

GEM 1501 Problem Solving With Computers

Lecture 5:

Correctness and Efficiency

Frank Stephan

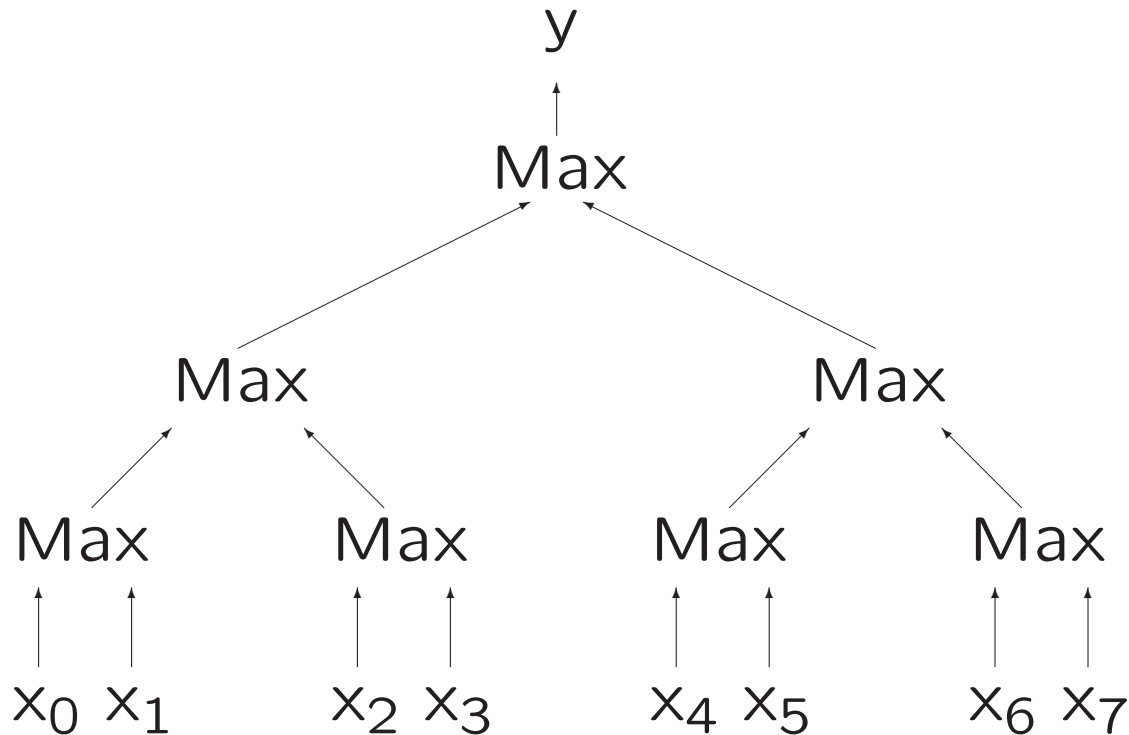
Summary of Previous Lecture

- Searches and traversals
- Divide and conquer
- Greedy algorithms
- Dynamic programming

Searches and traversals

- Exhaustive search is inefficient in large amount of data.
- Indices and Table of Contents in Books: pages are numbered, table of contents gives fast access to all chapters, indices permit to find information by keyword; alphabetical sorting of keywords in index supports faster finding.
- Internet search machines maintain large indices and records for supporting search. They contain copies of all relevant pages on the internet and use these in full text search.
- Databases to administer and maintain data including efficient methods to search, maintain and update the data. More than half of the data in a data base belongs to indices which support finding data fast.

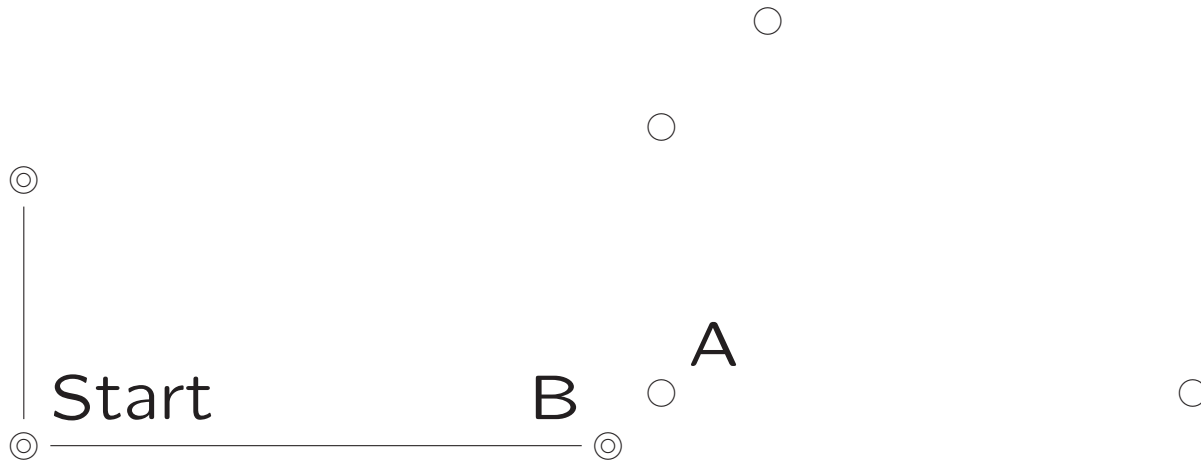
Maximum as Circuit



Divide and Conquer implemented as a Circuit.

Other algorithms: merge sort and pivot sort.

Greedy Algorithm - Networkconstruction



Goal: Increase network by shortest possible line.

Take city nearest to network: City A;

Connect it to nearest city in network: City B.

Local optimization gives global optimal solution.

Dynamic Programming – Solving Backwards

- Given: Cities (nodes) $a_0, a_1, a_2, \dots, a_n$
Costs $b_{i,j}$ for going from a_i to a_j with $i < j$
Impossible to go from a_i to a_j when $i \geq j$
- For $m = n - 1, n - 2, n - 3, \dots, 2, 1, 0$
Find cheapest path from a_m to a_n
Additional information to be stored:
 - Costs of cheapest path
 - Successor of a_m in this path
- Local optimization: cost from a_m to a_n through a_k is $b_{m,k}$ plus costs from a_k to a_n ; path is from a_m to a_k plus cheapest path from a_k to a_n
Cheapest path has to be compared to direct path from a_k to a_n having costs $a_{k,n}$
- Final table gives costs from a_0 to a_n
Path given by following immediate successors of each node

Overview of this Lecture

- **Correctness of algorithms**
 - Kinds of errors
 - Sources of errors
 - Correctness
- Efficiency of algorithms
 - Example
 - Big-O
 - Complexity of Algorithms and Problems

Computer Errors Do Happen

- Mars Polar Lander project started 3 January 1999 and was lost on 3 December 1999 after 11 months in space, traveled at least 35 million miles, cost of approximately USD 165 million and only 40 meters from landing.
- Premature engine shutdown most likely cause when Touchdown Monitor (TDM) software falsely indicated landing.
- Verification activities were comprehensive and performed by dedicated, experienced engineers.
- Fault not malicious, just difficult to find.
- According to Bob Knickerbocker (Director of Software at Lockheed Martin), Lockheed Martin had serendipitously found the bug a couple months after the crash.

Language Errors I

- Certain language constructions are formally illegal and browsers do not execute Java Script having such errors.
- Java Script Editors can find these syntactical errors.
- Example:
for (k=0;k<101) { sum = sum+k; }
instead of
for (k=0;k<101;k=k+1) { sum = sum+k; }
- There must be exactly three parts in the bracket of a for-statement which are separated by semicolons.

Language Errors II

- Certain unintended language constructions are formally legal.
- Example: `for (k=0;k=k+1;k<101) { sum = sum+k; }`
instead of `for (k=0;k<101;k=k+1) { sum = sum+k; }`
- Proper execution needs that the condition when to continue the loop is the second one.
- Reason:
“`k=k+1`” returns the value assigned to k also as the value of a condition. This value is interpreted as “true” if it is positive and as “false” if it is 0. So “`k=k+1`” is a syntactical legal condition. Furthermore, there are no constraints how the third condition updates k , anything is permitted in Java Script.
- Bad Style and Syntactically Illegal is not the same.
- Further example: `for (k=10;k=k-1;document.write((10-k)+" "));`
Produced output: 1 2 3 4 5 6 7 8 9

Restrictive Languages

- Restrictive languages like Pascal permit to catch many errors.
- Syntactically incorrect Pascal program:

```
program euclid(input,output);
var first, second, third: integer;
begin writeln('Input two positive numbers ');
      readln(first,secodn);
      while first <> second do begin
          if first>second then first = first-second
              else second := second-first end;
          third := 'Output is '+first
      writeln(third) end end.
```

- Errors found by compiler:

variable “secodn” undeclared (due to misspelling)

there should be an assignment, not a comparison after “then”

type conflict when adding a text and integer variable “third”

one “end” too much at the end of the program.

Techniques for analyzing programs

- Syntax analysis
- Type checking
- Type inference
- Abstract interpretation
- Code verification

Computers cannot catch everything

- Sample Program

```
document.write("The Square Numbers <br>");  
var x; var y;  
for (x=0;x<100;x=x+1)  
  { y = x+x;  
    document.write("The square of "+x+" is "+y+".<br>"); }  
}
```

- Computer cannot know that you want $y = x*x$; instead of $y = x+x$;
This error is not found by automatic tools.
- Many of such errors are less obvious for the human.

Logical Errors

- Remember the program that counts the number of sentences in which “money” appears.
- This program fails on the two following sentences and counts four:
The time came to count the **money** and while my brother typed “. . .” into the computer, my sister counted everyone’s **money**. Her **money** on the bank account is SGD 45.55 and my brother’s **money** is SGD 80.80.
- The logical assumption that a dot defines always the end of a sentence was wrong. A much more careful analysis is necessary. One has to take care of quotes, dots in abbreviated words and decimal points in numbers.

The Source of Errors in Cooking

- Recipe wrong
- Cook incompetent
- Food spoiled
- Hardware like oven defective

In cooking and also in computing, errors can occur in every step and component of the process.

Errors: The Usual Suspects

- Hardware?

Very rare, but if found such errors become famous:
“Intel Inside, Cannot Divide.”

http://en.wikipedia.org/wiki/Pentium_FDIV_bug

- System Problems?

If system is overloaded, the browser might be unable to read the Java Script program.

- Compilers, interpreters?

Occurs from time to time although is not that frequent.

- Software!

Most frequent source of errors.

The newest programs have the most errors.

A Famous Error

- It is quite common to write down only the last two digits of a year.
- One does not know whether the birthday “12.02.02” is “12.02.1902” or “12.02.2002”.
- This caused frequently errors at the registration of primary school pupils.
- Today mayors like to visit people turning 106 or 107 with presents plus a letter asking their (late) parents to enrol their child for primary school.

Fighting Errors in Software

- Prevent errors from being made (rigorous, disciplined software design)
- Find errors and remove them (debugging)
- Systematically search for erroneous behaviour and correct it (testing)
- Software verification, correctness proofs

Infinite Loops and Similar Problems

Some programs do not finish in time.

```
(1) y=0; for (x=100;x>0;x=x+1) { y=y+x; } alert("sum is "+y);
```

Variable x is **updated in wrong direction**, the loop is infinite.

```
(2) function fibonacci(z)
```

```
    { var r = 1;
```

```
      if (z>1) r = fibonacci(z-1)+fibonacci(z-2);
```

```
      return(r); }
```

```
u = fibonacci(10000); alert("10000th Fibonacci Number is "+u);
```

This program terminates only theoretically. All existing computers will eventually abort it or being aborted by their users. 10000th Fibonacci Number can be computed, but not this way.

```
(3) p=4; while (!(isprime(p)&&isprime(p+2)&&isprime(p+4))) { p=p+1; }
```

```
    document.write("Prime triple is ("+p+", "+(p+2)+", "+(p+4)+").");
```

Does not terminate as (3,5,7) is the only prime triple.

Program Specification

1. Specification of set of legal inputs
2. The relationship between the inputs and the desired outputs

Specifying Euclid's Algorithm

- Input: Two positive integers x, y ;
- Output: Positive integer z ;
- Common Divisor: z divides x and z divides y ;
- Greatest divisor: For any integer $u > z$, either u does not divide x or u does not divide y .

Partial and Total Correctness

- Partial correctness: Whenever the program terminates on a legal input, the desired relationship holds on the input and the output.
- Termination: The program terminates.
- Total correctness: Partial correctness + Termination

Floyd's Method

- Annotate program with assertions and convergents
- Prove these assertions and convergents
- Conclude correctness from assertions and convergents

Verifying Euclid's Algorithm

- **Specification**

Function *euclid* should compute greatest common divisor of x, y .

- **Implementation**

```
function euclid(x,y)
  { var v=x; var w=y;
    while (v != w)
      { if (v>w) { v = v-w; }
        if (w>v) { w = w-v; } }
    return(v); }
```

- **Notation:** Let z denote the greatest common divisor of x, y .

- **Main Assertions throughout the algorithm:**

Assertion 1: v and w are multiples of z ;

Assertion 2: $v \geq z$ and $w \geq z$;

Assertion 3: There is no larger common divisor of v, w than z .

Verification 1

- **Initialization**

```
var v=x; var w=y;
```

Assertion 1, 2, 3 are true by copying x, y into v, w .

- **If statement**

```
if (v>w) { v = v-w; }
```

Mathematical fact:

If $v > w$ then the common divisors of v, w and $v - w, w$ are the same.

If z divides v, w and $v > w$ then $v - w > 0$ and $v - w \geq z$.

All three assertions are preserved by this part of code.

- **Similar analysis** applies to statement

```
if (w>v) { w = w-v; }
```

Verification 2

- **The while-loop**

```
while (v != w)
  { if (v>w) { v = v-w; }
    if (w>v) { w = w-v; } }
```

- **Partial Correctness**

All three assertions remain true when executing a single statement in the body of the loop.

If loop terminates, $v == w$. First, v is greatest common divisor of v, w . Second, $v == z$.

- **Convergence and correctness**

Whenever the program goes into the body of the loop, $v \geq z > 0$ and $w \geq z > 0$ and $v \neq w$.

A positive number will be subtracted from either v or w .

The sum $v + w$ goes down by 1 each time the loop is executed.

As $v + w == x + y$ at beginning, loop is executed at most $x + y$ times.

Verification 3

- **Seen before**

Loop converges and $v == w$ afterwards. v is greatest common divisor of v and w .

By Assertions 1, 2, 3, $z == v$ after termination of loop.

- **Return Statement**

```
return(v);
```

Algorithm returns a variable having the value z which is the greatest common divisor of x and y .

Function *euclid* is correct.

Example: Reversing a List

- Given a list in an array “givenlist” , produce a new array “reverselist” where the elements are in reverse order.
- Idea is to use a stack: always pop the last element of givenlist and push it at the end of the reverselist. Stop when givenlist is empty.
- Implementation and Runtime Example

```
var reverselist = new Array();  
while (givenlist.length > 0)  
    { reverselist.push(givenlist.pop()); }
```

givenlist	reverselist
(7,10,23,21,3)	()
(7,10,23,21)	(3)
(7,10,23)	(3,21)
(7,10)	(3,21,23)
(7)	(3,21,23,10)
()	(3,21,23,10,7)

Computer and Mathematical Proofs

- Pythagoras: $3^2 + 4^2 = 5^2$.
Formulas for all positive integers a, b, c with $a^2 + b^2 = c^2$.
- Pierre de Fermat 1637: For all $n > 2$, $a^n + b^n \neq c^n$.
Found after his death 1665 without proof.
- Leonard Euler 1769: Generalizing $a^3 + b^3 \neq c^3$.
 $a^4 + b^4 + c^4 \neq d^4$, $a^5 + b^5 + c^5 + d^5 \neq e^5$ and so on.
- Computer-Search 1966: $27^5 + 84^5 + 110^5 + 133^5 = 144^5$.
- Andrew Wiles 1993: Proof of Fermat's Last Theorem without computers. Only experts understand the proof.

Automated Proofs

- Computers are used to proof correctness of programs and of other mathematical theorems.
- The Four Colour Theorem
Francis Guthrie coloured 1852 a map of England with four colours: neighbouring counties have different colours.
Kenneth Appel and Wolfgang Haken, 1976:
 - (a) The Four Colour Theorem is equivalent to being able to colour approximately 1936 basic forms of maps.
 - (b) Computer found four-colourings for these 1936 maps.

Can we trust computerized proofs?

- Is the search for the four-colourings of the basic maps correctly implemented?
- Is the software correct?
- Is the hardware correct?
- Are the language processors correct?

Several mathematicians have reproven the Four-Colour Theorem with different case-distinctions on other computers.

Overview of this Lecture

- Correctness of algorithms
 - Kinds of errors
 - Sources of errors
 - Correctness
- **Efficiency of algorithms**
 - Example
 - Big-O
 - Complexity of algorithms and problems

Efficiency of Algorithms 1

Example: Normalize the score of a class of students.

- set MAX to 0;
- for I from 1 to N do:
 - if $SCORE[I] > MAX$ then $MAX = SCORE[I]$;
- for I from 1 to N do:
 - $SCORE[I] = SCORE[I] \times 100 / MAX$;

Efficiency of Algorithms 2

Second version: Only one division needed.

- set MAX to 0;
- for I from 1 to N do:
 - if $SCORE[I] > MAX$ then $MAX = SCORE[I]$;
- set $FACTOR = 100/MAX$;
- for I from 1 to N do:
 - $SCORE[I] = SCORE[I] \times FACTOR$;

Order of Functions

- Quadratic Order

$$n^2, 2 \cdot n^2 + 17, 550 \cdot n^2 + 8, 371 \cdot n + 277.$$

- Formally

f has (at most) quadratic order iff there is a constant c such that for all $n > c$, $f(n) \leq c \cdot n^2$.

- $f \in O(g) \Leftrightarrow \exists c \forall n > c (f(n) \leq c \cdot g(n))$.
- $f \in O(g) \Leftrightarrow O(f) \subseteq O(g)$.
- f and g have same order $\Leftrightarrow O(f) = O(g)$.

Why Only Order of Algorithm

- Machine-Dependent whether basic instruction needs 0.01 sec or 0.0000001 sec, technical progress halves actual speed within one year.
- Order of algorithms can be computed fast from that of subalgorithms since minor terms can be dropped in this process.
- Basic runtime properties of algorithm can be seen well from Big-O-Analysis.
- It is better to improve from $3 \cdot n^2$ to $8 \cdot n \cdot \log(n)$ than to $2 \cdot n^2$, order analysis puts priority on right things.
- Detailed analysis can still be done if really needed.

Runtime of Algorithm

```
function listadd(list) // Adding Some Listelements
  { var n = list.length; var m; var k; 0(1)
    var sum=0; 0(1)
    for (m=0;m<n;m=m+1)
      { sum = sum+list[m]; sum=sum+1 } 0(n)
    for (m=0;m<n;m=m+1)
      { for (k=0;k<n;k=k+1)
        { sum = sum+list[m]*list[k]; } } 0(n*n)
    for (m=0;m<10;m=m+1)
      { sum = sum+list[0]; } 0(1)
    return(sum); }
```

Overall runtime is $O(1+1+n+n*n+1)$ which is $O(n*n)$

Reason: $n*n$ grows faster than n . Here some examples:

n	1	2	3	4	5	6	7	8
5*n	5	10	15	20	25	30	35	40
n*n	1	4	9	16	25	36	49	64
const	10	10	10	10	10	10	10	10

Classes of Big-O

- Logarithmic algorithms: $O(\log(n))$
example: binary search
- Linear algorithms: $O(n)$
example: find maximum in unordered list, reverse list
- Pseudo-linear algorithms: $O(n \cdot \log(n))$
example: merge sort
- Quadratic algorithms: $O(n^2)$
example: bubble sort

Binary search

- Given ordered list

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	2	12	17	22	23	29	32	34	34

- Is number 30 in list?

Known: a[10] first element not to exist.

Search from	to	middle	a[middle] <= target
0	10	5	YES
5	10	7	NO
5	7	6	YES
6	7	Done	

Search terminates as $a[6] \leq \text{target} < a[7]$.

As $a[6] \neq \text{target}$, target is not in list.

Binary Search

- Algorithm (Rounding ignored)

```
lower = 0; upper = list.length;
```

```
while (lower+1 <= upper)
```

```
  { if (list[(lower+upper)/2] <= target)
```

```
    then { lower = (lower+upper)/2; }
```

```
    else { upper = (lower+upper)/2; } }
```

```
if (list[lower]==target)
```

```
  then "target is in" else "target is not in"
```

- Runtime Analysis

Each time the algorithm goes through the loop, the difference $upper - lower$ is halved.

Runtime is $O(\log(n))$ where n is length of list.

Time Analysis of Reverse

- Algorithm is $O(n)$ as runtime example shows.

Original List

(7,10,23,21,3)

(7,10,23,21)

(7,10,23)

(7,10)

(7)

()

Reversed List

()

(3)

(3,21)

(3,21,23)

(3,21,23,10)

(3,21,23,10,7)

- Each element has to be touched, cannot be improved.

Time Analysis of Bubble Sort

- Algorithm

```
n = list.length;
while (list is not sorted)
  { for (m=0;m<n-1;m=m+1)
    { if (list[m]>list[m+1])
      { swap list[m] and list[m+1]; } } }
```

- Every forloop has $n - 1$ comparisons

If the smallest element is at the beginning the element `list[n-1]` then it takes $n - 1$ rounds until the smallest element is `list[0]`. So $n - 1$ rounds in worst case.

- Worst Case Complexity $O(n^2)$

Time Analysis of Merge Sort

- Merge Sort
 - * Split list in two halves
 - * Call merge sort for each half of at least two elements
 - * Merge the two halves using at most one comparison per element in final merged list.
- Recurrence relation, rounding ignored.
 - If $n > 2$, $C(n) = n + 2 * C(n/2)$.
 - If $n \leq 2$, $C(n) = 1$.

Recurrence for the case $n = 2^m$

$$\begin{aligned}C(n) &= C(2^m) \\&= 2^m + 2 \cdot C(2^{m-1}) \\&= 2^m + 2 \cdot (2^{m-1} + 2 \cdot C(2^{m-2})) \\&= 2^m + 2^m + 2^2 \cdot C(2^{m-2}) \\&= k \cdot 2^m + 2^k \cdot C(2^{m-k}) \\&= (m-1) \cdot 2^m + 2^{m-1} \cdot C(2^1) \leq m \cdot 2^m\end{aligned}$$

In general, merge sort uses $O(n \cdot \log(n))$ comparisons.

Upper and Lower Bounds

- Often it is useful to get general bounds on the possible efficiency
- Lower bounds: What is the least number of operations required?
- Example: Search in an unsorted list requires $O(n)$ comparisons.
- Upper bounds: A given big-O result provides an upper bound for other possible algorithms

Lower bound for Towers of Hanoi

- Let $T(n)$ be minimal number of moves needed for n rings.
- $T(1)$ is clearly 1.
- Having $T(n)$, one can analyze fastest algorithm for $n + 1$.
First and last move deal with smallest ring.
Any move of larger ring is followed by move of smallest ring.
Ignoring smallest ring, the other moves give algorithm for n .
- $T(n + 1) \geq 2 \cdot T(n) + 1$.
- $2^n - 1$ is lower bound which matches known algorithm.
- Four instead of three pegs: 33 instead of 255 moves for 8 rings, improvement might be possible.
Five instead of three pegs: 31 instead of 1023 moves.
<http://www.comp.nus.edu.sg/~gem1501/assignment08.html>

How many Comparisons for Sorting

- Task: Sort elements of (unsorted) list.
- Algorithm can compare pairs of elements and copy elements between array positions and variables.
No other information about elements available.
- Outcome of sorting procedure depends on the number of comparisons, each comparison has two possible answers, YES for $x < y$ and NO for $x \geq y$.
- 1 comparison permits to sort two elements $a_0 a_1$.
 $a_0 < a_1$: YES $\Rightarrow a_0 a_1$ is correct list;
 $a_0 < a_1$: NO $\Rightarrow a_1 a_0$ is correct list.

Sorting Three Elements

- 3 comparisons permit to sort three elements $a_0 a_1 a_2$:
 $a_0 < a_1$: YES, $a_1 < a_2$: YES $\Rightarrow a_0 a_1 a_2$
 $a_0 < a_1$: YES, $a_1 < a_2$: NO, $a_0 < a_2$: YES $\Rightarrow a_0 a_2 a_1$
 $a_0 < a_1$: YES, $a_1 < a_2$: NO, $a_0 < a_2$: NO $\Rightarrow a_2 a_0 a_1$
 $a_0 < a_1$: NO, $a_0 < a_2$: YES $\Rightarrow a_1 a_0 a_2$
 $a_0 < a_1$: NO, $a_0 < a_2$: NO, $a_1 < a_2$: YES $\Rightarrow a_1 a_2 a_0$
 $a_0 < a_1$: NO, $a_0 < a_2$: NO, $a_1 < a_2$: NO $\Rightarrow a_2 a_1 a_0$
- 2 comparisons give four possible lists $a_i a_j a_k$, one for each of the outcomes YES YES, YES NO, NO YES, NO NO of the two comparisons.
- There are six possibilities
 $a_0 a_1 a_2, a_0 a_2 a_1, a_1 a_0 a_2, a_1 a_2 a_0, a_2 a_0 a_1, a_2 a_1 a_0$.
- If $a_0 a_2 a_1$ does not appear in the output then the algorithm is false on input $a_0 = 15, a_1 = 20, a_2 = 19$.

Sorting n Elements

- An algorithm making at most m comparisons can produce at most 2^m different outputs.
- There are $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$ many permutations of $a_0 a_1 \dots a_{n-1}$.
- Half of the factors of $n!$ is at least $n/2$, thus $n! > (n/2)^{n/2}$.
- $2^m \geq n!$ gives $m > \log((n/2)^{n/2}) = n(\log(n) - 1)/2$.
- If $f(n)$ is the number of comparisons necessary to sort n elements then $n \log(n) \in O(f)$. So the order of any correct sorting algorithm is at least $n \log(n)$.
- Merge sort has this complexity, thus lower and upper bound are the same.

An Alternative Approach to Sorting

- List of numbers between 0 and 99.
- Count for each number a how often it occurs.
- Output for $a = 0, 1, \dots, 99$ the number a as often as it is in the list.
- Example: 2 2 8 3 2 3 \Rightarrow
3 times 2, 2 times 3, 1 times 8 \Rightarrow
2 2 2 3 3 8.
- Is this faster?

Sorting faster in special case

- Sorting n numbers between 0 and 99 has linear time.

```
function countsort(list)
  { var m; var n=list.length; var i; var j;
    var count = new Array(100);
    for (i=0;i<100;i=i+1) { count[i]=0; }
    for (m=0;m<n;m=m+1)
      { count[list[m]] = count[list[m]]+1; }
    m=0; for (i=0;i<100;i=i+1)
      { for (j=0;j<count[i];j=j+1)
        { list[m]=i; m=m+1; } } }
```

- Runtime of function countsort: first loop 100 times, second loop n times, inner part of third nested loop also n times $\Rightarrow O(n)$.
- **$O(n)$ is much faster than $O(n \cdot \log(n))$.**
- **Model matters:** this is a **special case** and the **access is more powerful** than just comparing.

Average Case Complexity

- What is the runtime for a **typical** input?
- Characterize **typical**!
- Much more effort required for good estimates.

Average Case for Sorting

- Assumption: all elements of list different.
- Smallest element with probability 0.5 in second half.
- For bubble sort, smallest element goes one position to front per round.
- Sorting needs at least $n/2$ rounds for half of possible inputs.
- Bubble sort has quadratic average case complexity.

Next Week

- Midterm examination 1
 - Midterm 1 covers Week 1 to Week 5
 - Midterm 2 covers Week 6 to Week 11
 - Final examination covers everything
 - Source: Slides, Assignments, Sample programs, Books
- Format of midterm examinations
 - 12 Questions graded in a 0-1-scheme each
 - Number of marks = number of correct answers
- Totals of marks
 - 12 for Midterm 1
 - 12 for Midterm 2
 - 26 for Assignments
 - 50 for Final Examination
- Selected Topics of Java Script

Test Questions 1

- What is the origin of the name “Algorithm” ?
 - (a) The Greek word for mathematics
 - (b) The name of the mathematician Mohammed al-Khowârizmî
 - (c) The name of the programming language Algol 58
- The following program for a function has a syntax error in each line. Underline the corresponding word or symbol.

```
function factorial(var n)
  { var do; var res = 1; var m = 1;
    while (m <= n)
      { res = res*m: m=m+1; }
    returning(res) }
```

Test Questions 2

- Complete the following Java Script program for bubble sort

```
var u;  
for (i=0;i<n-1;i=_____  
    { for (j=_____;j<n-1;j=j+1)  
        { if (list[j] > list[_____  
            { u=list[j]; list[j]=list[j+1]; list[j+1]=_____; } } }
```

- Write a function which computes the n-th Fibonacci number without recursion but instead uses an array.

Solutions

(1) (b) The word algorithm follows the name of the Arab mathematician.

(2) The syntactical correct function is:

```
function factorial(n)           // no "var" before "n"
  { var d; var res=1; var m=1; // variable names must not be keywords
    while (m<=n)                // "while" is spelled with one "l"
      { res = res*m; m=m+1; }    // there is no colon after a statement
    return(res); }              // "return" instead of "returning"
```

(3) The complete program for bubble sort is the following.

```
var u;
for (i=0;i<n-1;i=i+1)
  { for (j=0;j<n-1;j=j+1)
    { if (list[j] > list[j+1])
      { u=list[j]; list[j]=list[j+1]; list[j+1]=u; } } }
```

(4) function fibonacci(n)

```
{ var m = 1; var fibo = new Array(0,1,1);
  while (m<n) { fibo.push(fibo[m-1]+fibo[m]); m=m+1; }
  return(fibo[n]); }
```