# Faculty of Information Technology, Monash University

# FIT2004: Algorithms and Data Structures

## Week 1: Introduction, and Proof of Correctness

These slides are prepared by M. A. Cheema and are based on the material developed by Arun Konagurthu and Lloyd Allison.

# Outline

- Part 1: Introduction to the unit
- Part 2:
  - Proving Correctness of Algorithms
  - Complexity Analysis (Recap)
  - Solving Recurrence Relations

# What's this unit about

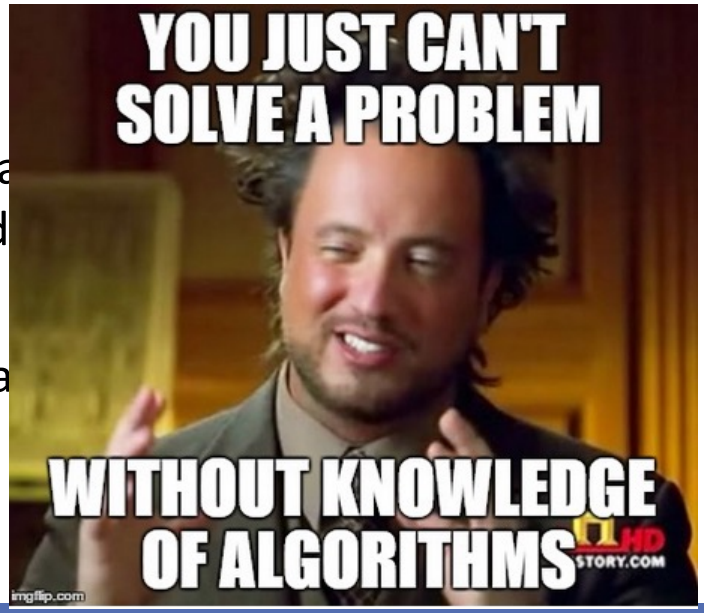
- The subject is about problem-solving with computers: algorithms and data structures.
- The subject is not (mainly) about programming.
- The subject just happens to use Python as the programming language in which lab work (etc.) is done. This subject is really language agnostic.
- Algorithms in this courseware will be presented/describe in English, pseudo-code, procedural set of instructions, Python, or even other programming language(s), as convenient.

# Expectations

You must take this unit seriously and work diligently.

- The subject is arguably the most important for computer and technology related careers
  - Big companies (e.g., Google, Microsoft, Facebook etc.) actively hunt for people good at algorithms and data structures
  - The things you learn will help you throughout your career
  - Expertise in algorithms and data structures is a must if you want to do research in computer science
- This unit is **CHALLENGING**
  - You have to be on top of it from week 1 – you ca
    cover up the material close to the assessment d
  - Missing lectures or tutorials will require double

**Good News:** If you work diligently, you will learn a



# What advice you will give to the students taking this unit in the next semester? I plan to share these with the students in upcoming semesters.

52 responses

**Survey conducted at the end of S1, 2018**

Be ready and attentive from the start because it will be a rewarding experience.

Read the book, helps more than just looking at the lecture slides. Ask Daniel any questions, he helps. Where there's a will there's a way and it isn't through depression and anxiety.

Study consistency each week. Do all the recommended readings. And make sure if you don't understand something you ask for help IMMEDIATELY.

Study the material, most of it is actually quite interesting

If you spend enough time on this unit, it is guaranteed that you will get the benefits and reward from this unit.

ADS is like having sex with your brain. Pure joy.

Start assignments early, keep up with the lectures and you will not struggle.

Don't fall behind! Implement each algorithm mentioned, and it will save you 5 times the amount of time it took you to implement in turn of banging your head against a wall!

Work hard!!

Man don't Procrastinate

## Continued…


Implement
week
make
concepts
Ask
important
understanding
assignment
hard
read
understand
time
work
help
just
Lectures
lecture
Study
assignments
unit
dont
topics
questions
try
Start
early
algorithms
sure
Keep

# Well, this looks challenging!!! Students must have hated the unit, right?

Overall, I am satisfied with this unit

68 re


What i an adapt

- Alm FIT.

I believe the reason is that the students who work hard get to learn a lot of very interesting stuff and find this very rewarding.

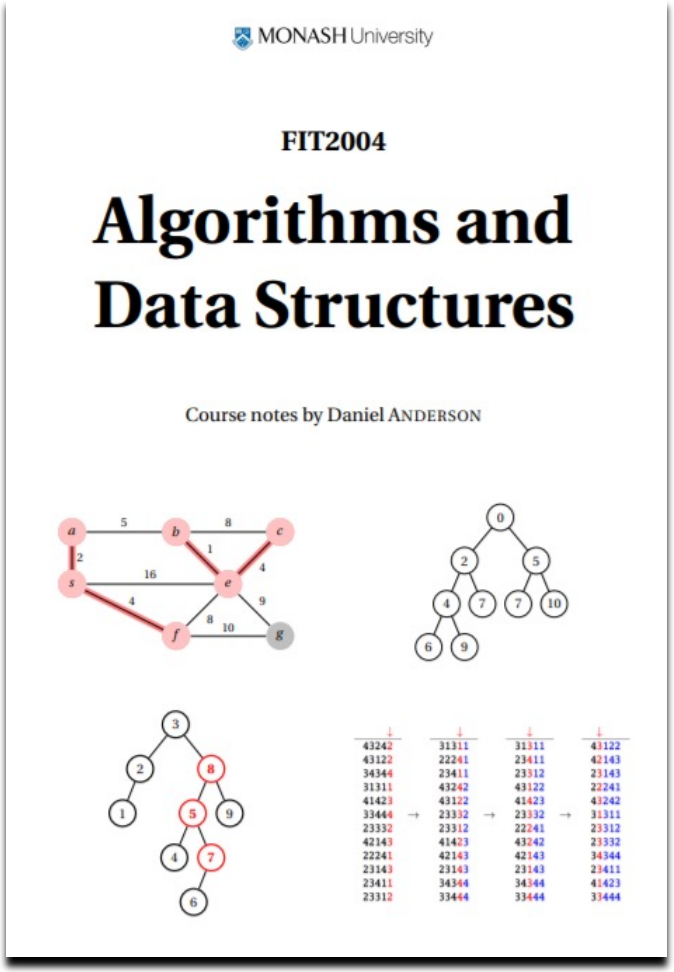
# Overview of the content

The unit aims to cover

- General problem-solving strategies and techniques, e.g.
  - Useful paradigms, e.g.
    - dynamic programming,
    - divide and conquer, etc.
  - Analysis of algorithms and data structures, e.g.
    - program proof / correctness
    - analysis and estimation of space and time complexity, etc.
- A selection of important computational problems, e.g.
  - sorting,
  - retrieval/searching, etc.
- A selection of important algorithms and data structures,
  - as a tool kit for the working programmer,
  - as example solutions to problems
  - as examples of solved problems, to gain insight into concepts

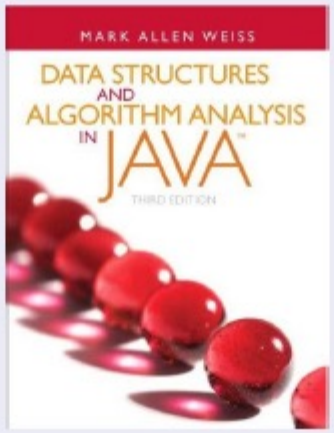
# Unit Notes

- Unit notes are written based on the material covered in lecture notes
- Notes for all 12 weeks are available in a single PDF file on Moodle
  - Click on “Unit Information”
  - Scroll down to the bottom of the page
  - Click on “Lecture Notes – written by Daniel Anderson”



# Textbook reference

Mark Allen Weiss, **Data Structures and Algorithm Analysis in Java** 3rd Edition. Addison-Wesley.




This book (and this holds true no matter what the recommend book on this topic is) covers some material not in the lectures and vice versa. Relevant to the course plan this semester are **Chapters 1, 2, 3, 4** (not splay trees), **Chapter 5** (linear, chaining, quadratic), **Chapter 6** (heap and heap sort), **Chapter 7** (relevant sorts), **Chapter 9** (topological sort, paths, spanning Trees, and **Chapter 8** (where relevant), and **Chapter 10**.

# Recommended Books

- This subject has immense practical value for your development into professional programmers, technicians, software engineers, computer scientists, etc. Therefore, you should read beyond what is prescribed for this unit.

- Additional books you might want to refer to (from time-to-time, even beyond this unit):

  - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliord Stein. *Introduction to Algorithms*. 3rd Edition. The MIT Press & Mc Graw Hill.
  - Donald Knuth, *The Art of Computer Programming*, Pearson. (Pretty expensive, but good-value-for-money for serious programmers; library has copies!)

# FIT2004 Staff

- **Chief examiner and unit lecturer:** A/Prof Reza Haffari
- **Other unit lecturer:** Dr. Davoud Mougouei
- **Malaysia unit lecturer:** Dr. Wern Han Lim
- **Tutors:**
  - Mohammad Shamsur Rahman
  - Nathan Companez
  - Chaluka Salgado
  - Zhou Shao
  - Kiana Zeighami
  - Saman Ahmadi
  - Tharindu Warnakula
  - Vishwajeet Kumar

# Course Material

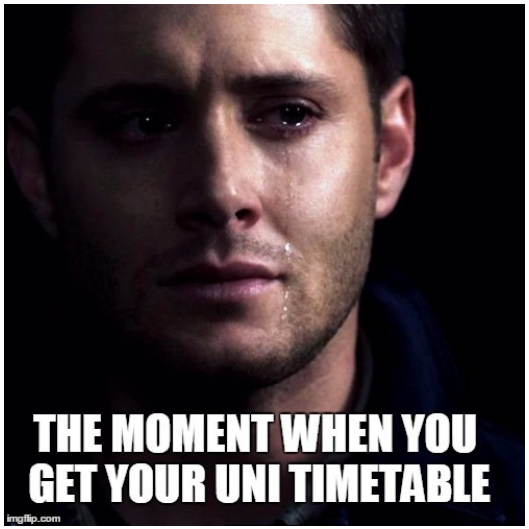
- Your main portal will be, as you already know, the unit's Moodle page: http://moodle.vle.monash.edu/
- Material available on Moodle will include:
  - Lecture slides
  - Lecture recordings
  - Unit Notes
  - Tutorial sheets
- Remember(!) to keep following the forum posts

Read the supplementary materials every week

Start early and read supplementary material for a thorough understanding. Attend consultations

# Course Structure

- Lecture 2hrs (Thursday 14:00 to 16:00, CL_21COL/E1)
  - Live Streaming!
  - All classes start at :00 and finish at :50
- Tutorial 3hrs (see allocate+)
  - No tutorial in week 01




**Bad timetable???** I tried my best to move classes around but timetabling is beyond my control!

# Assessment Summary

- Prac assignments 1-4 (week 4, 6, 10 and 12) 30%
- Tutorial participation 10%
- Final Exam 60%

# Prac Assignments 1-4 (30%)

- Four practical assignments (each worth 7.5%)
  - Due: Week 4, 6, 10, 12
- Focus is on implementing algorithms satisfying certain complexity requirements
- Must be implemented in Python
- Start early!!!

# Attending Lectures via Live Streaming

- Physical attendance is highly recommended
- There may be a delay of up to a few minutes
- DO NOT ASK QUESTIONS USING PANOPTO (not monitored)
- We will access live streaming record to see how much of the lecture you watched via live streaming
  - Don’t just drop-in/drop-out halfway through
- It is your responsibility to make sure you are able to participate via live streaming
  - Poor internet connection, crashed computer etc. are not accepted as valid reasons for special consideration

# Tutorial Participation (10%)

- We have 3 hrs weekly tutorials starting from week 2
  - Tutorial sheet for week 1 is uploaded and you are expected to complete it at your own time
- Each tutorial is worth 1 marks . There are 11 weeks. So that best 10 tutorials will be counted. (total marks capped at 10).
- In each lab/tutorial:
  - Hurdle: (no marks awarded if hurdle not met)
    - You must answer questions in the section (assessed preparation) in your Tutorial sheet before the class starts (give your tutors your solutions at the start of the class)
  - Active participation:
    - Actively engaging in discussing the tutorials in-class
    - 1 marks awarded if hurdle met **and** actively participated

# Final Exam (60%)

- 2 hours + 10 minutes reading time

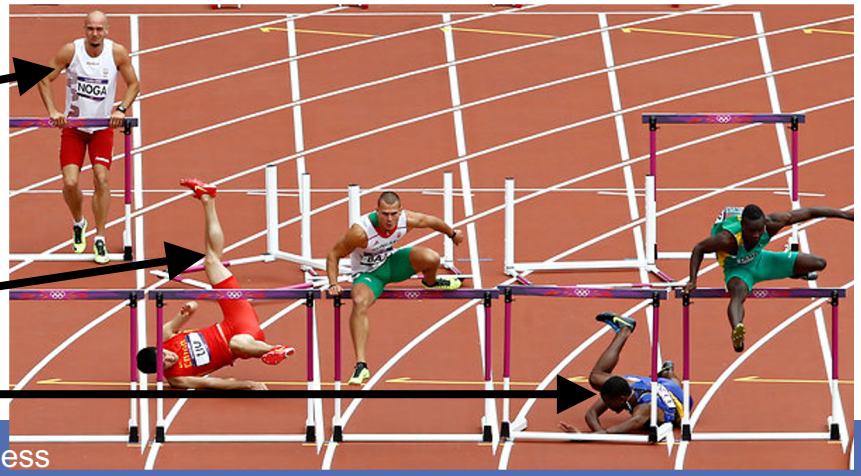
# Hurdles

- To pass this unit a student must obtain:
  - **40% or more in the unit's final exam (i.e., at least 24 out of 60 marks for final exam), and**
  - **40% or more in the unit's total non-examination assessment (i.e., at least 16 out of total 40 marks for in-semester assessments), and**
  - **an overall unit mark of 50% or more**.
- If a student does not pass these hurdles then a mark of no greater than **49N** will be recorded for the unit.

He is the worst of all. Didn't even try.

don't be that guy

(or that one)



# Submission of Assignments

- **Submission details will be specified on each assignment/laboratory sheet**
  - You will submit your assignments to Moodle.
  - Late submission will have 20% off the total assignment marks per day (including weekends).
  - Assignments submitted 5 days after the due date will normally not be accepted.
- **Extensions**
  - Not normally given.
  - Genuine and compelling reasons only.
  - Supporting documentations are needed BEFORE any consideration would be given.

# Cheating, Collusion, Plagiarism

- **Cheating:** Seeking to obtain an unfair advantage in an examination or in other written or practical work required to be submitted or completed for assessment.
- **Collusion:** Unauthorised collaboration on assessable work with another person or persons.
- **Plagiarism:** To take and use another person's ideas and or manner of expressing them and to pass them off as one's own by failing to give appropriate acknowledgement. This includes material from any source, staff, students or the Internet – published and un-published works.

**http://infotech.monash.edu.au/resources/student/assignments/policies.html**

# Cheating, Collusion, Plagiarism

## “Helping” others is NOT OK!

MOSS

| /home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]/ (68%) | | /home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]2/ (73%) | |
|---|---|---|---|
| 4-71 | | 2-66 | |
| 95-111 | | 90-106 | |
| 74-91 | | 69-86 | |
| 115-132 | | 110-127 | |

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

```
>>>> file: LongJump.py
#S[redacted]a

print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return when done.)")
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maxi,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    print("Competitor " each ":" sep="")
    at1 = input("Attempt 1:\n")
    at2 = input("Attempt 2:\n")
    at3 = input("Attempt 3:\n")
    x = (at1+at2+at3).lower()
    if (at1+at2+at3).find("oul") != -1:
        x = (at1+at2+at3).lower()
    d.append(at1)
    d.append(at2)
    d.append(at3)
    if x.find("oul") != -1:
        maxi.append(max(eval(at1),eval(at2),eval(at3)))
```

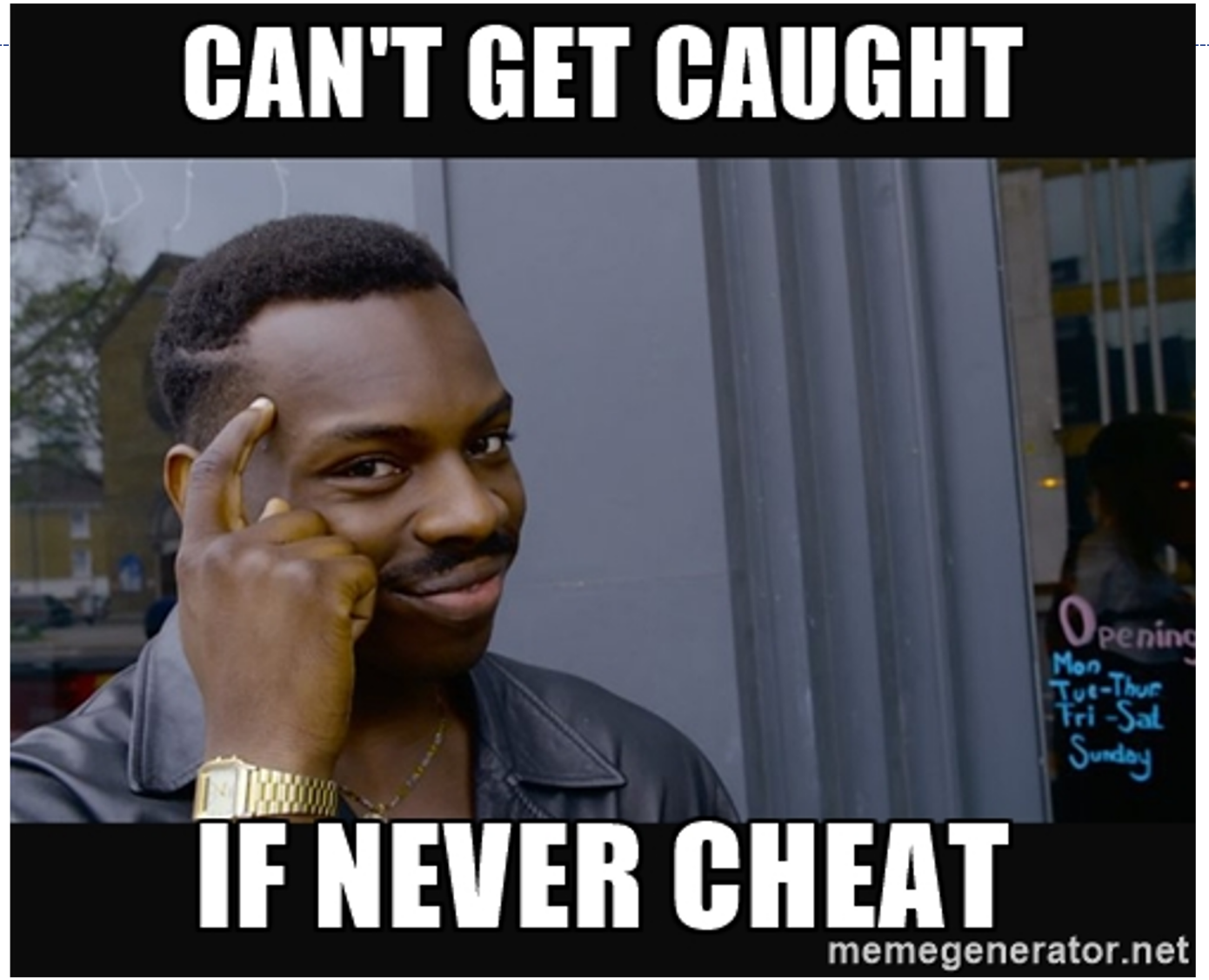
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

```
>>>> file: LongJump.py

print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return when done.)")
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maximums,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    print("Competitor " each ":" sep="")
    attempt1 = input("Attempt 1:\n")
    attempt2 = input("Attempt 2:\n")
    attempt3 = input("Attempt 3:\n")
    g = (attempt1+attempt2+attempt3).lower()
    if (attempt1+attempt2+attempt3).find("oul") != -1:
        g = (attempt1+attempt2+attempt3).lower()
    d.append(attempt1)
    d.append(attempt2)
    d.append(attempt3)
    if g.find("oul") != -1:
        maximums.append(max(eval(attempt1),eval(attempt2),eval(attempt3)))
    else:
        d.remove("foul")
        if not "foul" in d:
```

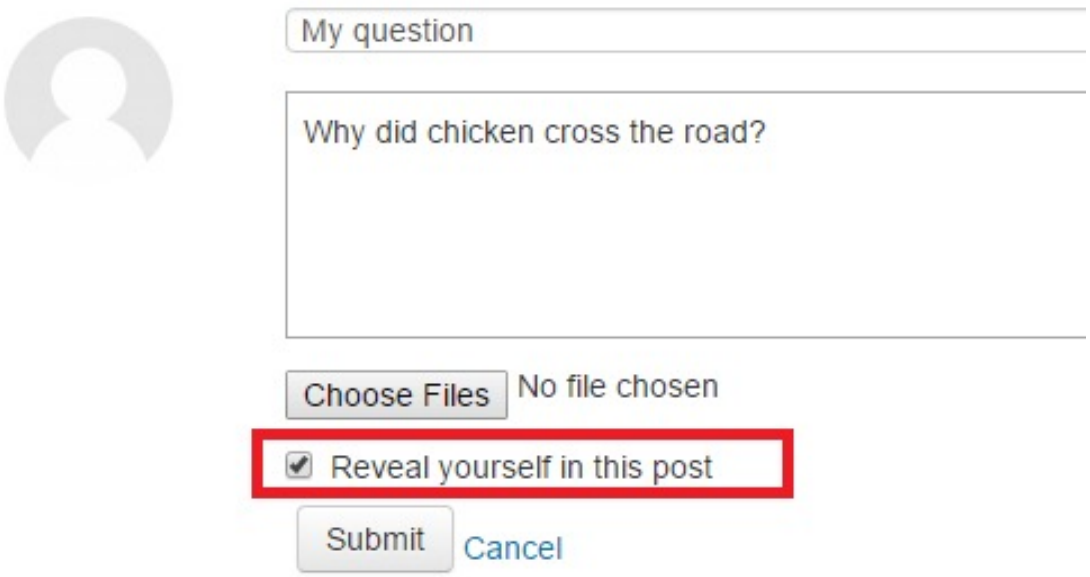


http://lightonphiri.org/wp-content/uploads/2015/09/moss_sample-initial_result-masked-021.png


CAN'T GET CAUGHT
Opening
Mon
Tue-Thur
Fri -Sat
Sunday
IF NEVER CHEAT
memegenerator.net

# Anonymous Moodle Forum

- The identity of a user is not visible to other users (including lectures)
  - Note: In case of inappropriate usage, lecturers can still identify users by looking at logs
- Rules (in addition to the rules for traditional non-anonymous forum)
  - Do not post hints/solutions to the assignments and workshop questions until one week after the due date
- You can reveal your identity if you like



# Short break

# Part 2: Outline

- Proving Correctness of Algorithms
  - Finding Minimum
  - Binary Search
- Complexity Analysis (Recap)
- Solving Recurrence Relations

# Algorithmic Analysis

In algorithmic analysis, one is interested in (at least) two things:

- An algorithm's correctness.
- The amount of resources used by the algorithm

- In this lecture we will see how to prove correctness.
- Next week, we will analyse the resources used by the algorithm aka complexity analysis.

# Consequences of errors

Explosion of unmanned Ariane 5 rocket in 1996

- Exploded within 40 seconds after launch
- Horizontal velocity incorrectly computed
- Loss ~$7 Billion dollars

# Consequences of errors

American Patriot Missile battery in Saudi Arabia failed to intercept an incoming Iraqi Scud Missile

- Killed 28 US soldiers
- Incorrect computation of the time since boot

# Consequences of errors

Incorrect maps almost started a war between Costa Rica and Nicaragua

# Proving correctness of algorithms

- Commonly, we write programs and then test them.
- However, testing can only show that a program is **wrong**.
- It cannot guarantee that it is **always** correct!
  - It may give correct results for 1 Billion test cases but may still be incorrect …
- [Logic], on the other hand, can prove that a program is always correct. This is usually achieved in two parts:
  1. Show that the program **always** terminates, and
  2. Show that a program produces correct results when it terminates

# Part 2: Outline

- Proving Correctness of Algorithms
  - Finding Minimum
  - Binary Search
- Complexity Analysis (Recap)
- Solving Recurrence Relations

# Finding minimum value

```
//Find minimum value in an unsorted array of N>0 elements
min = array[1]//in this unit, we assume index starts at 1
index = 2

while index <= N

        if array[index] < min
                min = array[index]
        index = index + 1

return min
```

# Does it always terminate?

```
//Find minimum value in an unsorted array of N>0 elements
min = array[1]
index = 2

while index <= N

        if array[index] < min
                min = array[index]
        index = index + 1

return min
```

# Correct result at termination?

```
//Find minimum value in an unsorted array of N>0 elements
min = array[1]
index = 2

while index <= N

        if array[index] < min
                min = array[index]
        index = index + 1

return min
```

# Correctness using Loop Invariant

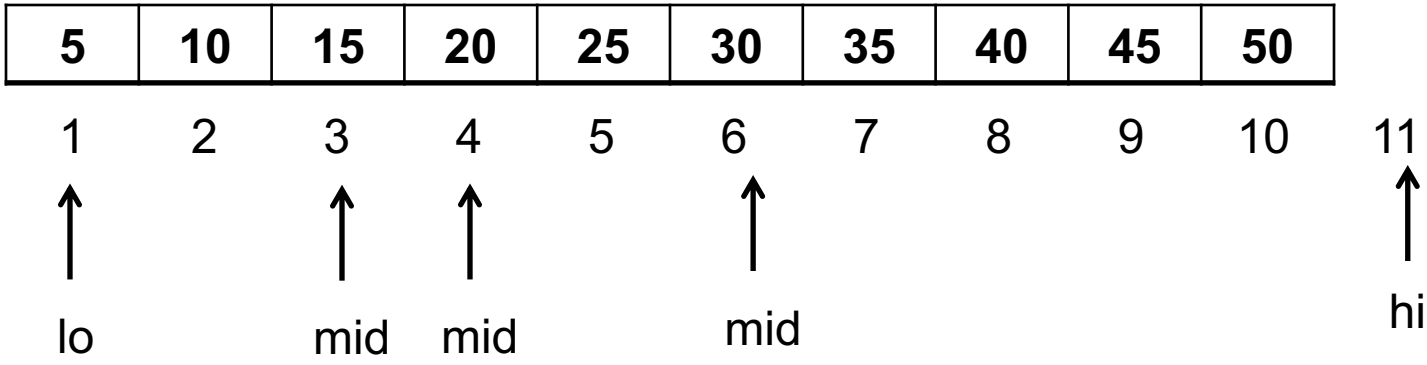
```
//A Loop Invariant (LI) is a property that is true in each
iteration
min = array[1]
index = 2
//LI: min equals the minimum value in array[1 … index - 1]
while index <= N
        //LI: min equals the minimum value in array[1 … index-1]
        if array[index] < min
                min = array[index]
        //min equals the minimum value in array[1 … index]
        index = index + 1

//LI: min equals the minimum value in array[1 … index-1]
// and index = N + 1; thus min is min value in array [1 … N]
return min
```

# Part 2: Outline

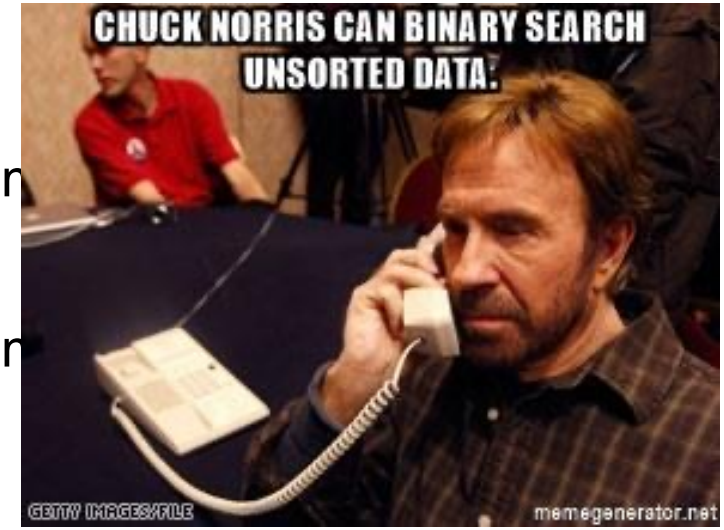
- Proving Correctness of Algorithms
  - Finding Minimum
  - Binary Search
- Complexity Analysis (Recap)
- Solving Recurrence Relations

# Binary Search revisited




Binary search 20 in a **sorted** array

- Since 20 < array[mid],
  - Search from lo to mid (e.g., move hi to m
- Since 20 > array[mid]
  - Search from mid to hi (e.g., move lo to m
- …



# Algorithm for Binary Search

```
lo = 1//we are assuming indices range is 1 to N inclusive
hi = N + 1

while ( lo < hi )
    mid = floor( (lo+hi)/2 )
    if key >= array[mid]
        lo=mid
    else
        hi=mid

if N > 0 and array[lo] == key
    print(key found at index lo)
else
    print(key not found)
```

Is this algorithm correct?

To prove correctness, we need to show that

1. it **always** terminates, and
2. it returns correct result when it terminates

# Does this algorithm always terminate?

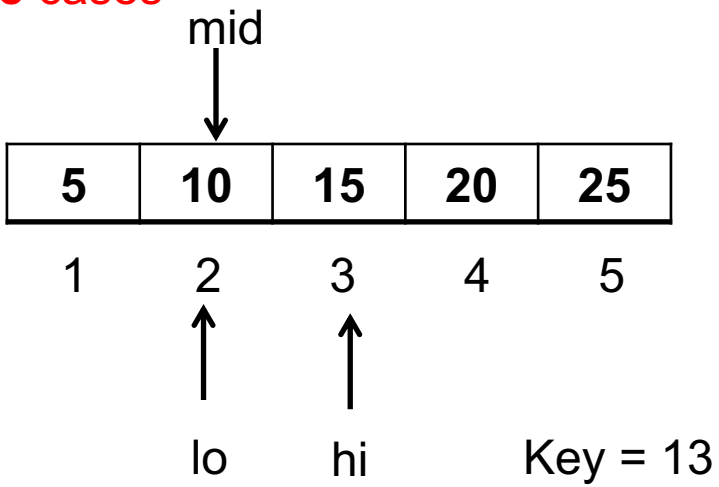
```
lo = 1
hi = N+1

while ( lo < hi )
    mid = floor( (lo+hi)/2 )
    if key >= array[mid]
        lo=mid
    else
        hi=mid

if N>0 and array[lo] == key
    print(key found at index lo)
else
    print(key not found)
```

Let us try to fix this

This algorithm may never terminate in **some** cases



# Does it always terminate?

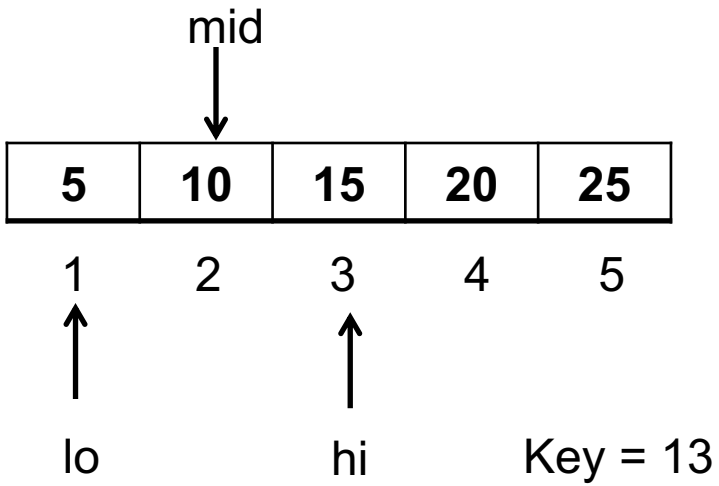
```
lo = 1//we are assuming indices range is 1 to N inclusive
hi = N + 1

while( lo < hi - 1 )
    mid = floor( (lo+hi)/2 )
    if key >= array[mid]
        lo=mid
    else
        hi=mid

if N > 0 and array[lo] == key
    print(key found at index lo)
else
    print(key not found)
```

Proof that it always terminates

- lo < hi – 1implies that the difference between lo and hi is always at least 2
- Therefore, lo < mid < hi.
- Hence, the search space always shrinks (e.g., lo and hi get closer after every iteration of the while loop until lo ≥ hi – 1 in which case the algorithm terminates)



# Correctness using Loop Invariant

```
lo = 1
hi = N + 1
// LI: key in array[1 … N] if and only if (iff) key in array[lo … hi - 1]
while ( lo < hi - 1 )
    // LI: key in array[1 … N] iff key in array[lo … hi-1]
    mid = floor( (lo+hi)/2 )
    if key >= array[mid]
       // key in array[1 … N] iff key in array[mid … hi-1]
       lo=mid
       // LI: key in array[1 … N] iff key in array[lo … hi-1]
    else
       // key in array[1 … N] iff key in array[lo … mid-1]
       hi=mid
       // LI: key in array[1 … N] iff key in array[lo … hi-1]
// LI: key in array[1 … N] iff key in array[lo … hi-1]
```

# Correctness using Loop Invariant

```
// LI: key in array[1 … N] if and only if (iff) key in array[lo … hi - 1]
while ( lo < hi - 1 )
    mid = floor( (lo+hi)/2 )
    if key >= array[mid]
      lo=mid
    else
      hi=mid
// LI: key in array[1 … N] iff key in array[lo … hi-1]
//  lo ≥ hi – 1 → lo + 1 ≥ hi                    ----    (A)
//  lo < hi → lo + 1 ≤ hi                        ----    (B)
// From (A) and (B): lo + 1 = hi → lo = hi - 1
// Hence,  key in array[1 … N] iff key in array[lo … lo];   (Proof Complete)
 if N > 0 and array[lo] == key
       print(key found at index lo)
    else
       print(key not found)
```

Note: lo < hi when loop terminates, because

- lo < mid < hi in each iteration and
- we update either lo to be mid or hi to be mid

# Part 2: Outline

- Proving Correctness of Algorithms
  - Finding Minimum
  - Binary Search
- Complexity Analysis (Recap)
- Solving Recurrence Relations

# Complexity Analysis

- Time complexity
  - The amount of time taken by an algorithm to run as a function of the input size
- Space complexity
  - The amount of space taken by an algorithm to run as a function of the input size
- Worst-case complexity
- Best-case complexity
- Average-case complexity

# Recap from FIT1045: Complexity

How to compute time complexity?

- Count the number of steps taken by the algorithm as a function of input size, e.g., $2N^2 + 10N + 100$
- The big-O notation of this function is its complexity, e.g., $2N^2 + 10N + 100 \rightarrow O(N^2)$

# Recap from FIT1045: Big-O notation

Let N be the number of days

- # of rabbits → $2N^2$
- # of goats → 10N
- # of lions → 100
- #total population → $2N^2$ +10N + 100

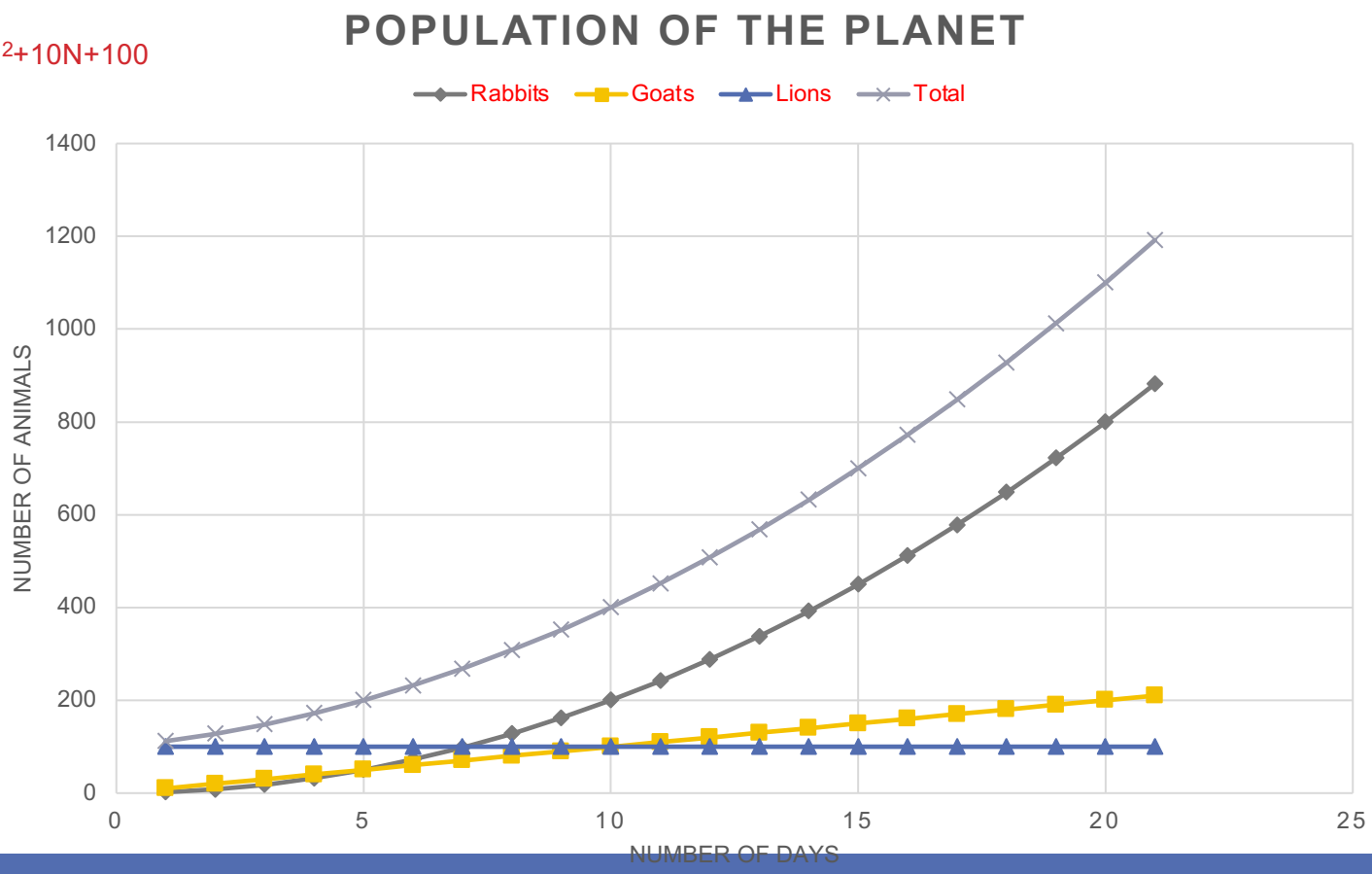
- ✓ The population grows mainly because of growth rate of rabbits
- ✓ For large values of N, # of other animals is insignificant as compared to the # of rabbits
- ✓ We can say population grows in "**O**rder of rabbits' growth"
- ✓ i.e., population growth is $O(N^2)$

| N | rabbits | goats | lions | Total |
|---|---|---|---|---|
| 1 | 2 | 10 | 100 | 112 |
| 2 | 8 | 20 | 100 | 128 |
| 3 | 18 | 30 | 100 | 148 |
| 4 | 32 | 40 | 100 | 172 |
| 5 | 50 | 50 | 100 | 200 |
| 6 | 72 | 60 | 100 | 232 |
| 7 | 98 | 70 | 100 | 268 |
| 8 | 128 | 80 | 100 | 308 |
| 9 | 162 | 90 | 100 | 352 |
| 10 | 200 | 100 | 100 | 400 |
| 11 | 242 | 110 | 100 | 452 |
| 12 | 288 | 120 | 100 | 508 |
| 13 | 338 | 130 | 100 | 568 |
| 14 | 392 | 140 | 100 | 632 |
| 15 | 450 | 150 | 100 | 700 |
| 16 | 512 | 160 | 100 | 772 |
| 17 | 578 | 170 | 100 | 848 |
| 1000 | 2000000 | 10000 | 100 | 2010100 |

# Recap from FIT1045: Big-O notation

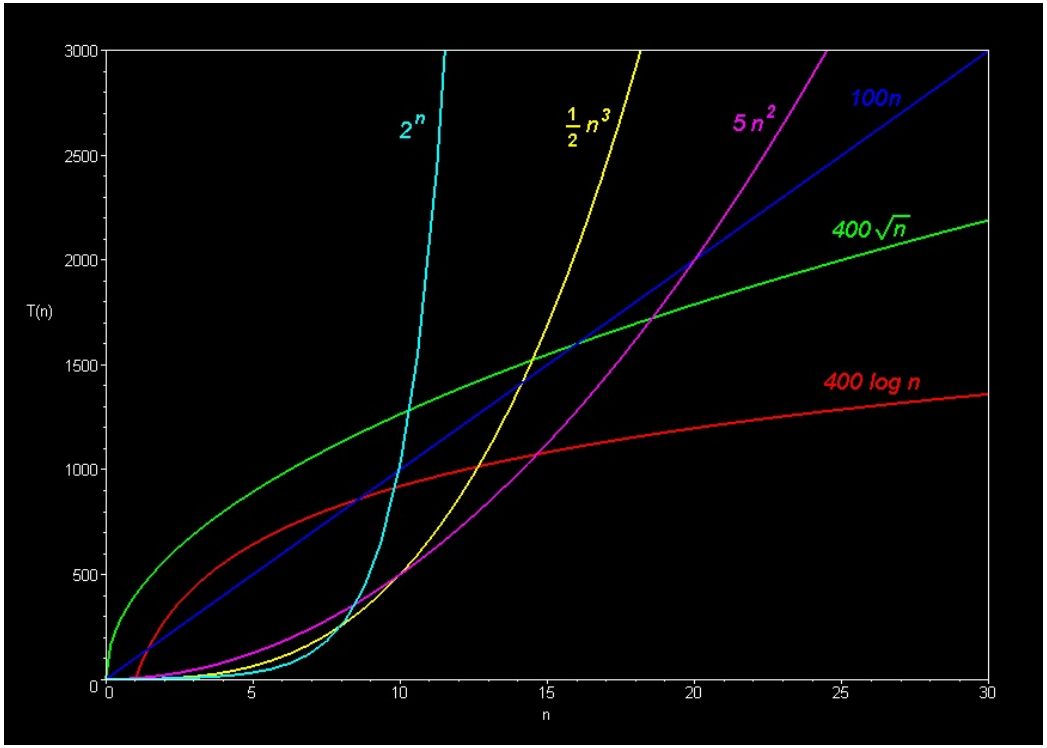
Let N be the number of days

- # of rabbits → $2N^2$
- # of goats → 10N
- # of lions → 100
- #total → $2N^2+10N+100$



# Recap from FIT1045: Big-O notation

- Typically, we use the following simplification rules
  - If function is a product of several terms, any constants that do not depend on N can be ignored
  - If function is a sum of several terms, if there is one with the largest growth rate, it can be kept and others can be omitted
- E.g.,
  - **$12 N^2 + 4 N^3$**
    - **→ $O(N^3)$**
  - **$12 N^2 + 3 N \log(N)$**
    - **→ $O(N^2)$**
  - **$8N^4 + N^2 \log(N) + 12000$**
    - **→ $O(N^4)$**
  - **1000 + 5000**
    - **→ $O(N^0)$→ O(1)**



# What is the complexity of an algorithm in Big-O notation that runs in $30N \log (N^2) + 10 \log N + 8N$?

A. $O(N \log N)$

B. $O(N \log (N^2))$

C. $O(N \log (N^2) + N + \log N)$

D. Option D because

# Part 2: Outline

- Proving Correctness of Algorithms
  - Finding Minimum
  - Binary Search
- Complexity Analysis (Recap)
- Solving Recurrence Relations

# Recurrence Relations

A recurrence relation is an equation that recursively defines a sequence of values, and one or more initial terms are given.

E.g.,

$$T(1) = b$$

$$T(N) = T(N-1) + c$$

- Complexity of recursive algorithms can be analysed by writing its recurrence relation and then solving it

# Solving Recurrence Relations

```
// Compute Nth power of x
power(x,N)
{
    if (N==0)
        return 1
    if (N==1)
        return x
    else
        return x * power(x, N–1)
}
```

Our goal is to reduce this term to T(1)

Time Complexity

Cost when N = 1: T(1) = b (b&c are constant)

Cost for general case: T(N) = T(N-1) + c (A)

Cost for N-1: T(N-1) = T(N-2) + c

Replacing T(N-1) in (A)

T(N) = (T(N-2) + c) + c = T(N-2) + 2*c (B)

Cost for N-2: T(N-2) = T(N-3) + c

Replacing T(N-2) in (B)

T(N) = T(N-3) + c+c+c = T(N-3) + 3*c

Do you see the pattern?

T(N) = T(N-k) + k*c

Find the value of k such that N-k = 1 → k = N-1

T(N) = T(N-(N-1)) + (N-1)*c = T(1) + (N-1)*c

T(N) = b + (N-1)*c = c*N + b - c

Hence, the complexity is O(N)

# Solving Recurrence Relations

```
// Compute Nth  power x
power2(x,N)
{
  if (N==0)
    return 1
  if (N==1)
    return x
  if (N is even)
   return power2( x * x, N/2)
 else
   return power2( x * x, N/2 ) * x
}
```

Time Complexity

Cost when N = 1: T(1) = b (b&c are constant)

Cost for general case: T(N) = T(N/2) + c (A)

Cost for N/2: T(N/2) = T(N/4) + c

Replacing T(N/2) in (A)

T(N) = T(N/4) + c + c = T(N/4) + 2*c (B)

Cost for N/4: T(N/4) = T(N/8) + c

Replacing T(N/4) in (B)

T(N) = T(N/8) + c+c+c = T(N/8) + 3*c

Do you see the pattern?

$T(N) = T(N/2^k) + k*c$

Find the value of k such that $N/2^k = 1 \rightarrow k = \log_2 N$

$T(N) = T(N/2^{\log N}) + c*\log N = T(1) + c* \log_2 N$

$T(N) = b + c* \log_2 N$

Hence, the complexity is O(log N)

# Logarithmic complexity

Recurrence relation:

$$T(N) = T(N/2) + c$$

$$T(1) = b$$

Solution:

$$O(\log N)$$

# Linear Complexity

Recurrence relation:

$$T(N) = T(N-1) + c$$

$$T(1) = b$$

Solution:

$$O(N)$$

# Linearithmic complexity

Recurrence relation:

$$T(N) = 2*T(N/2) + c*N$$

$$T(1) = b$$

Solution:

$$O(N \log N)$$

# Quadratic complexity

Recurrence relation:

$$T(N) = T(N-1) + c*N$$

$$T(1) = b$$

Solution:

$$O(N^2)$$

# Exponential complexity

Recurrence relation:

$T(N) = 2*T(N-1) + c$

$T(0) = b$

Solution:

$O(2^N)$

# Revision: Proof by induction

## 2 steps

1. Prove the base case, e.g., for the first state
2. Assume the proof holds for a state k. Show that it also holds for the next state k+1.

## Theorem

Let S(N) be the sum of the integers from 1 to N

Prove that S(N) = N(N+1)/2

Step 1: Base Case: N = 1

S(1) = 1 and N(N+1)/2 is 1 for N=1

Step 2: Inductive step

Assume it holds for a positive integer k

S(k) = k(k+1)/2

Show that S(k+1) = (k+1)(k+2)/2

continued on next slide…

# Revision: Proof by induction

## Theorem

Let S(N) be the sum of the integers from 1 to N

Prove that S(N) = N(N+1)/2

### Step 2: Inductive step

Assume it holds for a positive integer k

S(k) = k(k+1)/2

Show that S(k+1) = (k+1)(k+2)/2

S(k+1) = 1 + 2 + 3 + … + k+1

S(k+1) = 1 + 2 + 3 + … + k + k+1

S(k+1) = k(k+1)/2 + k+1

S(k+1) = (k+1)(k+2)/2

For more details, watch it on Khan Academy (click here)

# Concluding Remarks

**Summary**

- This unit demands your efforts from week 1
- Testing cannot guarantee correctness; Requires logical reasoning to formally prove correctness

**Coming Up Next**

- Analysis of algorithms
- Non-comparison based sorting (Counting Sort, Radix Sort) – related to Assignment 1

**IMPORTANT: Preparation required before the next lecture**

- Revise computational complexity covered in earlier units (FIT1045, FIT1008). You may also want to watch videos or read other online resources
- Complete Tutorial 1 (at your own time)
- Revise abstract data types, loop invariants, binary search, insertion sort, and selection sort