# Advance Algorithm Analysis (COMP 502 A)

Sharoon Nasim

sharoonnasim@fccollege.edu.pk

**Office Hours:**

**Office: S-426 D**

Pre-requisite: Design and Analysis of Algorithms

**Course Description**:

This course is an introductory graduate-level and advanced undergraduate course on design and analysis of algorithms. Techniques such as divide-and-conquer, dynamic programming, randomized algorithms, and graph algorithms will be taught.

# Course Material

**Textbooks:**

- Introduction to Algorithms by T. Cormen, C. Lieserson et al.
- Algorithms 2006 (DPV) S. Dasgupta, .et al
- Algorithm Design by J. Kleinberg and E. Tardos

**Lectures**

Will be uploaded on Moodle.

## Lectures and Examinations:

- One weekly lecture of 150 minutes duration
- Two in-class midterm examinations
- Comprehensive final examination
- Home works

**Course Assessment (Tentative):**

| | |
|---|---|
| Attendance | 5% |
| Class Activities/ Participation | 5% |
| Assignments/ Homework | 10 % |
| Quizzes | 15 % |
| Midterm – 1 | 15 % |
| Midterm - 2 | 20% |
| Final Exam | 30 % |
| **Total** | **100.00%** |

# Assessment

- What is the difference between dynamic Programming and Divide and Conquer?
- What is Recursion?
- What approach do you use to crack a password?
- Best sorting algorithm and its complexity?
- What is the Edit distance problem and which approach is used to solve this problem?
- Difference between P and NP class?

# Algorithm

An algorithm is a
- sequence of computational steps
- to solve any problem
that transforms input into output.

It should be Correct & Terminate in finite amount of time

# Basic Goal for an Algorithm

- always correct
- always terminates
- This class: performance
  - Performance often draws the line between what is possible and what is impossible.

# Focus of this Course

- Understanding Problem
- Design Algorithm (Different Approaches)

  OR Understand the existing Algorithm
- Analysis of Algorithm (Complexity)

# Approach

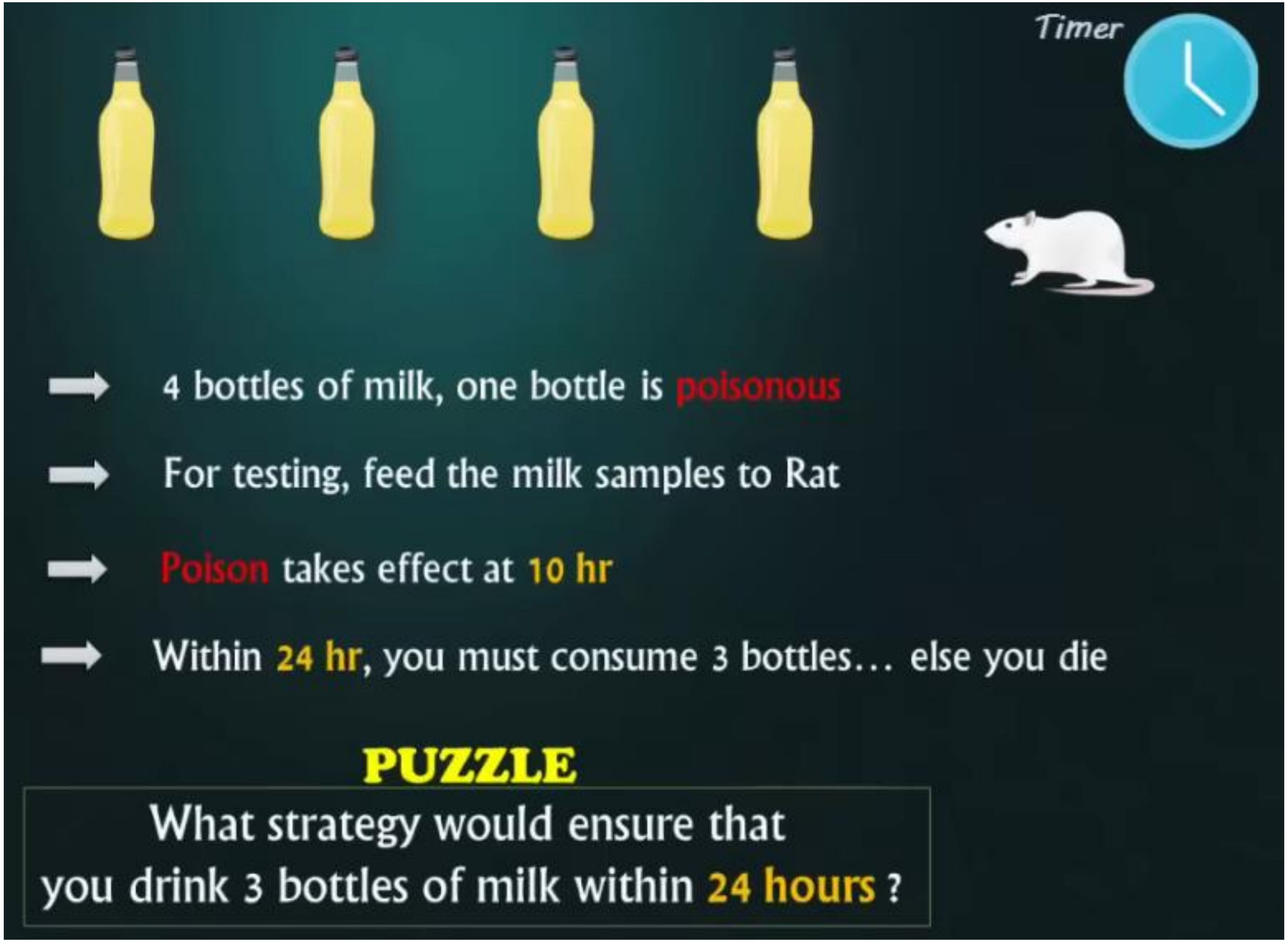
- Brute Force Algorithm
- Divide and Conquer
- Dynamic Programing Algorithm
- Greedy Algorithm
- Backtracking Algorithm

# Class Activity

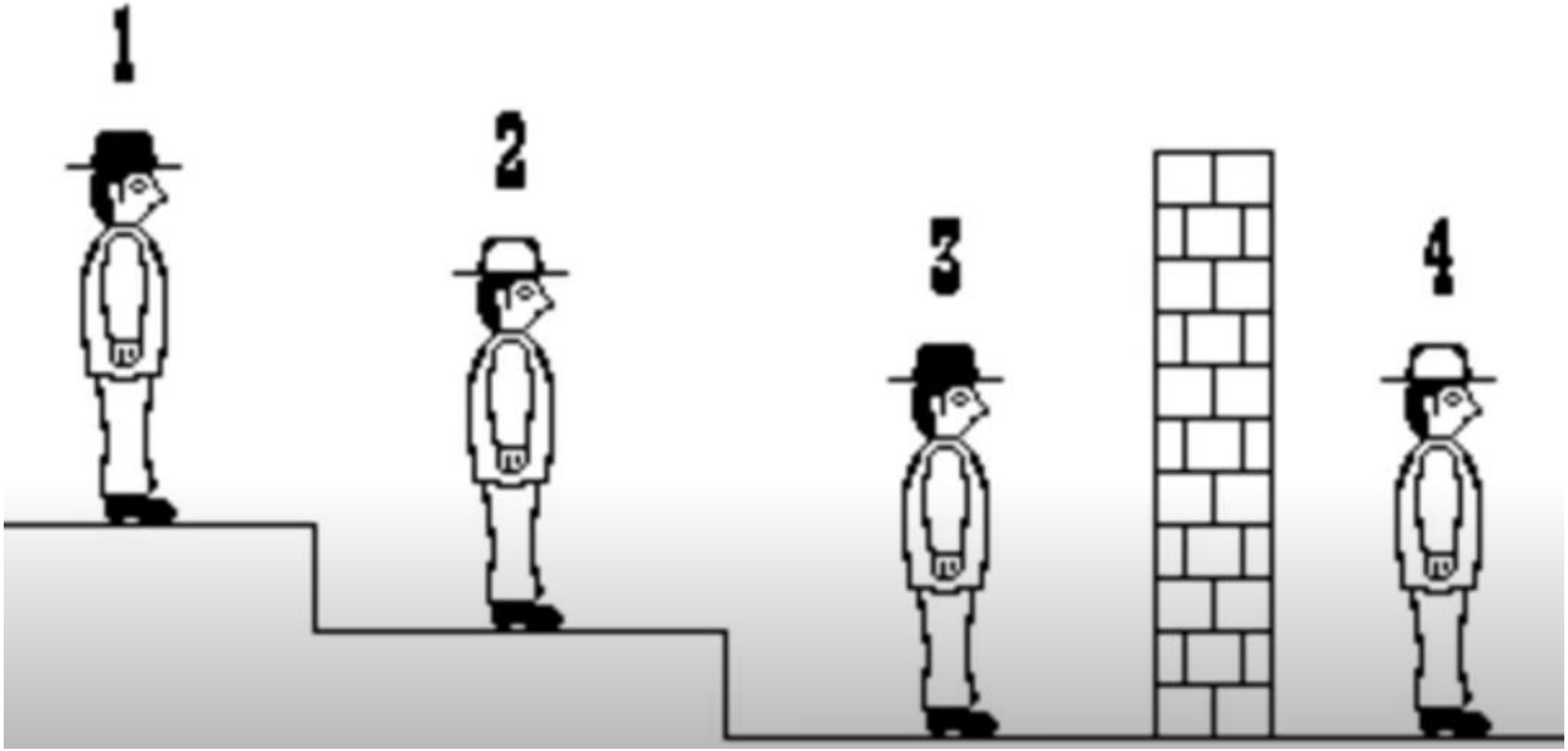
Think of Logic to guess a number between (1 – 100)

At every guess, you will know that your guess is lower or above.

In how many attempts you can guess any number?

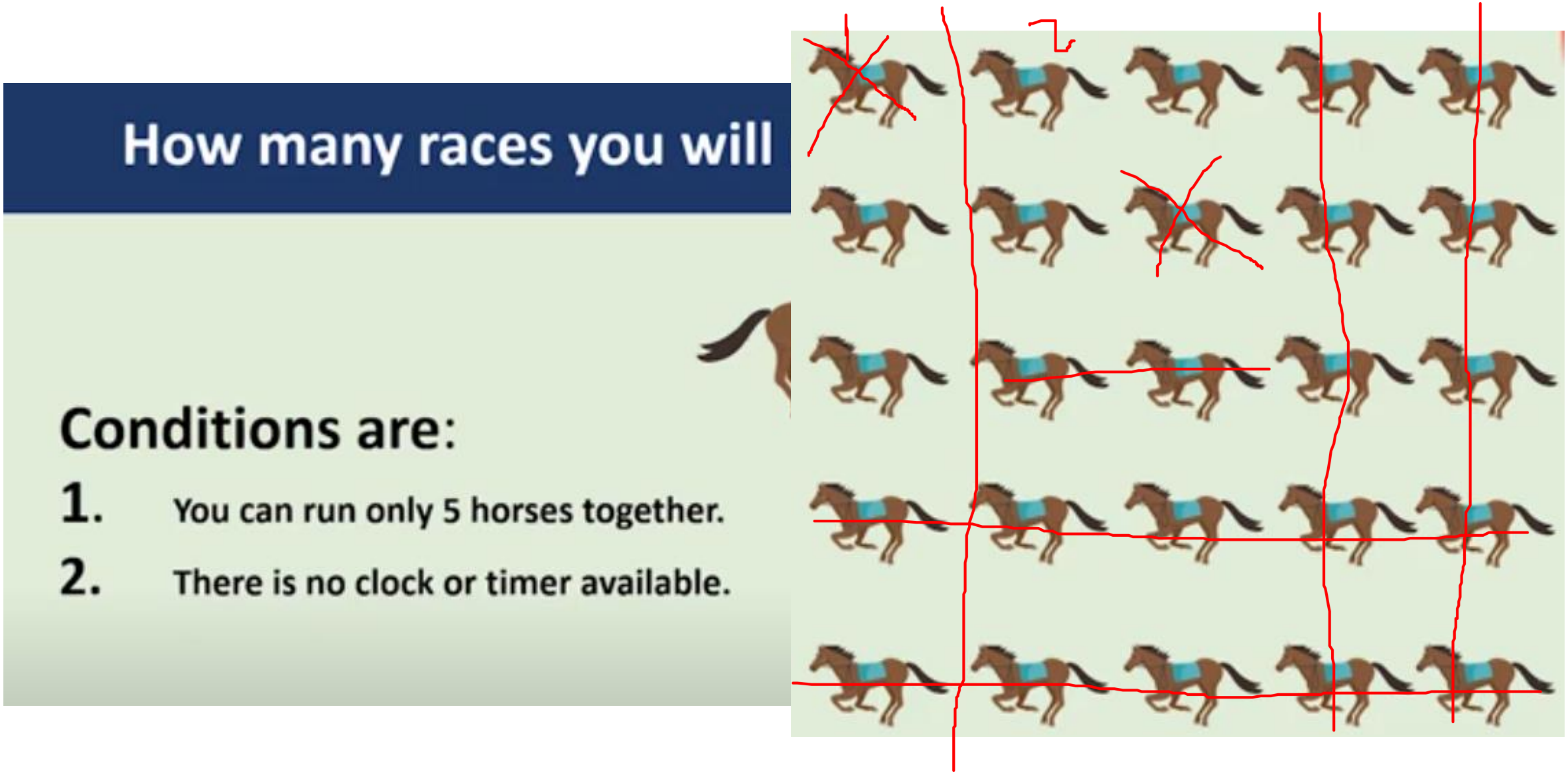

Timer
4 bottles of milk, one bottle is poisonous
For testing, feed the milk samples to Rat
Poison takes effect at 10 hr
Within 24 hr, you must consume 3 bottles… else you die
PUZZLE
What strategy would ensure that
you drink 3 bottles of milk within 24 hours ?

# Who will shout first and tells his hat colour?



# Hat Problem

You have total number 25 horses.



$O\left(n^{\frac{1}{2}}\right)$

O (log n)

$O\ (n^2)$

$O(n)$

$O\left(2^{2^n}\right)$

O(log log n)

O(1)

$O(n^n)$

$O(2^n)$

$O(n^3)$

O(1) < O(log log n) < O (log n)

$$< O\left(n^{\frac{1}{2}}\right) < O(n) < O\ (n^2) < O(n^3)$$

$$< O(2^n) < O\left(2^{2^n}\right) < O(n^n)$$

Fibonacci Series:
0,1,1,2,3,5,8,13,21,34,55,...............

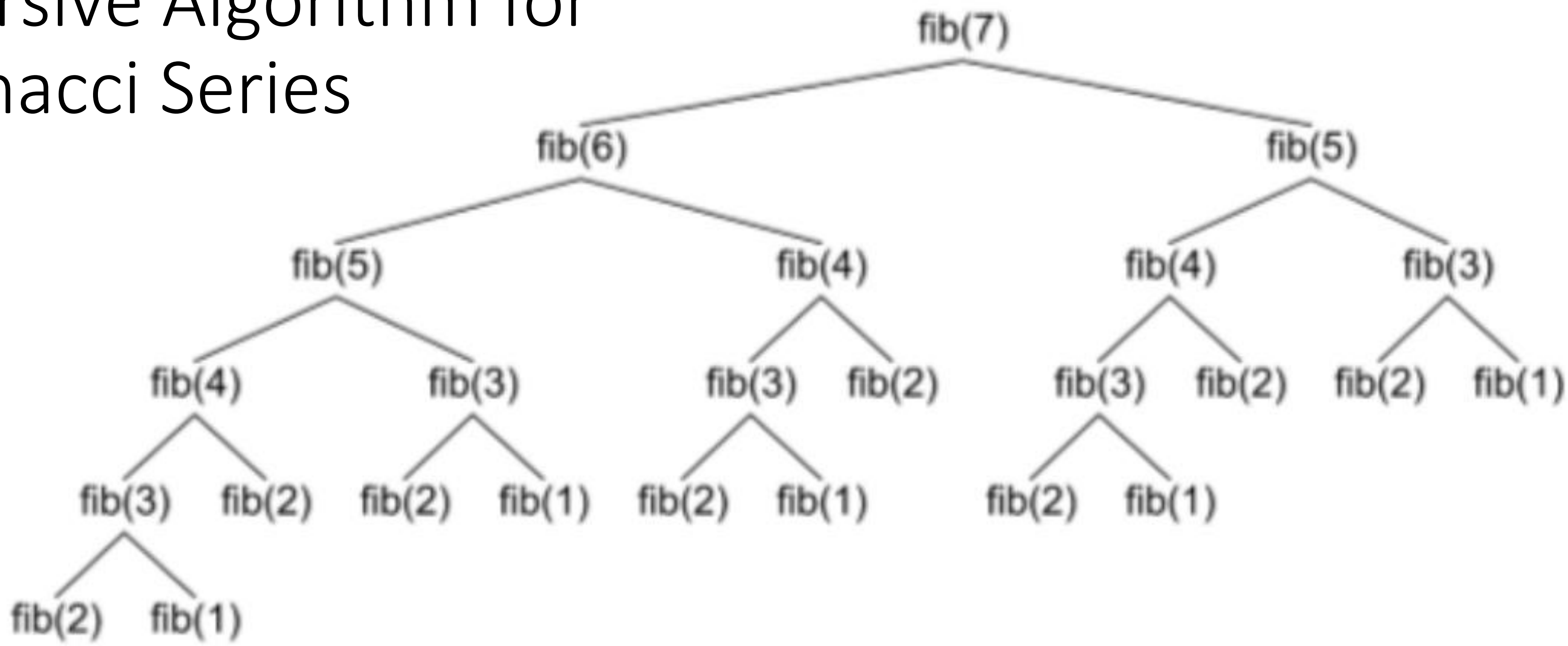
**Each number is the sum of the previous two number**

# Iterative Approach

```
def fib(n):
    a,b = 0,1
    for i in range(n):
        a,b = b,a+b
    return a
```

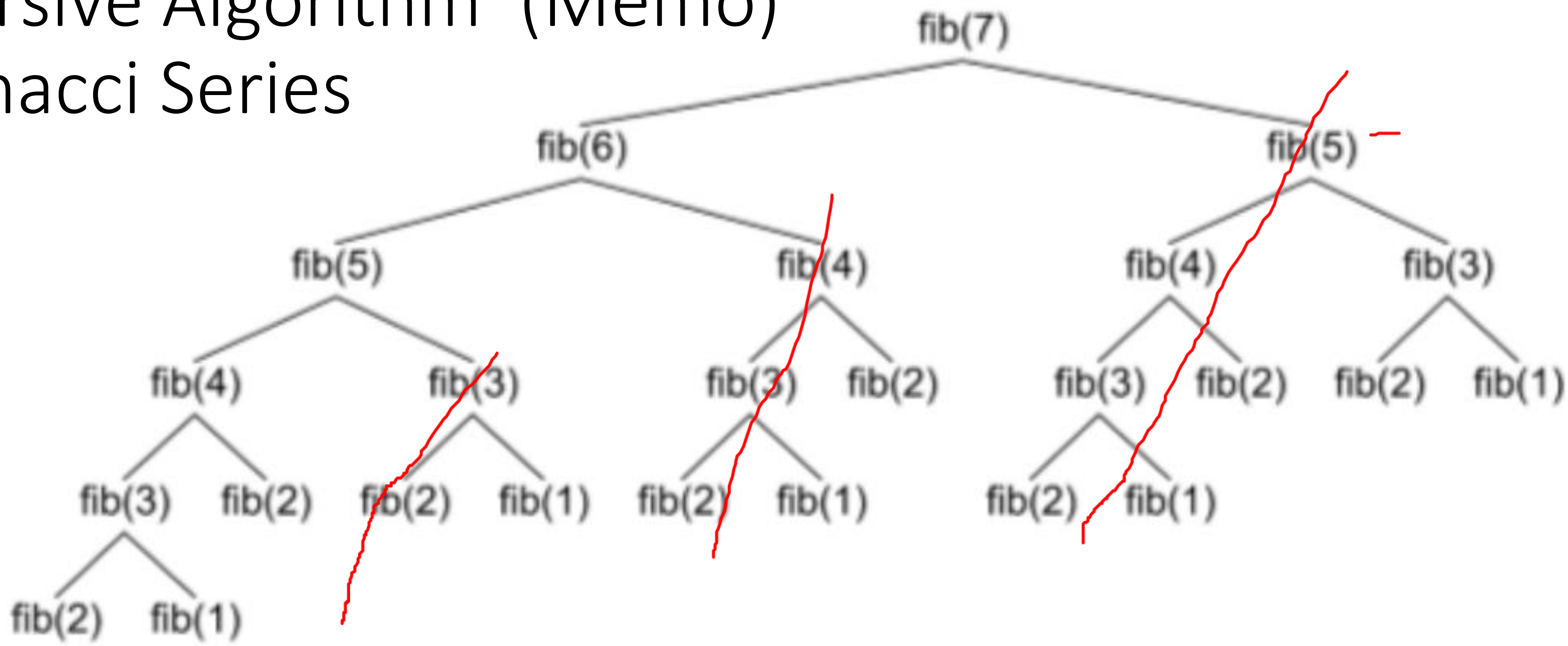
O(n)

# Recursive Algorithm for Fibonacci Series



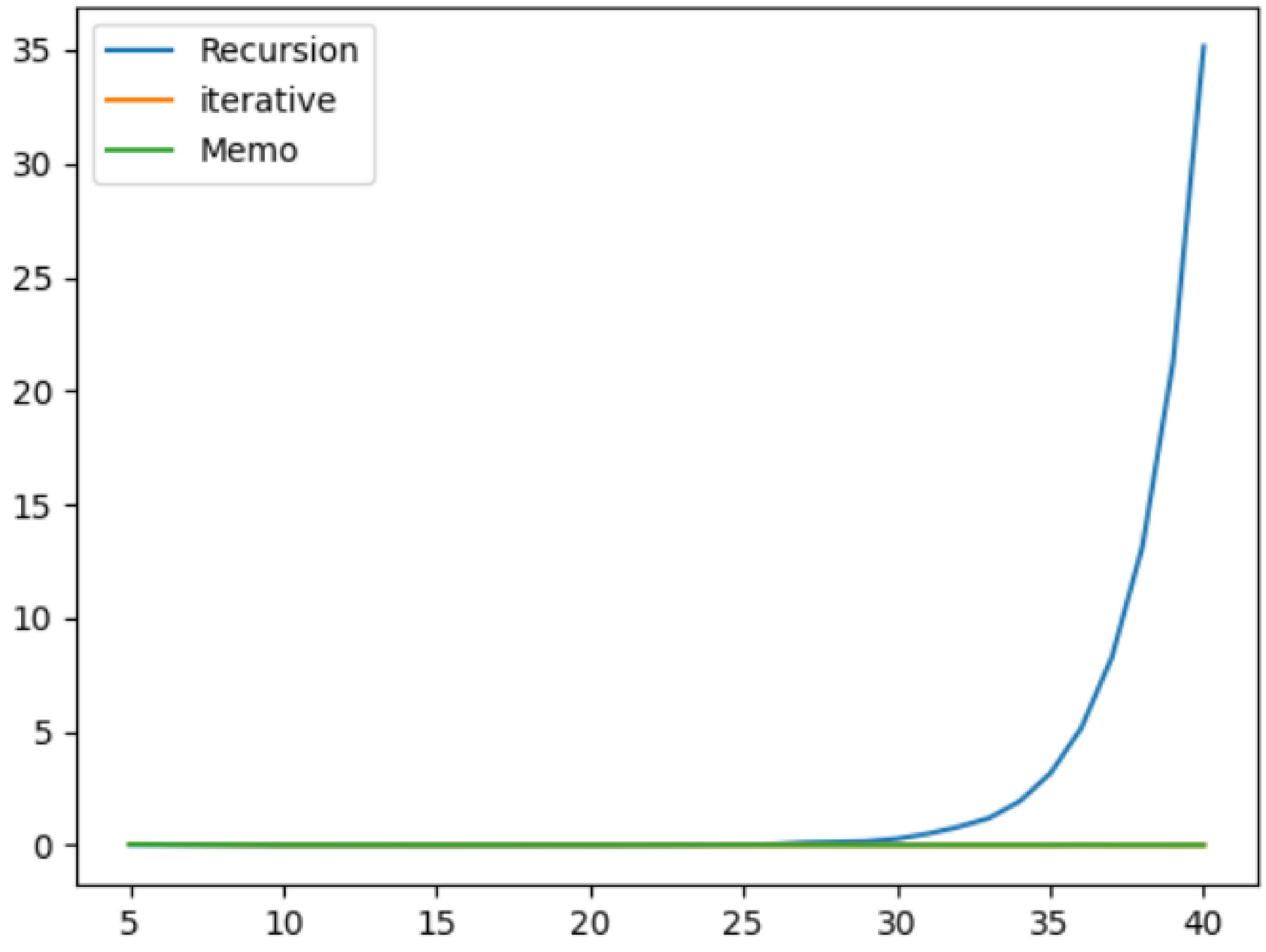

```
def recursive_fib(n):
    if n <= 1:
        return n
    else:
        return(recursive_fib(n-1) + recursive_fib(n-2))
```

# Recursive Algorithm (Memo)
# Fibonacci Series




| 0 | 1 | | | | | | |
|---|---|---|---|---|---|---|---|
| Fib(0) | Fib(1) | Fib(2) | Fib(3) | Fib(4) | Fib(5) | Fib(6) | |

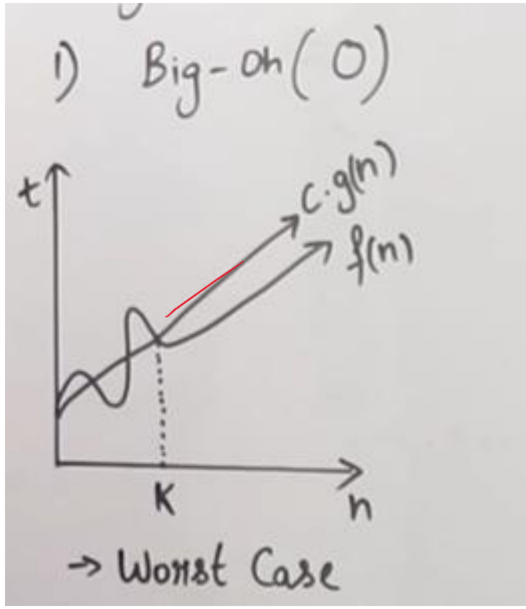
```
def dp_fib(n):
    if n in memo.keys():
        return memo[n]
    else:
        memo[n] = dp_fib(n-1) + dp_fib(n-2)
        return(memo[n])
```

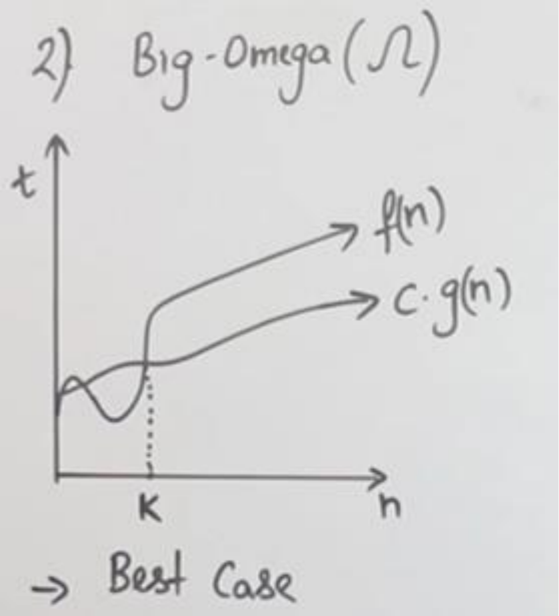

Recursion
iterative
Memo
35
30
25
20
15
10
5
0
5
10
15
20
25
30
35
40

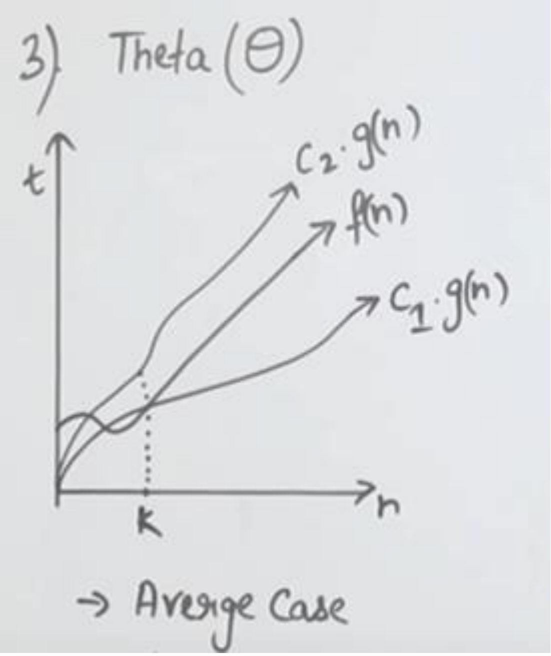
f = Big O(g)
can be thought of as
f <= g

f = Big Omega (g)
can be thought of as
f >= g

f = Big Theta (g)
can be thought of as
f = g



# Posterior and Priori Analysis

**Posteriori analysis** is a relative analysis. It depends on the compiler's language and the hardware type. time algorithm takes to execute on the system.

**Priori analysis** is an absolute analysis. It is independent of the compiler's language and the hardware types. Asymptotic Notations are used to estimate run time instead of running it on the machine.

# Tractability vs Intractability

Tractable Problem: a problem that is solvable by a polynomial-time algorithm. The upper bound is polynomial.

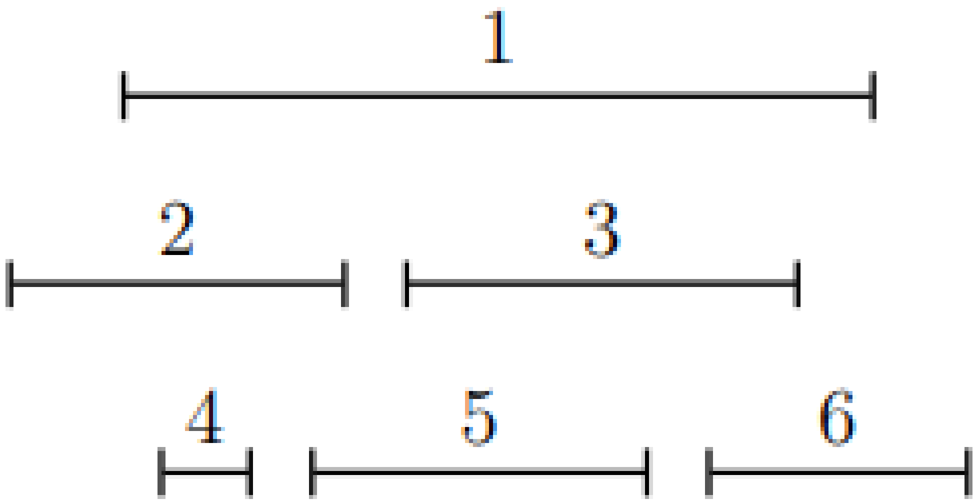
P Class

Intractable Problem: a problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential.

NP Class

| | 10 | 50 | 100 | 300 | 1000 |
|---|---|---|---|---|---|
| $5n$ | 50 | 250 | 500 | 1500 | 5000 |
| $n\times$ $\log n$ | 33 | 282 | 665 | 2469 | 9966 |
| $n^2$ | 100 | 2500 | 10000 | 90000 | 1 million (7 digits) |
| $n^3$ | 1000 | 125000 | 1 million (7 digits) | 27 million (8 digits) | 1 billion (10 digits) |
| $2^n$ | 1024 | a 16-digit number | a 31-digit number | a 91-digit number | a 302-digit number |
| $n!$ | 3.6 million (7 digits) | a 65-digit number | a 161-digit number | a 623-digit number | unimaginably large |
| $n^n$ | 10 billion (11 digits) | an 85-digit number | a 201-digit number | a 744-digit number | unimaginably large |

# Interval Scheduling

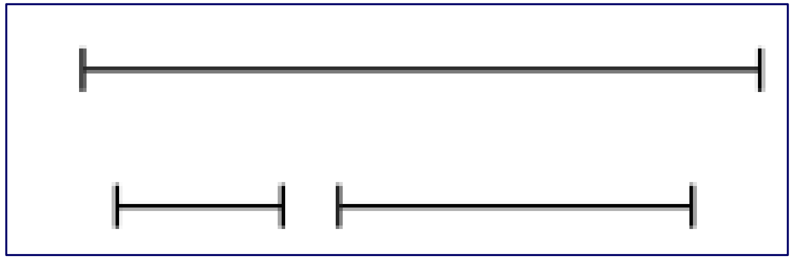



Goal: Select a compatible subset of requests of maximum size.
Claim: We can solve this using a greedy algorithm. A greedy algorithm is a myopic algorithm that processes the input one piece at a time with no apparent look ahead.

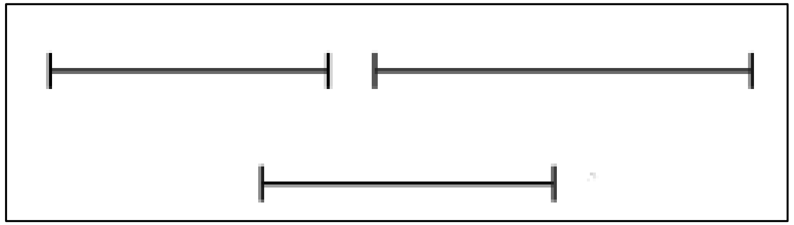
Greedy Interval Scheduling

1. Use a simple rule to select a request i.
2. Reject all requests incompatible with i.
3. Repeat until all requests are processed

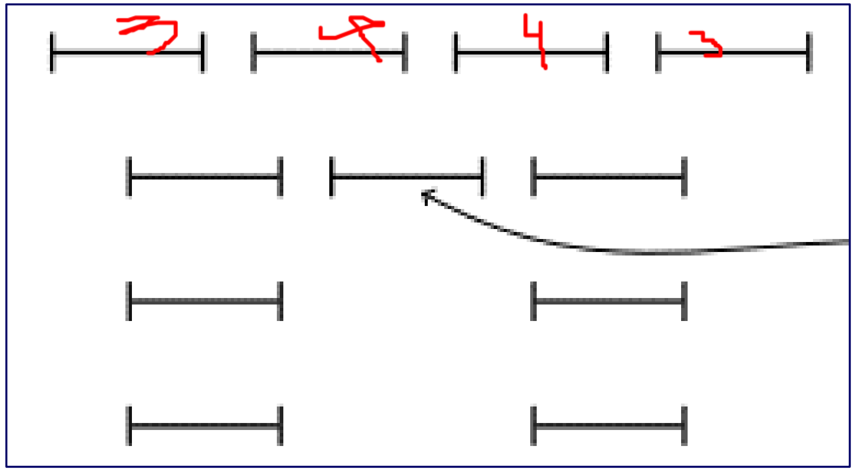
# Possible rules?



1. Select request that starts earliest, i.e., minimum s(i).



2. Select request that is smallest, i.e., minimum f(i) – s(i).



3. For each request, find number of incompatibles, and select request with minimum such number
4. Select request with earliest finish time, i.e., minimum f(i)

**Claim:** Given list of intervals L, greedy algorithm with earliest finish time produces k∗ intervals, where k∗ is optimal.

Proof. Induction on k∗.

Base case: k∗ = 1 : this case is easy, any interval works.

Inductive step:

Suppose claim holds for k∗ and we are given a list of intervals whose optimal schedule has k∗ + 1 intervals, namely

$$S^*[1,2,\ldots,k^*+1] =< s(j_1), f(j_1) >, \ldots, < s(j_{k^*+1}), f(j_{k^*+1}) >$$

Say for some generic k, the greedy algorithm gives a list of intervals

$$S[1,2,\ldots,k] =< s(i_1), f(i_1) >, \ldots, < s(i_k), f(i_k) >$$

By construction, we know that f(i1) ≤ f(j1), since the greedy algorithm picks the earliest finish time.

Now we can create a schedule

$$S^{**} = \underline{< s(i_1), f(i_1) >,} < s(j_2), f(j_2) >, \ldots, < s(j_{k^*+1}), f(j_{k^*+1}) >$$

since the interval < s(i1), f(i1) > does not overlap with the interval < s(j2), f(j2) > and all intervals that come after that. Note that since the length of S∗∗ is k∗ +1, this schedule is also optimal.

# L’ = list of intervals that are compatible

L’ is the set of intervals with $s(i) \geq f(i1)$.

Since $S{\ast\ast}$ is optimal for L,

$S{\ast\ast}[2, 3, \ldots, k{\ast} + 1]$ is optimal for L' , which implies that the optimal schedule for L' has $k{\ast}$ size.

We now see by our initial inductive hypothesis that running the greedy algorithm on L' should produce a schedule of size $k{\ast}$. Hence, by our construction, running the greedy algorithm on L' gives us $S[2, \ldots, k]$.

This means $k - 1 = k{\ast}$ or $k = k{\ast} + 1$,

which implies that $S[1, \ldots, k]$ is indeed optimal, and we are done.

# Weighted Interval Scheduling

Each request i has weight w(i). Schedule subset of requests that are non-overlapping with maximum weight.

Greedy Approach ☹

# Weighted Interval Scheduling – Dynamic Programming

WIS( 1…n)  = max { W1 + WIS(R) , WIS( 2 … n) }

# R > remaining compatible requests

Sort all interval according to start time in ascending order.

| | Intervals | W | Intervals (sorted) | W |
|---|---|---|---|---|
| 1 | 1-3 | 5 | 1-3 | 5 |
| 2 | 2-5 | 6 | 2-5 | 6 |
| 3 | 4-6 | 7 | 4-6 | 7 |
| 4 | 6-7 | 4 | 5-8 | 11 |
| 5 | 5-8 | 11 | 6-7 | 4 |
| 6 | 7-9 | 2 | 7-9 | 2 |

WIS( 1…n) = max { W1 + WIS(R) , WIS( 2 … n) }

19

wis(1–6) = max( 5 + wis(3–6)$^{13}$, wis(2–6)$^{19}$ )

wis(2–6) = max( 6 + wis(4–6)$^{13}$, wis(3–6)$^{13}$ )$^{19}$

2, 4, 6

wis(3–6)$^{13}$ = max( 7 + wis(5–6)$^{6}$, wis(4–6)$^{13}$ )

3, 5, 6

wis(4–6)$^{13}$ = max( 11 + wis(6)$^{2}$, wis(5–6)$^{6}$ )

4, 6

wis(5–6)$^{6}$ = max( 4 + wis(6)$^{2}$, wis(6)$^{2}$ )

wis(6) = 2