CSE5311

Design and Analysis of Algorithms

What are algorithms?

- An algorithm is a precise and unambiguous specification of a sequence of steps that can be carried out to solve a given problem or to achieve a given condition.
- An algorithm is a computational procedure to solve a well defined computational problem.
- An algorithm accepts some value or set of values as input and produces a value or set of values as output.
- An algorithm transforms the input to the output.
- Algorithms are closely intertwined with the nature of the data structure of the input and output values.
  - Data structures are methods for representing the data models on a computer whereas data models are abstractions used to formulate problems.

Where do we use Algorithms?

- *Everyday Life*
  - Going from Point A to Point B
  - A recipe for preparing a food item
  - Decision making
- *Computer Science*
  - AI
  - Databases
  - Networks
  - Multimedia Systems
- *Biology*
  - Bioinformatics
  - Ant colonies
- *Economics*
- *Marketing*
- *Running a Business*
- *Music*
- *Games*
- *Others … please add*

Example Algorithms

- An algorithm to sort a sequence of natural numbers into non-decreasing order
  - Application : Lexicographical ordering
- An algorithm to find a shortest path from one node to another in a graph
  - Application: Routing in computer networks
- An algorithm to find the best scheduling of events to resources
  - Application: Lecture halls to courses
- An algorithm to recognize a substring in a string of letters
  - Application: Word *find* in a text.

Real-life Examples

- Travelling sales person problem
- Google
  - Probe, crawl, search, sort, rank….
- Amazon
  - Search, mine, match, rank….
- Travelocity
  - Search, mine, match, rank,…
- Netflix (http://netflix.com)
  - ***New Releases, Classics, TV episodes and more on DVD***
    Over 100,000 DVD titles.
    - **100 shipping points nationwide and more than 95 percent of our members receive their DVDs in about one business day.**

Problem 1

- ***A man needs to transport a wolf, a goat and a cabbage across a river. The boat has room only for the man and one other item (either the wolf, the goat or the cabbage). In the absence of the man the wolf would eat the goat and the goat would eat the cabbage. Solve this problem for the man.***

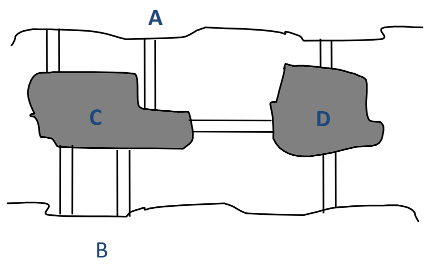
Problem 2

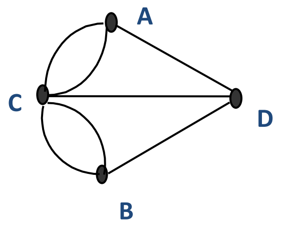
- **Four persons A,B, C, and D wish to cross a bridge. It is dark at night and they need to use the only flashlight in their possession, that has a battery life only 17 mins. A maximum of two people can cross the bridge at any given time. Each person walks at a different pace and a pair must walk at the slower person's pace. The times taken by the four persons (if allowed to cross individually) are given as: A- 1 min; B – 2 mins; C – 5 mins; and D-10 mins;**

Problem 3

- **The town of Konigsberg (now Kaliningrad) lay on the banks and on two islands of the Pregel river. The city was connected by 7 bridges. The puzzle (as encountered by Leonhard Euler in 1736) : Whether it was possible to start walking from anywhere in town and return to the starting point by crossing all bridges exactly once.**

Konigsberg bridges





Course Description

- Design and Analysis of Algorithms is THE most important basic course in any graduate computer science and engineering curriculum.
  - It is vital for every computer science student to be fluent with algorithms and their analysis.
  - Students are encouraged to solve homework problems and discuss/solve problems in the class.
  - Each student will be given one specific algorithm or problem to carry out an in-depth study.
- Typically, this course should be taken in the very first semester of your graduate study because algorithms are used in Networks, Operating Systems, Databases, and other (including advanced) courses.

Course Objectives

- The objective of this course is to build a solid foundation of the most important fundamental subject in computer science. Creative thinking is essential to algorithm design. Algorithm analysis and verification demands sound mathematical acumen and programming skills.

Mode of Teaching

- The class meets twice a week (Mondays and Wednesdays 1:00 to 2:20pm).
  - Power point slides and other lecture material will be used.
  - At the end of each topic, students must attempt to solve exercise problems.
  - There will be no specific text book for the class –
    - *the instructor will provide comprehensive notes and references to relevant material.*
    - *Exercise problems can be found on the course web page and in reference books.*
    - *All students are expected to work on these problems and participate in the class discussions.*
- The course on Algorithms is critical to your development as a computer scientist, a researcher, a creative thinker and/or a problem solver.
  - algorithms are extensively used in
    - *databases, networks, artificial intelligence, bioinformatics, pervasive and mobile computing, robotics, security, architecture, all engineering and science disciplines, finance, management, music, biology and indeed in everyday life.*
- You will be encouraged to think on your own and to discuss solutions with your peers.
- The course is not limited to any programming language.
- Students are strongly encouraged to use reference books and course material that will be available and updated from time to time on the course page.

Information

- **Instructor**
  - Mohan Kumar
    - *333 NH*
    - ***Email**: mkumar@uta.edu*
    - ***Phone**: (817) 272-3610*
- **Class**:
  - Mon/Wed - 1:00 to 2:20 PM 314 PH
- **Office Hours**
  - Mon/Wed - 2:30 to 4:00 PM
- **Course site**:
  - http://crystal.uta.edu/~kumar/cse5311_09FLDAA
- **GTA**: TBA

Syllabus

- Review of Asymptotic Analysis and Growth of Functions;
  - Trees, Heaps, and Graphs; and Recurrences.
- Greedy Algorithms:
  - Minimum spanning tree, Union-Find algorithms, Kruskal's Algorithm, Clustering, Huffman Codes, and Multiphase greedy algorithms.
- Dynamic Programming:
  - Shortest paths, negative cycles, matrix chain multiplications, sequence alignment, RNA secondary structure, application examples.
- Network Flow:
  - Maximum flow problem, Ford-Fulkerson algorithm, augmenting paths, Bipartite matching problem, disjoint paths and application problems.
- NP and Computational tractability:
  - Polynomial time reductions; The Satisfiability problem; NP-Complete problems; and Extending limits of tractability.
- Approximation Algorithms
  - Local Search and Randomized Algorithms
- Applications of Algorithms, sample examples

Reference Books

- Class Notes, Power Point Slides, and Exercise Problems
  - *Mohan Kumar*
- Algorithm Design
  - *Jon Kleinberg and Éva Tardos*, Pearson Addison-Wesley
- The Design and Analysis of Algorithms 1974
  - *AV Aho, JE Hopcroft and JD Ullman*, Addison-Wesley Publishing Company
- Introduction to Algorithms: A Creative Approach, Reprinted 1989
  - *Udi Manber*, Addison-Wesley Publishing Company
- Introduction to Algorithms, Second Edition, 2001
  - *T Cormen, C E Leiserson, R L Rivest and C Stein* McGraw Hill and MIT Press
- Graph Algorithms, 1979
  - *Shimon Even*, Computer Science Press
- The Design & Analysis of Algorithms, 2005
  - *Anany Levitan*, Addison Wesley
- The Art of Computer Programming, Vols. 1 and 3
  - *Knuth*, Addison Wesley Publishing Company

Assessment

- Quizzes and class participation: 40%
- The structure of the quizzes will be discussed in class, at least one week prior to the quiz.
  - **Quiz 1** (10%): September 9, 2009
  - **Quiz 2** (10%): September 23, 2009
  - **Quiz 3** (10%): October 07, 2009
  - **Quiz 4** (10%): October 28, 2009
- **Final Exam** (30 %): December 02, 2009
- Lab Assignment: 30%
- Quizzes 1 thru 4 are of duration 60 minutes and the Final Exam is of duration 2 hours.

QUESTIONS?

Study of Algorithms

- Designing algorithms
- Expressing algorithms
- Algorithm Validation
- Algorithm Analysis
- Alternative techniques

Algorithms vs. Program Code

Algorithms

- *An algorithm,*
    - **is an abstraction of an actual program**
    - **is a computational procedure that terminates**
- *An algorithm is composed of a finite set of steps,*
    - **each step may require one or more operations,**
    - **each operation must be definite and effective**

Program Code

- ***A program is an expression of an algorithm in a programming language.***
- ***A program code conforms to the dictates of policies and limitations of a programming language.***

Presenting algorithms

- **Description : The algorithm will be described in English, with the help of one or more examples**

- **Specification : The algorithm will be presented as pseudo code**

  (We don't use any programming language)

- **Validation : The algorithm will be proved to be correct for all problem cases**

- **Analysis: The running time or time complexity of the algorithm will be evaluated**

Algorithms

- An algorithm is designed to solve a given problem
- An algorithm does not take into account the intricacies and limitations of any programming language.
  – we are free to express ourselves when designing an algorithm.
- An algorithm should be unambiguous
  – it should have precise steps
- An algorithm has three main components:
  – input
  – the algorithm itself and
  – output.
- An algorithm will be implemented using a programming language
- An algorithm designer is like an architect while programmers are like masons, carpenters, plumbers etc.)

Algorithms (Contd.)

- *The algorithms we design should be*
  - Simple
    - ***Unambiguous (e.g. The students should understand algorithms the instructor gives in the class and the GTA should understand the algorithms students write in a test or exam)***
  - Feasible
    - ***Should be implementable using a programming language and***
    - ***executable on a computer.***
  - Cost effective
    - ***CPU time***
    - ***Memory used***
    - ***Communication***
    - ***Energy***

Algorithm Design

- Abstract solution to the problem
  - Algorithmic solution to problems are applicable to many applications
- Resource limitations and constraints
  - Time
    - *Most common criteria*
    - *Modern applications demand more computing power and time*
  - Memory
    - *Modern applications demand more computing power and time*
    - *Data in main memory*
  - Energy
    - *Critical to battery operated devices*
- Application requirements
  - Input/output limitations
  - Time, space

Algorithm Evaluation

- *We evaluate the performance of algorithms based on*
  - **time** (CPU-time) and
  - **Space** (semiconductor memory)
    - *Both are expensive*
- *The time taken to execute an algorithm is dependent on one or more of the following,*
  - ***number of data elements***
  - ***the degree of a polynomial***
  - ***the size of a file to be sorted***
  - ***the number of nodes in a graph***
- *Computer scientists should endeavour to minimize time, space and energy required.*

Expressing Algorithms

Procedure ALGO_X (A [1,…,n])

- **Input : unsorted array A**
- **Output : Sorted array A**

```
for i ← 1 to n-1
        small ← i;
    for j ← i+1 to n
      if A[j] < A[small]  then
          small ← j;
          temp ← A[small];
          A[small] ← A[i];
          A[i] ← temp;
end
```

Procedure **AlGO_Y (A[1,…,n],i,n)**

- **Input : Unsorted array A**
- **Output : Sorted array A**

```
while  i < n
    do   small ← i;
        for  j ← i+1 to n
             if A[j] < A[small] then
                small ← j;
                temp ← A[small];
                A[small] ← A[i];
                A[i] ← temp;
                ALGO_Y(A,i+1,n)
        end
end
```

Analyzing Algorithms

Procedure **AlGO_Y (A[1,…,n],i,n)**

- **Input : Unsorted array A**
- **Output : Sorted array A**

```
while  i < n
    do  small ← i;
        for  j ← i+1 to n
            if A[j] < A[small] then
                small ← j;
                temp ← A[small];
                A[small] ← A[i];
                A[i] ← temp;
                ALGO_Y(A,i+1,n)
end
```

- We start with data size *n*
- Last line – same algorithm recalled, but for data size *n*-1
- The algorithm takes (*n*-1) steps to find the smallest element in the array
- $T(n) = \underline{T(n-1)} + \underline{b * n}$

Recursive call for data size *n*-1

Steps for finding the smallest element plus swap

Analysis (Contd.)

- $T(n) = T(n-1) + b.\ n$ (1)
- $T(n-1) = T(n-2) + (n-1)b$ (2)
- $T(n-2) = T(n-3) + (n-2)\ b$ (3)
- . . .
- $T(n-i) = T(n-(i+1)) + (n-i)b$ (4)

  Using (2) in (1)

  $T(n) = T(n-2) + b\ [n+(n-1)]$

  $= T(n-3) + b\ [n+(n-1)+(n-2)]$

  $= T(n-(n-1)) + b[n+(n-1)+(n-2) + . . . +(n-(n-2))]$

  - $T(n) = O(n^2)$

Asymptotic Notations

– O-notation

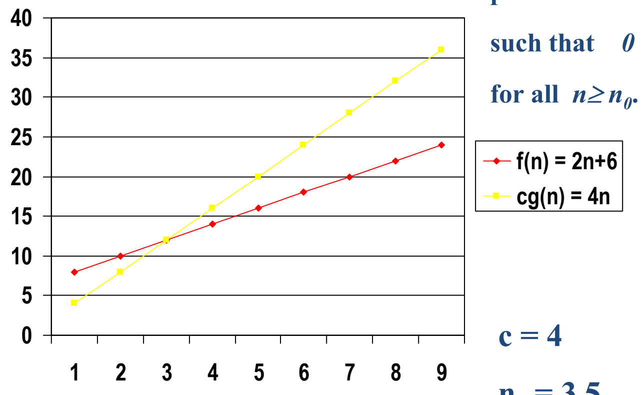
» *Asymptotic upper bound*

- A given function *f(n)*, is *O (g(n))* if there exist positive constants *c* and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$.
- *O (g(n))* represents a set of functions, and

$O(g(n)) = \{f(n)$: there exist positive constants *c* and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0\}$.

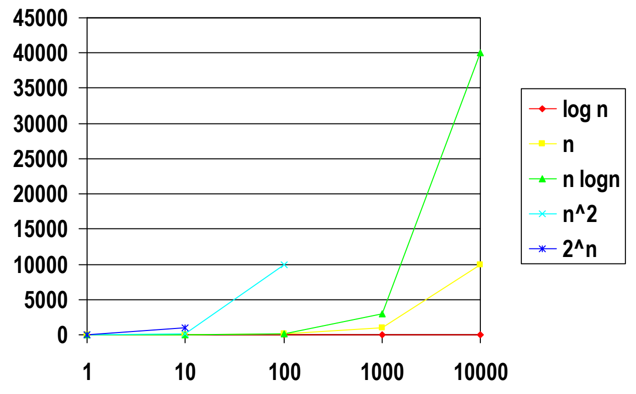
O Notation

f(n), is *O (g(n))* if there exist positive constants *c* and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$.



c = 4

$n_0 = 3.5$



Ω-notation

Asymptotic lower bound

- A given function *f(n)*, is Ω *(g(n))* if there exist positive constants *c* and n_0 such that $0 \le c\, g(n) \le f(n)$ for all $n \ge n_0$.
- Ω *(g(n))* represents a set of functions, and

$\Omega(g(n)) = \{f(n)$: there exist positive constants *c* and n_0 such that $0 \le c\, g(n) \le f(n)$ for all $n \ge n_0\}$

Θ-notation

Asymptotic tight bound

- A given function *$f(n)$*, is *$\Theta(g(n))$* if there exist positive constants *c_1*, *c_2*,and *n_0* such that *$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$* for all *$n \geq n_0$*.
- *$\Theta(g(n))$* represents a set of functions, and

$\Theta(g(n)) = \{f(n)$: there exist positive constants *c_1*, *c_2*, and *n_0* such that *$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$* for all *$n \geq n_0$*.

O, Ω, and Θ correspond (loosely) to "$\leq$", "$\geq$", and "$=$".

Running Times and Space

- How many times each statement is executed?
  - Are there loops in the algorithm?
  - Is the algorithm iterative, repetitive, recursive etc.
  - How much memory is used in executing the algorithm?

Algorithm complexity

	$\log_2 n$	*n*	*$n\log_2 n$*	*n^2*	*n^3*	*2^n*	*$n!$*
n=10	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 secs
n=20	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} yrs
n=50	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	36 yrs	🟥
n=100	< 1 sec	< 1 sec	< 1 sec	< 1 sec	1 sec	10^{17} yrs	🟥
$n=10^3$	< 1 sec	< 1 sec	< 1 sec	1 sec	18min	🟥	🟥
$n=10^4$	< 1 sec	< 1 sec	< 1 sec	2 min	12 days	🟥	🟥
$n=10^5$	< 1 sec	< 1 sec	2 secs	3 hours	32 yrs	🟥	🟥
$n=10^6$	< 1 sec	1 sec	20 secs	12 days	31710 yrs	🟥	🟥

Processor performing 1 million high-level instructions per second

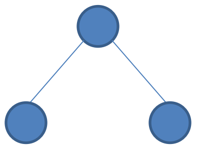
J. Kleinberg and E. Tardos, Algorithm Design, Addison Wesley, 2005

Constant Time

- Constant number of statements
  - e.g., Let X = 4;
  - Y = 6;
  - if A[j] < A[small] then A[j] = SMALL
- The complexity (or running time) is *O(1)*

Logarithmic Time

- Divide and conquer algorithm
- Problem divided into two or more equal parts and each part solved recursively
- Binary Search Tree
  - ***T (n) = c • T (n/2) + O(1)***
  - ***Time to solve problem of size n is equal to time to solve problem of size n/2 (multiplied by a constant) PLUS constant time.***



Linear Time

- The running time increases linearly with the size of the problem
  - Computing the minimum of *n* data elements
  - Merging two sorted lists
- *$O(n \log_2 n)$* time algorithms
  - Merge sort
  - Quick sort
  - Heap sort

Quadratic Time

- There are *n* points in a plane. If each point is specified by its (x,y) coordinates, find the closest pair of points.
- A brute force algorithm

```
D_min = ∞
For each point $p_i (x_i, y_i)$
        for each point $p_j (x_j, y_j)$  such that $i \neq j$
                compute $D_{ij} = \sqrt{[ (x_i - x_j)^2 + (y_i - y_j)^2 ]}$
                If $D_{ij} < D_{min}$ then $D_{ij} = D_{min}$
        End
End
```

  - What is the complexity of the algorithm?
  - n(n-1)/2 computational steps
  - **Quadratic time**

Polynomial Time

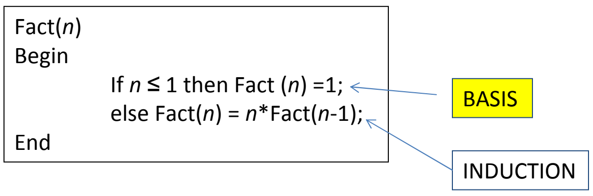
- Problems that can be solved in polynomial time
  - Algorithms when implemented, can be executed in polynomial time – O(n^k)

Beyond Polynomial Time

- Some problems cannot be solved in polynomial time
- There are NO known polynomial solutions for these problems
  - Traveling Salesperson is a classic example of such a problem
  - We will study such problems and approximate solutions to these problems

Recursive Algorithms

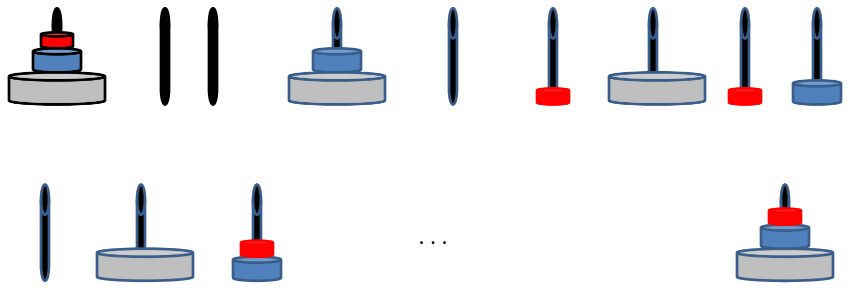
- A recursive function is one that is called from within its own body.
  - The call be direct or indirect
    - *F calls itself; F1 calls F2, F2 calls F1.*



```
Fact($n$)
Begin
        If $n \leq 1$ then Fact ($n$) =1;
        else Fact($n$) = $n$*Fact($n$-1);
End
```

Applications of recursion

- The Towers if Hanoi problem consists of three pegs A, B, and C, and *n* rings of varying size. Initially the rings are stacked on peg A in order of decreasing size, the largest ring at the bottom. The problem is to move the rings from peg A to peg B one at a time in such a way that no ring is ever placed on a smaller ring. Peg C may be used for temporary storage of rings. Write a recursive algorithm to solve this problem. What is the execution time of your algorithm in terms of the number of times a ring is moved?



Recursive Solution

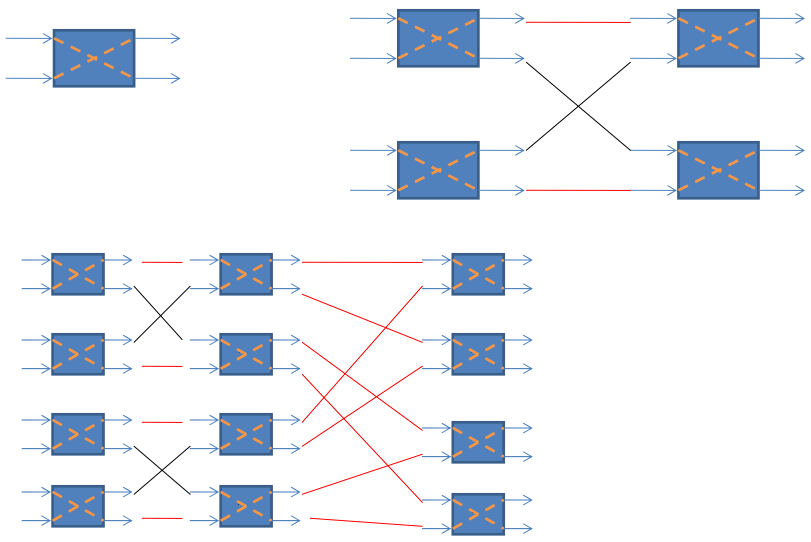
- The problem
  - Move *n* rings from A to B, using C as auxiliary

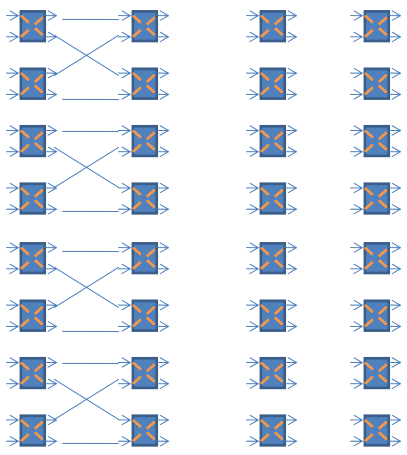
```
if n > 1
    Begin
        move recursively (n-1) rings from A to C, using B as auxiliary
        move largest disk from A to B
        move recursively (n-1) disks from C to B, using A as auxiliary
    End
```

- Time taken

$$T(1) = 1$$
$$T(n) = 2T(n-1) + 1$$

Switches





What is the recursion?
Define?

Exercise Problems

1. Write a recursive algorithm to find the maximum of *n* real numbers in an array *A [0 .. n-1].* What is the complexity of your algorithm*?*

2. Derive an expression to find the sum of the first *n* squares, where n is a positive integer. Provide a proof for the sum using induction. Write an algorithm to find the sum of the first *n* squares. What is the complexity of the algorithm?

3. The input is a set *S* containing *n* real numbers, and a real number *x.*

a. Design an algorithm to determine whether there are two elements of *S* whose sum is exactly *x.* The algorithm should run in *O(n log n)* time*.*

b. Suppose now that the set *S* is given in a sorted order. Design an algorithm to solve the above problem in time O *(n).*

4. Given an array of integers *A[1..n],* such that, for all *i, 1 ≤i ≤ n,* we have *|A[i]-A[i+1]| ≤ 1.* Let *A[1]* = x and *A[n]* = y*,* such that *x < y.* Design an efficient search algorithm to find *j* such that *A[j] = z* for a given value *z, x ≤ z ≤ y.* What is the maximal number of comparisons to *z* that your algorithm makes*?*

Exercise Problems(Contd.)

- The Towers if Hanoi problem consists of three pegs A, B, and C, and *n* rings of varying size. Initially the rings are stacked on peg A in order of decreasing size, the largest ring at the bottom. The problem is to move the rings from peg A to peg b one at a time in such a way that no ring is ever placed on a smaller ring. Peg C may be used for temporary storage of rings. Write a recursive algorithm to solve this problem. What is the execution time of your algorithm in terms of the number of times a ring is moved?

Compare the following pairs of functions in terms of order of magnitude. In each case, say whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, and/or $f(n) = \Theta(g(n))$

	$f(n)$	$g(n)$
a.	$100n + \log n$	$n + (\log n)^2$
b.	$\log n$	$\log(n^2)$
c.	$n^2/\log n$	$n(\log n)^2$
d.	$(\log n)\log n$	$n/\log n$
e.	$\sqrt{n}$	$(\log n)^5$
f.	$n\,2^n$	3^n