

Week 10: Randomised Algorithms, Algorithm and Data Ethics, Course Review

Randomised Algorithms

1/76

Algorithms employ randomness to

- improve worst-case runtime
 - compute correct solutions to hard problems more efficiently but with low probability of failure
 - compute approximate solutions to hard problems
-

Randomness

2/76

Randomness is also useful

- in computer games:
 - may want aliens to move in a random pattern
 - the layout of a dungeon may be randomly generated
 - may want to introduce unpredictability
 - in physics/applied maths:
 - carry out simulations to determine behaviour
 - e.g. models of molecules are often assume to move randomly
 - in testing:
 - *stress test* components by bombarding them with random data
 - random data is often seen as *unbiased data*
 - gives average performance (e.g. in sorting algorithms)
 - in cryptography
-

Sidetrack: Random Numbers

3/76

How can a computer pick a number at random?

- it cannot

Software can only produce *pseudo random numbers*.

- a pseudo random number is one that is predictable
 - (although it may appear unpredictable)

⇒ Implementation may deviate from expected theoretical behaviour

... Sidetrack: Random Numbers

4/76

The most widely-used technique is called the *Linear Congruential Generator (LCG)*

- it uses a **recurrence** relation:
 - $X_{n+1} = (a \cdot X_n + c) \bmod m$, where:
 - m is the "modulus"
 - a , $0 < a < m$ is the "multiplier"
 - c , $0 \leq c \leq m$ is the "increment"
 - X_0 is the "seed"

- if $c=0$ it is called a *multiplicative congruential generator*

LCG is not good for applications that need extremely high-quality random numbers

- the period length is too short
- *period length* ... length of sequence at which point it repeats itself
- a short period means the numbers are correlated

... Sidetrack: Random Numbers

5/76

Trivial example:

- for simplicity assume $c=0$
- so the formula is $X_{n+1} = a \cdot X_n \bmod m$
- try $a=11=X_0$, $m=31$, which generates the sequence:

11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10, 17, 1, 11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10, 17, 1, 11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10, 17, 1, 11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10, 17, 1, 11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10, 17, 1, ...

- all the integers from 1 to 30 are here
- period length = 30

... Sidetrack: Random Numbers

6/76

Another trivial example:

- again let $c=0$
- try $a=12=X_0$ and $m=30$
 - that is, $X_{n+1} = 12 \cdot X_n \bmod 30$
 - which generates the sequence:

12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6, ...

- notice the period length (= 4) ... clearly a terrible sequence

... Sidetrack: Random Numbers

7/76

It is a complex task to pick good numbers. A bit of history:

Lewis, Goodman and Miller (1969) suggested

- $X_{n+1} = 7^5 \cdot X_n \bmod (2^{31}-1)$
- note:
 - 7^5 is 16807
 - $2^{31}-1$ is 2147483674
 - $X_0 = 0$ is not a good seed value

Most compilers use LCG-based algorithms that are slightly more involved; see www.mscs.dal.ca/~selinger/random/ for details (including a short C program that produces the exact same pseudo-random numbers as `gcc` for any given seed value)

... Sidetrack: Random Numbers

8/76

- Two functions are required:

```
srand(unsigned int seed) // sets its argument as the seed

rand() // uses a LCG technique to generate random
       // numbers in the range 0 .. RAND_MAX
```

where the constant `RAND_MAX` is defined in `stdlib.h`
(depends on the computer: on the CSE network, `RAND_MAX` = 2147483647)

- The period length of this random number generator is very large
approximately $16 \cdot ((2^{31}) - 1)$

... Sidetrack: Random Numbers

9/76

To convert the return value of `rand()` to a number between 0 .. RANGE

- compute the remainder after division by `RANGE+1`

Using the remainder to compute a random number is not the best way:

- can generate a 'better' random number by using a more complex division
- but good enough for most purposes

Some applications require more sophisticated, *cryptographically secure* pseudo random numbers

Exercise #1: Random Numbers

10/76

Write a program to simulate 10,000 rounds of Two-up.

- Assume a \$10 bet at each round
- Compute the overall outcome and average per round

```
#include <stdlib.h>
#include <stdio.h>

#define RUNS 10000
#define BET 10

int main(void) {
    srand(1234567); // choose arbitrary seed
    int coin1, coin2, n, sum = 0;
    for (n = 0; n < RUNS; n++) {
        do {
            coin1 = rand() % 2;
            coin2 = rand() % 2;
        } while (coin1 != coin2);
        if (coin1==1 && coin2==1)
            sum += BET;
        else
            sum -= BET;
    }
}
```

```

}
printf("Final result: %d\n", sum);
printf("Average outcome: %f\n", (float) sum / RUNS);
return 0;
}

```

... Sidetrack: Random Numbers

12/76

Seeding

There is one significant problem:

- every time you run a program with the same seed, you get exactly the same sequence of 'random' numbers (why?)

To vary the output, can give the random seeder a starting point that varies with time

- an example of such a starting point is the current time, `time(NULL)`
(NB: this is different from the UNIX command `time`, used to measure program running time)

```

#include <time.h>
time(NULL) // returns the time as the number of seconds
           // since the Epoch, 1970-01-01 00:00:00 +0000

// time(NULL) on July 31st, 2020, 12:59pm was 1596164340
// time(NULL) about a minute later was 1596164401

```

Randomised Algorithms

Analysis of Randomised Algorithms

14/76

Randomised algorithm to find *some* element with key k in an unordered array:

```

findKey(L,k):
|   Input   array L, key k
|   Output some element in L with key k
|
|   repeat
|       randomly select  $e \in L$ 
|   until key(e)=k
|   return e

```

... Analysis of Randomised Algorithms

15/76

Analysis:

- p ... ratio of elements in L with key k (e.g. $p = \frac{1}{3}$)
- Probability of success: 1 (if $p > 0$)
- Expected runtime: $\frac{1}{p}$ ($= \lim_{n \rightarrow \infty} \sum_{i=1..n} i \cdot (1-p)^{i-1} \cdot p$)

◦ Example: a third of the elements have key $k \Rightarrow$ expected number of iterations = 3

... Analysis of Randomised Algorithms

16/76

If we cannot guarantee that the array contains any elements with key k ...

```

findKey(L,k,d):
    Input   array L, key k, maximum #attempts d
    Output some element in L with key k

    repeat
        if d=0 then
            return failure
        end if
        randomly select  $e \in L$ 
        d=d-1
    until key(e)=k
    return e

```

... Analysis of Randomised Algorithms

17/76

Analysis:

- p ... ratio of elements in L with key k
- d ... maximum number of attempts
- *Probability of success:* $1 - (1-p)^d$
- *Expected runtime:* $\left(\sum_{i=1..d} i \cdot (1-p)^{i-1} \cdot p \right) + d \cdot (1-p)^{d-1}$
 - $O(1)$ if d is a constant

Randomised Quicksort

18/76

Quicksort applies divide and conquer to sorting:

- **Divide**
 - pick a *pivot* element
 - move all elements smaller than the *pivot* to its left
 - move all elements greater than the *pivot* to its right
- **Conquer**
 - sort the elements on the left
 - sort the elements on the right

Non-randomised Quicksort

19/76

Divide ...

```

partition(array,low,high):
    Input   array, index range low..high
    Output selects array[low] as pivot element
             moves all smaller elements between low+1..high to its left
             moves all larger elements between low+1..high to its right
             returns new position of pivot element

    pivot_item=array[low], left=low+1, right=high

```

```

while left<right do
    left = find index of leftmost element > pivot_item    // left=high if none
    right = find index of rightmost element <= pivot_item
    if left<right then
        swap array[left] with array[right]
    end if
end while
array[low]=array[right] // right is final position for pivot
array[right]=pivot_item
return right

```

... Non-randomised Quicksort

20/76

... and Conquer!

```

Quicksort(array, low, high):
    Input  array, index range low..high
    Output array[low..high] sorted

    if high > low then    // termination condition low >= high
        pivot = partition(array, low, high)
        Quicksort(array, low, pivot-1)
        Quicksort(array, pivot+1, high)
    end if

```

... Non-randomised Quicksort

21/76

3 6 5 2 4 1 // swap a[left=1] and a[right=5]

3 1 5 2 4 6 // swap a[left=2] and a[right=3]

3 1 2 5 4 6 // swap pivot and a[right=2]

2 1 | 3 | 5 4 6

1 2 | 3 | 5 4 6

1 2 | 3 | 4 | 5 | 6

Worst-case Running Time

22/76

Worst case for Quicksort occurs when the pivot is the unique minimum or maximum element:

- One of the intervals $\text{low} \dots \text{pivot}-1$ and $\text{pivot}+1 \dots \text{high}$ is of size $n-1$ and the other is of size 0
 \Rightarrow running time is proportional to $n + n-1 + \dots + 2 + 1$
- Hence the worst case for non-randomised Quicksort is $O(n^2)$

1 2 3 4 5 6

1 | 2 3 4 5 6

1 | 2 | 3 4 5 6

...

1 | 2 | 3 | 4 | 5 | 6

Randomised Quicksort

23/76

```
partition(array, low, high):
    Input   array, index range low..high
    Output randomly select a pivot element from array[low..high]
             moves all smaller elements between low..high to its left
             moves all larger elements between low..high to its right
             returns new position of pivot element

    randomly select pivot_index ∈ [low..high]
    pivot_item = array[pivot_index], swap array[low] with array[pivot_index]
    left = low + 1, right = high
    while left < right do
        left = find index of leftmost element > pivot_item    // left = high if none
        right = find index of rightmost element ≤ pivot_item
        if left < right then
            swap array[left] with array[right]
        end if
    end while
    array[low] = array[right], array[right] = pivot_item
    return right
```

... Randomised Quicksort

24/76

Analysis:

- Consider a recursive call to `partition()` on an index range of size s
 - Good call*:
 - both $\text{low}.. \text{pivot}-1$ and $\text{pivot}+1.. \text{high}$ shorter than $\frac{3}{4}s$
 - Bad call*:
 - one of $\text{low}.. \text{pivot}-1$ or $\text{pivot}+1.. \text{high}$ greater than $\frac{3}{4}s$
- Probability that a call is good: 0.5
(because half the possible pivot elements cause a good call)

Example of a bad call:

6 3 7 5 8 2 4 1

4 3 6 5 1 2 | 7 | 8

Example of a good call:

4 3 6 5 1 2 | 7 | 8

1 2 | 3 | 5 6 4 | 7 | 8

25/76

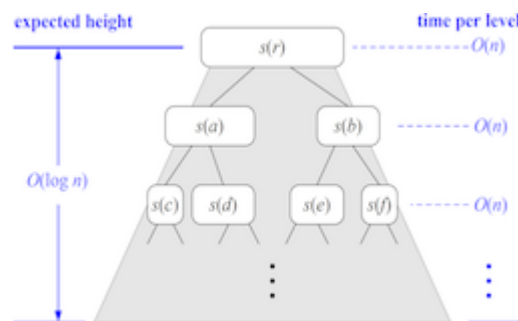
... Randomised Quicksort

n ... size of array

From probability theory we know that the expected number of coin tosses required in order to get k heads is $2 \cdot k$

- For a recursive call at depth d we expect
 - $d/2$ ancestors are good calls
 \Rightarrow size of input sequence for current call is $\leq (3/4)^{d/2} \cdot n$
- Therefore,
 - the input of a recursive call at depth $2 \cdot \log_{4/3} n$ has expected size 1
 \Rightarrow the expected recursion depth thus is $O(\log n)$
- The total amount of work done at all the nodes of the same depth is $O(n)$

Hence the expected runtime is $O(n \log n)$



Minimum Cut Problem

26/76

Given:

- undirected graph $G=(V,E)$

Cut of a graph ...

- a partition of V into $S \cup T$
 - S, T disjoint and both non-empty
- its *weight* is the number of edges between S and T :

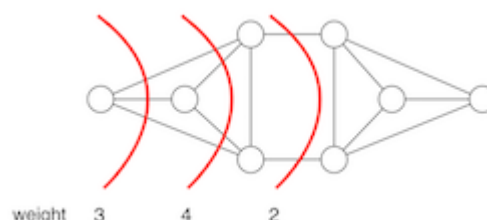
$$\omega(S, T) = |\{ \{s, t\} \in E : s \in S, t \in T \}|$$

Minimum cut problem ... find a cut of G with minimal weight

... Minimum Cut Problem

27/76

Example:

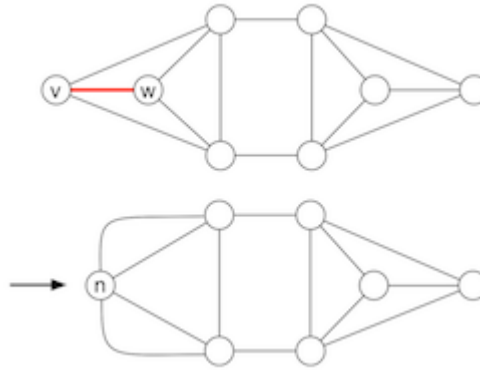


Contraction

Contracting edge $e = \{v, w\}$...

- remove edge e
- replace vertices v and w by new node n
- replace all edges $\{x, v\}, \{x, w\}$ by $\{x, n\}$

... results in a *multigraph* (multiple edges between vertices allowed) Example:



... Contraction

29/76

Randomised algorithm for *graph contraction* = repeated edge contraction until 2 vertices remain

```

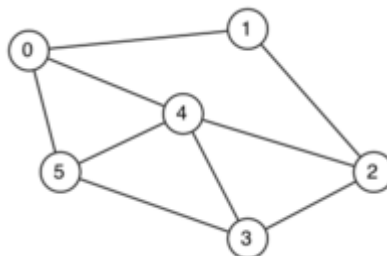
contract(G):
    Input  graph  $G = (V, E)$  with  $|V| \geq 2$  vertices
    Output cut of  $G$ 

    while  $|V| > 2$  do
        randomly select  $e \in E$ 
        contract edge  $e$  in  $G$ 
    end while
    return the only cut in  $G$ 
  
```

Exercise #2: Graph Contraction

30/76

Apply the contraction algorithm twice to the following graph, with different random choices:



... Contraction

31/76

Analysis:

V ... number of vertices

- Probability of **contract** to result in a minimum cut:

$$\geq 1 / \binom{V}{2}$$

- This is much higher than the probability of picking a minimum cut at random, which is

$$\leq \binom{V}{2} / (2^{V-1} - 1)$$

because every graph has $2^{V-1}-1$ cuts, of which at most $\binom{V}{2}$ can have minimum weight

- Single edge contraction can be implemented in $O(V)$ time on an adjacency-list representation \Rightarrow total running time: $O(V^2)$

(Best known implementation uses $O(E)$ time)

Karger's Algorithm

32/76

Idea: Repeat random graph contraction several times and take the best cut found

MinCut(G):

```

Input  graph G with  $V \geq 2$  vertices
Output smallest cut found

min_weight =  $\infty$ , d = 0
repeat
    cut = contract(G)
    if weight(cut) < min_weight then
        min_cut = cut, min_weight = weight(cut)
    end if
    d = d + 1
until d >  $\text{binomial}(V, 2) \cdot \ln V$ 
return min_cut

```

... Karger's Algorithm

33/76

Analysis:

V ... number of vertices

E ... number of edges

- *Probability of success:* $\geq 1 - \frac{1}{V}$
 - probability of not finding a minimum cut when the contraction algorithm is repeated $d = \binom{V}{2} \cdot \ln V$ times:

$$\leq \left[1 - 1 / \binom{V}{2} \right]^d \leq \frac{1}{e^{\ln V}} = \frac{1}{V}$$

- Total running time: $O(E \cdot d) = O(E \cdot V^2 \cdot \log V)$
 - assuming edge contraction implemented in $O(E)$

Sidetrack: Maxflow and Mincut

34/76

Given: flow network $G=(V,E)$ with

- edge weights $w(u,v)$

- source $s \in V$, sink $t \in V$

Cut of flow network G ...

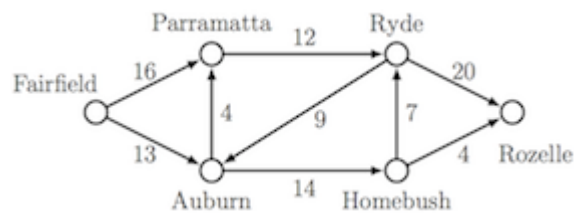
- a partition of V into $S \cup T$
 - $s \in S, t \in T$, S and T disjoint
- its *weight* is the sum of the weights of the edges between S and T :

$$\omega(S, T) = \sum_{u \in S} \sum_{v \in T} w(u, v)$$

Minimum cut problem ... find cut of a network with minimal weight

Exercise #3: Cut of Flow Networks

35/76



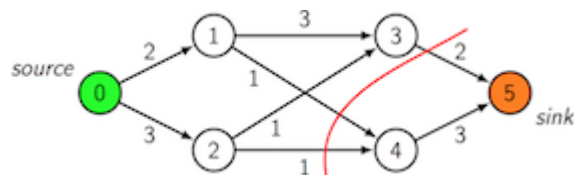
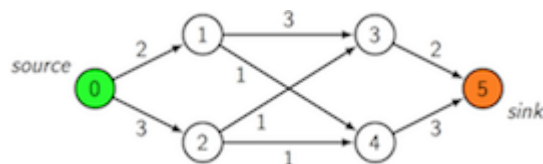
What is the weight of the cut $\{\text{Fairfield, Parramatta, Auburn}\}, \{\text{Ryde, Homebush, Rozelle}\}$?

$$12 + 14 = 26$$

Exercise #4: Cut of Flow Networks

37/76

Find a minimal cut in:



$$\omega(S, T) = 4$$

... Sidetrack: Maxflow and Mincut

39/76

Max-flow Min-cut Theorem.

In a flow network G the following conditions are equivalent:

1. f is a maximum flow in G
2. the residual network G relative to f contains no augmenting path
3. value of flow f = weight of some minimum cut (S, T) of G

40/76

Randomised Algorithms for NP-hard Problems

Many NP-hard problems can be tackled by randomised algorithms that

- compute nearly optimal solutions
 - with high probability

Examples:

- travelling salesman
- constraint satisfaction problems, satisfiability
- ... and many more

Simulation

Simulation

42/76

In some problem scenarios

- it is difficult to devise an analytical solution
- so build a software *model* and run *experiments*

Examples: weather forecasting, traffic flow, queueing, games

Such systems typically require random number generation

- distributions: uniform, numerical, normal, exponential

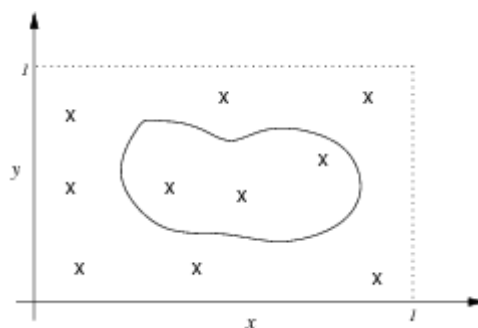
Accuracy of results depends on accuracy of model.

Example: Area inside a Curve

43/76

Scenario:

- have a closed curve defined by a complex function
- have a function to compute "X is inside/outside curve?"



... Example: Area inside a Curve

44/76

Simulation approach to determining the area:

- determine a region completely enclosing curve
- generate very many random points in this region

- for each point x , compute $inside(x)$
- count number of insides and outsides
- $areaWithinCurve = totalArea * insides / (insides + outsides)$

i.e. we approximate the area within the curve by using the ratio of points inside the curve against those outside

This general method of approximating is known as **Monte Carlo estimation**.

Summary

45/76

- Analysis of randomised algorithms
 - *probability of success*
 - *expected runtime*
 - Randomised Quicksort
 - Karger's algorithm
 - Simulation
 - Suggested reading:
 - Moffat, Ch. 9.3, 9.5
-

Algorithm and Data Ethics

Data Breaches

47/76

Major incidents ...

- *TJ Maxx credit and debit card theft (2005-07)*

Hackers gained access to accounts of over 100 million customers
⇒ Customers exposed to credit/debit card fraud
 - *Yahoo! data breach (2013-16)*

Hackers gained access to all 3 billion user accounts
Details taken included names, DOBs, passwords, answers to security questions
⇒ Customers exposed to identity theft
⇒ Over 20 class-action lawsuits filed against Yahoo!
 - *Facebook-Cambridge Analytica data scandal (2018)*

Millions of people's Facebook profiles used for political purpose without their consent
⇒ Cambridge Analytica went bust as a consequence
-

... Data Breaches

48/76

The Guardian, 30/03/15 ...

Personal details of world leaders accidentally revealed by G20 organisers

Exclusive: Obama, Putin, Merkel, Cameron, Modi and others kept in the dark after passport numbers and other details were disclosed in Australia's accidental privacy breach

- Follow our full coverage of this exclusive story
- Read the immigration department's letter outlining the circumstances of the G20 privacy breach



▲ Tony Abbott and Vladimir Putin cuddle koalas before the start of the first G20 meeting in November 2014. Photograph: Andrew Taylor/G20 Australia/Getty Images

The personal details of world leaders at the last G20 summit were accidentally disclosed by the Australian immigration department, which did not consider it necessary to inform those world leaders of the privacy breach.

The Guardian can reveal an employee of the agency inadvertently sent the passport numbers, visa details and other personal identifiers of **all world leaders attending the summit** to the organisers of the Asian Cup football tournament.

... Data Breaches

49/76

Australia's *Privacy Act 1988* ...

- outlines how personal information must be used and managed
- applies to government agencies, businesses and organisations with annual turnover of >\$3 million, private health services, ...

Individuals have the right to:

- have access to their personal information
- know why and how information is collected and who it will be disclosed to
- ask to stop unwanted direct marketing

Businesses and organisations must comply with the *Australian Privacy Principles*:

- how to collect personal information
- how (not) to use personal information
- how to secure personal information

... Data Breaches

50/76

Australia's Privacy Act 1988 *Notifiable Data Breaches scheme*

In the event of a **suspected or known data breach** ...

- contain breach where possible
- assess if personal information is likely to result in serious harm to affected individuals
 - individuals must be notified promptly
 - Australian Information Commissioner must also be notified
- take action to prevent future breaches

Data (Mis-)use

In 2012 several newspapers reported that ...

- Target used data analysis to predict whether female customers are likely pregnant
- Target then sent coupons by mail
- A Minneapolis man thus found out about the pregnancy of his teenage daughter

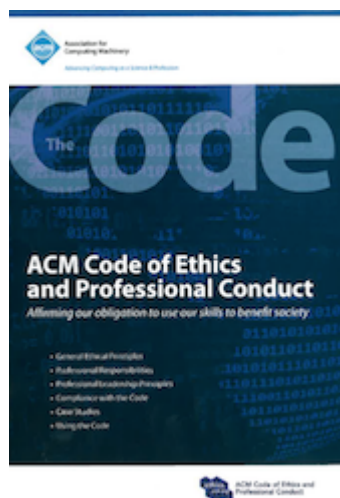
Not based on a factual story, but not implausible either

Who "owns" your data?

- big companies (Google, Facebook, ...)?
- governments?
- you?

... Data (Mis-)use

52/76



- Respect privacy
 - Store only the minimum amount of personal information necessary
 - Prevent re-identification of anonymised data
- Carefully analyse the consequences of data aggregation
- Access data only when authorised or compelled by the public good
 - **Whistleblower Manning's** disclosing of classified military data to Wikileaks (2010-11)
 - **Paradise papers** that disclosed offshore investments (2017)

Source: [ACM Code of Ethics and Professional Conduct](#)

Costly Software Errors

53/76

NASA's Mars Climate Orbiter ...

- launched 11/12/1998
- reached Mars on 23/9/1999
- came too close to surface and disintegrated

Cause of failure:

- spec said impulse must be calculated in *newton seconds*

- one module calculated impulse in *pound-force seconds*
- 1 newton \approx 0.2248 pound-force

... Costly Software Errors

54/76

Toyota vehicle recall (2009-11)

- Vehicles experienced sudden unintended acceleration
- 89 deaths have been linked to the failure
- 9 million cars recalled worldwide

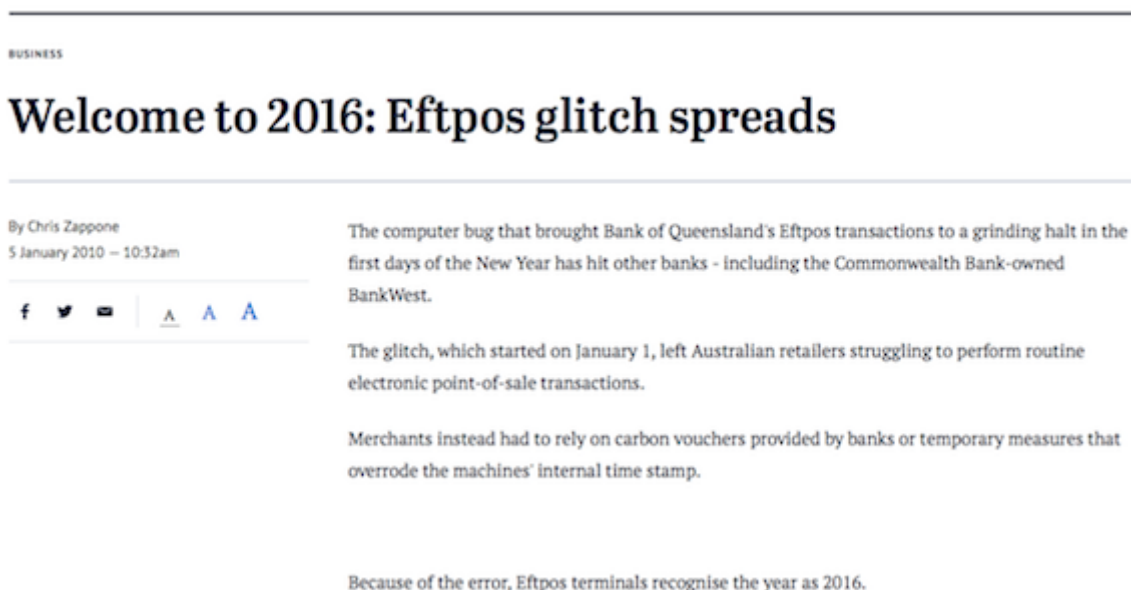
Causes of failure included ...

- a deficiency in the electronic throttle control system:
stack overflow
⇒ stack grew out of boundary, overwrote other data

... Costly Software Errors

55/76

Sydney Morning Herald, 05/01/10:



EFTPOS terminals inoperable for several days in early 2010

- customers' cards rejected as expired

Cause of failure:

- one module interpreted the current year as hexadecimal
 - **0x**09 = 09
 - **0x**10 = 16 (\neq 10)

Sidetrack: Year 2038 Problem

56/76

Recall:


```
#include <time.h>
time(NULL) // returns the time as the number of seconds
            // since the Epoch, 1970-01-01 00:00:00 +0000
```

Year 2038 problem ...

- **time(NULL)** on 19 January 2038 at 03:14:07 (UTC) will be 2147483647 = 0x7FFFFFFF
- a second later it will be **0x80000000** = -2,147,483,648
- $\Rightarrow -2^{31}$ seconds since 01/01/1970 ("Epoch") is 13 December 1901 ...

Programming Ethics

57/76

From the [ACM/IEEE Software Engineering Code](#) ...

- **Software engineers shall ensure that their products meet the highest professional standards possible**
 - Strive to fully understand the specifications for software
 - Ensure that specifications have been well documented and satisfy the users' requirements
 - Ensure adequate testing, debugging, and review of software and related documents
- Approve software only if it
 - **is safe**
 - **meets specifications**
 - passes appropriate tests
 - does not diminish quality of life, diminish privacy or harm the environment

... Programming Ethics

58/76

Algorithms can save lives.

Uberlingen airplane collision 1/7/02 at 11:35pm ...

- passenger jet V9 2937 and cargo jet QY 611 on collision course at 36,000 feet
- ground air traffic controller instructed V9 pilot to descend
- seconds later, the automatic Traffic Collision Avoidance System (TCAS)
 - instructed V9 2937 to climb
 - instructed QY 611 to descend
- flight 611's pilot followed TCAS, flight 2937's pilot ignored TCAS
- all 71 people on board the two planes killed

\Rightarrow Collision would not have occurred had both pilots followed TCAS

Exercise #5: Collision Avoidance Algorithm

59/76

The TCAS ...

- builds 3D map of aircraft in the airspace
- determines if collision threat occurs
- automatically negotiates mutual avoidance manoeuvre
- gives synthesised voice instructions to pilots ("climb, climb")

What algorithm would you use for reaching an agreement (climb vs. descent)?

Moral Dilemmas

How to program an autonomous car ...

- for a potential crash scenario
- when you have to choose between two actions that are both harmful

This is a modern version of the *Trolley Problem* ...

- A runaway trolley is on course to kill five people
- You stand next to a lever that controls a switch
- If the trolley is diverted, it will kill one person on the side track

Is it ethical to pull the lever and kill the one in order to save the five?

Exercise #6: Moral Dilemmas

61/76

What would you do?

Variations:

- Fat man on bridge
- Transplant

⇒ try it yourself on the [Moral Machine](#)

Course Review

Course Review

63/76

Goal:

For you to become competent Computer Scientists able to:

- choose/develop effective data structures
- choose/develop algorithms on these data structures
- analyse performance characteristics of algorithms (time/space complexity)
- package a set of data structures+algorithms as an abstract data type
- represent data structures and implement algorithms in C

Assessment Summary

64/76

```
assn1 = mark for programs/quizzes (out of 8+8)
assn2 = mark for mid-term test      (out of 12)
assn3 = mark for large assignment  (out of 12)
exam  = mark for final exam         (out of 60)
```

```
if (exam >= 25)
    total = assn1 + assn2 + assn3 + exam
else
    total = exam * (100/60)
```

To pass the course, you must achieve:

- at least **50/100** for **total**

which implies that you must achieve at least **25/60** for **exam**

... Assessment Summary

65/76

Check your results on WebCMS or using

```
prompt$ 9024 classrun -sturec
```

ClassKey: 20T2COMP9024	ClassKey: 20T2COMP9024
...	...
Assn1_weekly: 11.5/16	Assn1_weekly: 11.5/16
Assn2_midterm: 9/12	Assn2_midterm: 9/12
Assn3_goNSW: 8/12	Assn3_goNSW: 8/12
Exam: 43/60	Exam: 23/60
Total: 72/100	Total: 38/100

Final Exam

66/76

Goal: to check whether you have become a competent Computer Scientist

Requires you to demonstrate:

- understanding of fundamental data structures and algorithms
- ability to analyse time complexity of algorithms
- ability to develop algorithms from specifications

Lectures, problem sets and assignments have built you up to this point.

... Final Exam

67/76

2-hour (+10mins reading time) online test on **Monday 24th August**
2:00pm – 4:15pm

Must start test between 2pm and 2:05pm to get the full 130 minutes (= 2hrs+10mins)

- 8 multiple-choice questions, 4 open questions
 - Covers *all* of the contents of this course
 - Each multiple choice question is worth 4 marks ($8 \times 4 = 32$)
Each open question is worth 7 marks ($4 \times 7 = 28$)
 - Multiple-choice questions can have *one or more* correct options
 - You will get partial marks for partially correct answers
-

... Final Exam

68/76

Sample prac exam available on Moodle

- 2 multiple-choice questions, 2 open questions
- maximum time: 45 minutes
- feedback given immediately
 - marks for multiple-choice questions

- *sample* solution for open questions

... Final Exam

69/76

Of course, assessment isn't a "one-way street" ...

- I get to assess you in the final exam
- you get to assess me in UNSW's MyExperience Evaluation
 - go to <https://myexperience.unsw.edu.au/>
 - login using `zID@ad.unsw.edu.au` and your zPass

Response rate (as of Friday week 9): 16.9% 🥵

Please fill it out ...

- give me some feedback on how you might like the course to run in the future
- even if that is "Exactly the same. I like how it was run."

Revision Strategy

70/76

- Re-read lecture slides and example programs
- Read the corresponding chapters in the recommended textbooks
- *Review/solve problem sets*
- Attempt the prac exam on Moodle
- Invent your own variations of the weekly exercises (problem solving is a *skill* that improves with practice)

Supplementary Exam

71/76

If you attend an exam

- you are making a statement that you are "fit and healthy enough"
- it is your only chance to pass (i.e. no second chances)

Supplementary exam *only* available to students who

- do *not* attend the final exam *and*
- apply formally for special consideration
 - with a documented and accepted reason for not attending

If you experience a technical issue:

- Take screenshots of as many of the following as possible:
error messages, screen not loading, timestamped speed tests, power outage maps
- If issue was severe, apply for Special Consideration after conclusion of exam. Attach screenshots.

Note: Exam has been designed to account for up to 10mins of non-severe technical issues

Assessment

72/76

Assessment is about determining how well *you* understand the syllabus of this course.

If you can't *demonstrate your understanding*, you don't pass.

In particular, we don't pass people just because ...

- please, please, ... my parents will be ashamed of me
- please, please, ... I tried *really hard* in this course
- please, please, ... I'll be excluded if I fail COMP9024
- please, please, ... this is my final course to graduate
- etc. etc. etc.

Failure is a fact of life. For example, my scientific papers or project proposals get rejected sometimes too.

Summing Up ...

So What Was the Real Point?

74/76

The aim was for you to become a better computer scientist

- more confident in your own ability to design data structures and algorithms
 - with an expanded set of fundamental structures and algorithms to draw on
 - able to analyse and justify your choices
 - ultimately, enjoying the software design and development process
-

Finally ...

75/76



*Book 9
Epilogue*

Thus spake the Master Programmer:

"Time for you to leave."

... Finally ...

76/76

T h a t ' s A l l F o l k s

Stay Healthy

&&

Good Luck with the Exams

and with your future studies



Produced: 10 Aug 2020