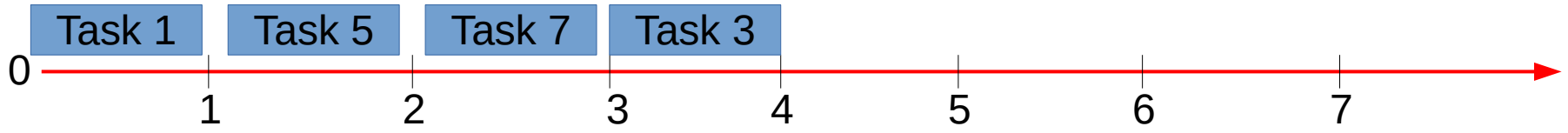
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



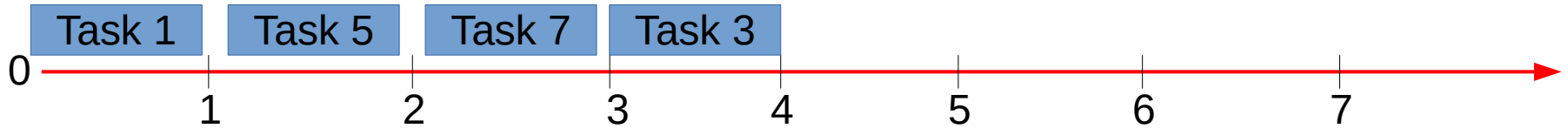
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



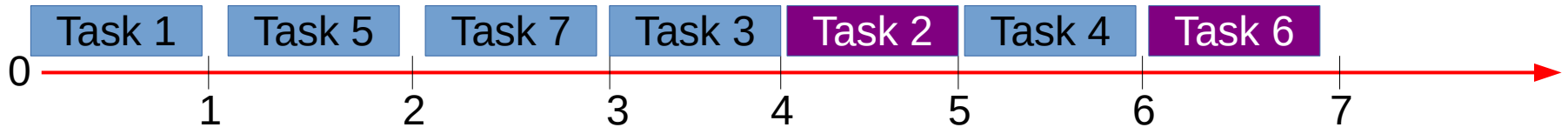
Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50



Task 3, Task 5, Task 7, Task 1, Task 2, Task 6, Task 4

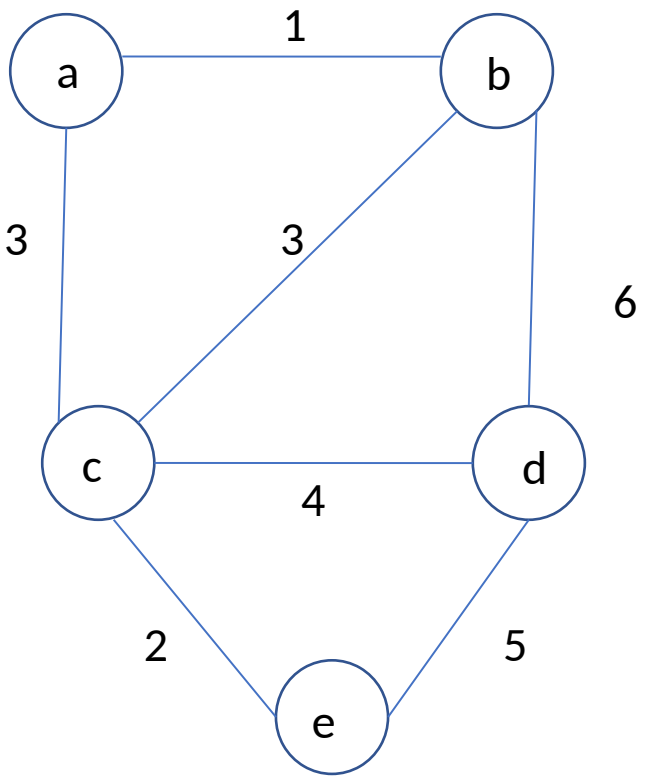
Tasks	Deadline	Fine
1	3	40
2	1	30
3	4	70
4	6	10
5	2	60
6	4	20
7	4	50

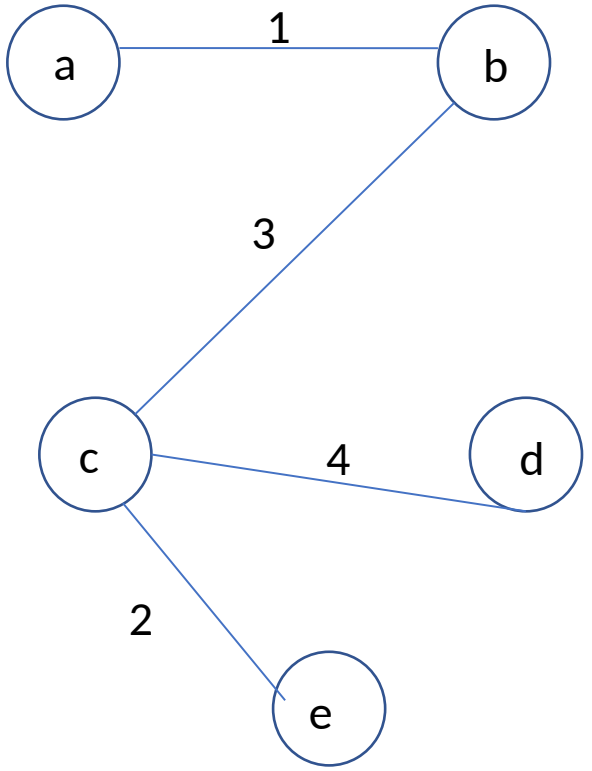


Next

- Some graph algorithms!

Minimum Spanning Tree





Prime

- Starting point

1. Empty set of edges
2. A set Y includes an arbitrary vertex of the graph

Prime

$F = \emptyset$

$Y = \{v_1\}$

$While\ (the\ instance\ is\ not\ solved)$

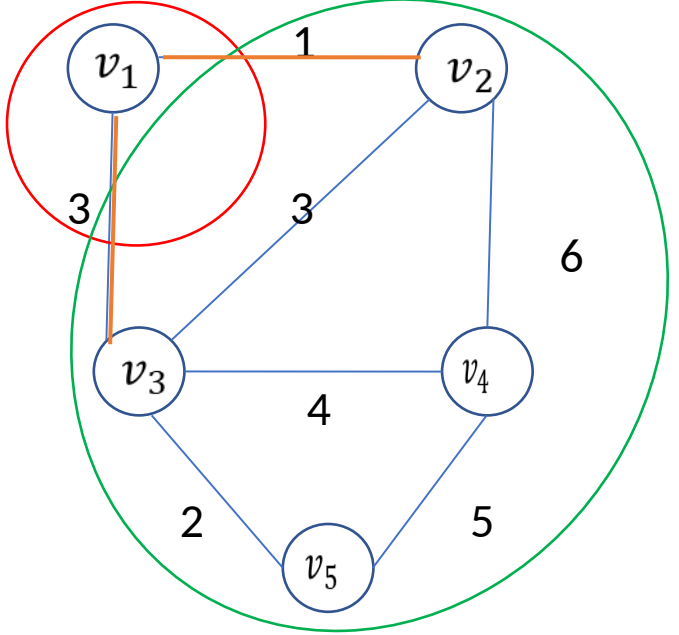
$\{\ Select\ a\ vertex\ v\ in\ the\ set\ V - Y\ which\ is\ closest\ to\ Y;$

$Add\ v\ to\ Y;$

$Add\ the\ corresponding\ edge\ to\ F;$

$If\ (V == Y)$

$The\ instance\ is\ solved;\quad \}$



$F = \emptyset \quad \Rightarrow F = \{v_1 v_2\}$

$Y = \{v_1\} \quad \Rightarrow Y = \{v_1, v_2\}$

$V - Y = \{v_2, v_3, v_4, v_5\}$

Prime

$F = \emptyset$

$Y = \{v_1\}$

$While\ (the\ instance\ is\ not\ solved)$

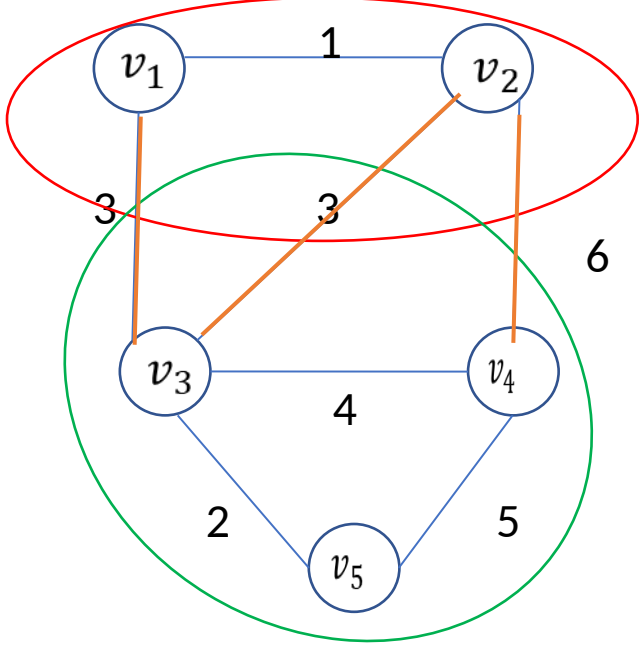
$\{\ Select\ a\ vertex\ v\ in\ the\ set\ V - Y\ which\ is\ closest\ to\ Y;$

$Add\ v\ to\ Y;$

$Add\ the\ corresponding\ edge\ to\ F;$

$If\ (V == Y)$

$The\ instance\ is\ solved;\quad \}$



$F = \{v_1v_2\} \quad \Rightarrow F = \{v_1v_2, v_2v_3\}$

$Y = \{v_1, v_2\} \quad \Rightarrow Y = \{v_1, v_2, v_3\}$

$V - Y = \{v_3, v_4, v_5\}$

Prime

$F = \emptyset$

$Y = \{v_1\}$

$While\ (the\ instance\ is\ not\ solved)$

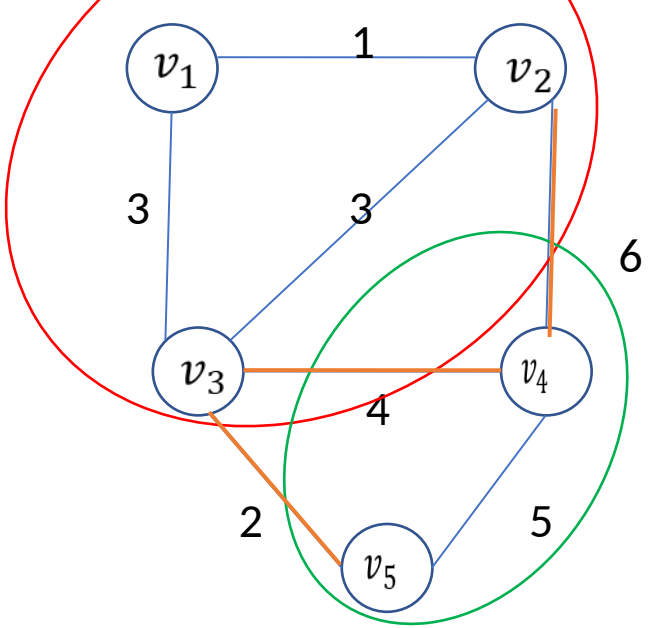
$\{\ Select\ a\ vertex\ v\ in\ the\ set\ V - Y\ which\ is\ closest\ to\ Y;$

$Add\ v\ to\ Y;$

$Add\ the\ corresponding\ edge\ to\ F;$

$If\ (V == Y)$

$The\ instance\ is\ solved;\quad \}$



$F = \{v_1v_2, v_2v_3\} \quad \Rightarrow F = \{v_1v_2, v_2v_3, v_3v_5\}$

$Y = \{v_1, v_2, v_3\} \quad \Rightarrow Y = \{v_1, v_2, v_3, v_5\}$

$V - Y = \{v_4, v_5\}$

Prime

$F = \emptyset$

$Y = \{v_1\}$

$While\ (the\ instance\ is\ not\ solved)$

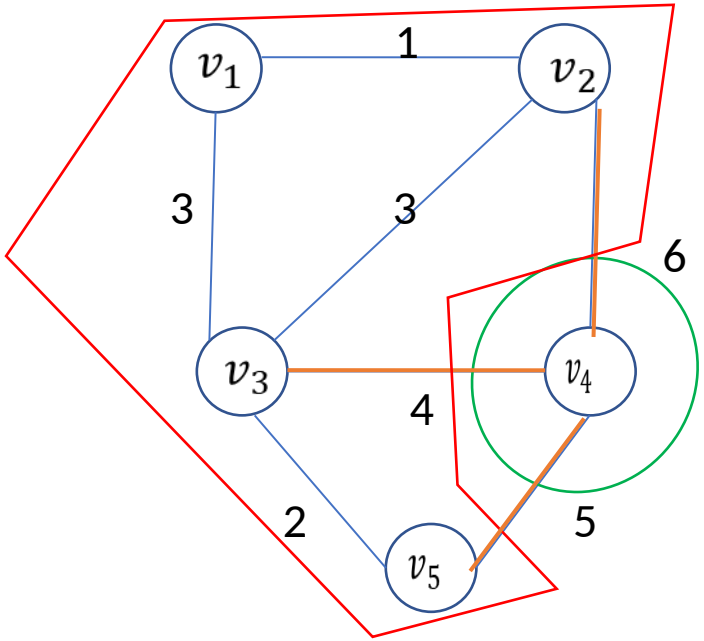
$\{\ Select\ a\ vertex\ v\ in\ the\ set\ V - Y\ which\ is\ closest\ to\ Y;$

$Add\ v\ to\ Y;$

$Add\ the\ corresponding\ edge\ to\ F;$

$If\ (V == Y)$

$The\ instance\ is\ solved;\quad \}$



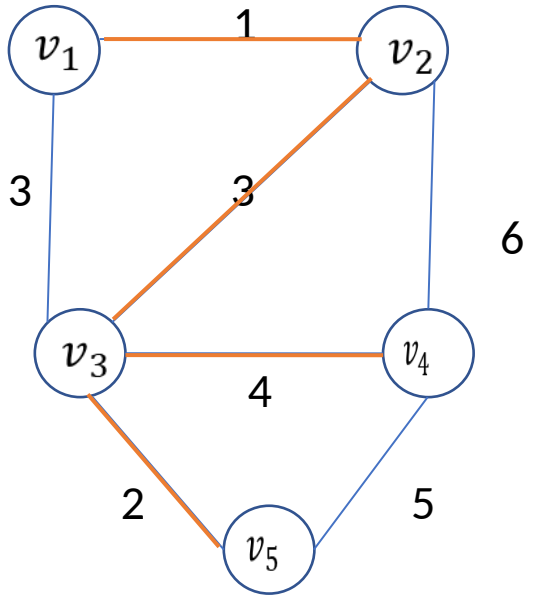
$F = \{v_1v_2, v_2v_3, v_3v_5\} \quad \Rightarrow F = \{v_1v_2, v_2v_3, v_3v_5, v_3v_4\}$

$Y = \{v_1, v_2, v_3\} \quad \Rightarrow Y = \{v_1, v_2, v_3, v_5, v_4\}$

$V - Y = \{v_4\}$

$$\Rightarrow F = \{v_1v_2, v_2v_3, v_3v_5, v_3v_4\}$$

$$\Rightarrow Y = \{v_1, v_2, v_3, v_5, v_4\}$$



$\theta(n^2)$