CSE 5311 Design and Analysis of Algorithms

Fall 2007

Instructor: Dr. Mohan Kumar

Venue: 106CH

Time: M/W 1:00 – 2:20 PM

Example Algorithm: 1

A man needs to transport a wolf, a goat and a head of 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.

All on LB
Man and Goat *cross* (Cabbage and Wolf on left bank)
Man returns (Goat on right bank)
Wolf and Man *cross* (Cabbage on LB and Goat on RB)
Man and Goat *return* (Wolf on RB and Cabbage on LB)
Man and Cabbage *cross* (Goat and LB, Wolf on RB)
Man *returns* (Cabbage and Wolf on right bank)
Man and Goat *cross* (All on RB)

Input

Output

Resources

Conditions

Limitations

Example Algorithm: 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;

A,B *cross bridge (2mins)*

A *returns* with FL *(1 min)*

C,D *cross bridge (10 mins)*

B returns (2 mins)

A, B cross bridge (2mins)

Input

Output

Resources

Conditions

Limitations

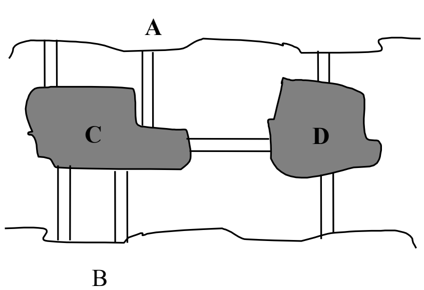
Processor

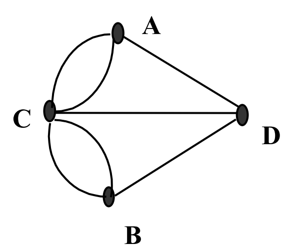
Memory

Time

Example Algorithm: 3

Konigsberg bridges





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.

Course Syllabus

- Review of Asymptotic Analysis and Growth of Functions, Recurrences
- Trees, Heaps, and Graphs;.
- 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

Course Info

- **Instructor**: Mohan Kumar, 333 NH **Email**: mailto:kumar@cse.uta.edu **Phone**: (817) 272-3610
- **Class**: Mon/Wed - 1:00 to 2:20 PM
- **Office Hrs.:** Tue – 1:30 to 3:00 PM and Wed - 2:30 to 4:00 PM
- **Course site**: http://crystal.uta.edu/~kumar/cse5311_07FALL
- **GTA**: TBA

Books

- **Text book**
- **Algorithm Design**
  by Jon Kleinberg, Éva Tardos
- Pearson Addison-Wesley
- ISBN 0-321-29535-8
- **References**
- Class Notes, Power point slides, and Exercise Problems
- 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
- Introduction to the Theory of Computation, 1992
  – Michael Sipser, PWS Publishing Company
- 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 12, 2007
- Quiz 2 (10%): September 26, 2007
- Quiz 3 (10%): October 10, 2007
- Quiz 4 (10%): October 31, 2007
- **Final Exam (25 %):** November 28, 2007.
- Quizzes 1 thru 4 are of duration 30 minutes and the Final Exam is of duration 2 hours.
- **Group Project: 35%**

Group Project: 35%

- Students will have the option of doing a group study or group project.
- Project problems will be handed out by September 15, 2007 and the expected date of Completion is November 30, 2007. The students will be required to write programs and run experiments.
- *Presentation and demonstration of the projects/research problem will be during the first week of December 2007.*

Homework and Class Participation

- **Homework Assignments**: No Grades awarded directly!
- **Class participation**: ACTIVE Participation will prepare you well for Quizzes and Exams Students are expected to interact actively during lectures. All students are expected to solve homework problems and discuss solutions in the class.

CSE5311 Design and Analysis of Algorithms

- This Class
  - What is an algorithm?
  - Asymptotic Analysis
  - Iterative algorithms
  - Recursive algorithms

- At the end of the class
  - Difference between an algorithm and a program
  - O, Ω, and Θ notations
    - How to use them
    - Determine complexity of a given algorithm
  - Write recurrence relations for your algorithms

Chapters 1 and 2

Algorithm Design *Kleinberg and Tardos*

Course Syllabus

- **Review of Asymptotic Analysis and Growth of Functions, Recurrences**
- Trees, Heaps, and Graphs;.
- 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

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.

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 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.
- A computer algorithm is a computational procedure to solve a well defined computational problem.

Hereafter, we mean computer algorithm when we say 'algorithm'

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. In other words, 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:
  - The input
  - the algorithm itself and
  - the 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

- 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

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
    - At colonies
- Economics
- Marketing
- Running a Business
- Music
- Games
- Others … please add

Problem types

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

What are these algorithms?
Input? Output? Complexity?

ALGO_DO_SOMETHING (A [1,…,n],1,n))

```
•1.for i ← 1 to n-1
•2.        small ← i;
•3.            for j ← i+1 to n
•4.                if A[j] < A[small] then
•5.                    small ← j;
•6.            temp ← A[small];
•7.            A[small] ← A[i];
•8.            A[i] ← temp;
•9.end
```

ALGO_IMPROVED (A[1,…,n],i,n)

```
•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_IMPROVED(A,i+1,n)
•End
```


Examples

An algorithm to sort a sequence of numbers in nondecreasing order.

Application : lexicographical ordering

An algorithm to find the shortest path from a source node to a destination node in a graph

Application: To find the shortest path from one city to another.

An algorithm to fill a knapsack with the most cost effective objects

Application: An algorithm to increase the 'hit ratio' of a cache

- **Data Models:**

  Lists, Trees, Sets, Relations, Graphs

- **Data Structures :**

  Linked List is a data structure used to represent a List

  Graph is a data structure used to represent various cities in a map.

SELECTION SORT Algorithm (*Iterative method)*

Procedure SELECTION_SORT (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
```

Example: Given sequence

	5	2	4	6	1	3
i=1	1	2	4	6	5	3
i=2	1	2	4	6	5	3
i=3	1	2	3	6	5	4
i=4	1	2	3	4	5	6

```
1.   for i ← 1 to n-1
2.     small ← i;
3.       for j ← i+1 to n
4.           if A[j] < A[small] then
5.             small ← j;
6.       temp ← A[small];
7.       A[small] ← A[i];
8.       A[i] ← temp;
9.   end
```

Complexity:

The statements 2,6,7,8, and 5 take O(1) or constant time.
The outer loop 1-9 is executed n-1 times and the inner loop 3-5 is executed (n-i) times.
The upper bound on the time taken by all iterations as i ranges from 1 to n-1 is given by, **$O(n^2)$**

- Study of algorithms involves,
  - **designing algorithms**
  - **expressing algorithms**
  - **algorithm validation**
  - **algorithm analysis**
  - **Study of algorithmic techniques**

Algorithms and Design of Programs

- *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
- *An algorithm*,
  - ∗ is an abstraction of an actual program
  - ∗ is a computational procedure that terminates

*A program is an expression of an algorithm in a programming language.
*Choice of proper data models and hence data structures is important for expressing algorithms and implementation.

- We evaluate the performance of algorithms based on
  - time (CPU-time) and
  - Space (semiconductor memory)
    - Both are expensive
  - computer scientists should endeavour to minimize time taken and space required.

- 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

Asymptotic Notations

– **O-notation**

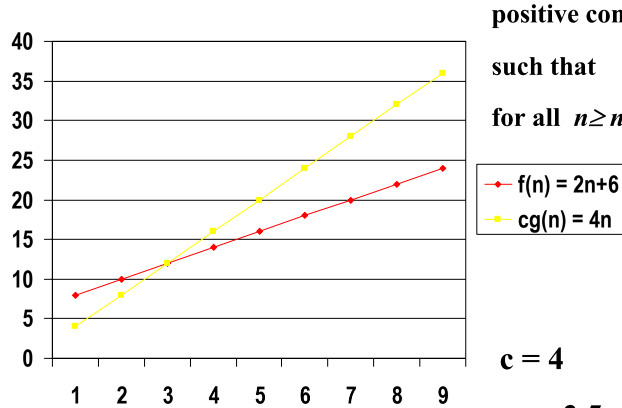
» ***Asymptotic upper bound***

- **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 $n \geq n_0\}$.

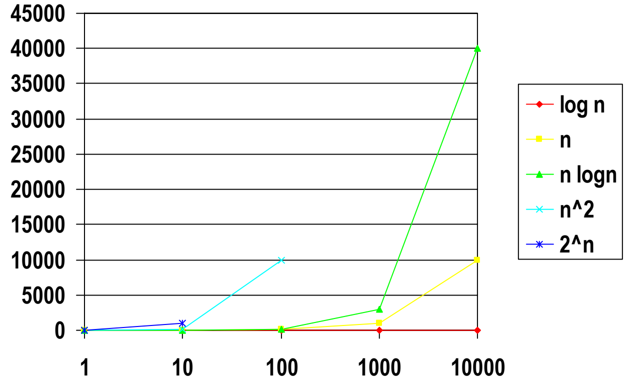
O Notation

$f(n)$, 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 $\Omega(g(n))$ if there exist positive constants c and n_0 such that $0 \le c\, g(n) \le f(n)$ for all $n \ge n_0$.**
- **$\Omega(g(n))$ 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 $0 \le c_1 g(n) \le f(n) \le c_2\, g(n)$ for all $n \ge n_0$.**
- **$\Theta(g(n))$ represents a set of functions, and**

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

O, Ω, and Θ correspond (loosely) to "$\le$", "$\ge$", 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?

We should endeavor to design algorithms that run fast and use least possible memory.

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

$$T(n) = c \bullet T(n/2) + O(1)$$

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

Please note: It is Log to base 2, in most cases

Linear Time

- The running time increases linearly with the size of the problem
- Computing the minimum of *n* numbers

```
MIN = A[1]
FOR i = 2 to n
        IF A[i] < MIN then
            MIN = A[i]
```

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

O ($n \log_2 n$) time

- Some sorting algorithms have this complexity
- e.g. Merge sort
  - Divide the input into two equal parts
  - Sort each part and merge the two parts together, recursively
- $T(n) = c \bullet T(n/2) + O(n)$

  $= O(n \log n)$

Quadratic Time

- The selection sort algorithm

  $T(n) = T(n\text{-}1) + O\ (n)$

During each big step,

Problem is reduced from size i to i-1

Each big step takes O(n) 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

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 pseudocode**

  (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**

SELECTION SORT Algorithm (*Iterative method)*

Procedure SELECTION_SORT (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
```

$O(n^2)$

Recursive Selection Sort Algorithm

Given an array A[i, …,n], selection sort picks the smallest element in the array and swaps it with A[i], then sorts the remainder A[i+1, …, n] recursively.

Example :
Given A [26, 93, 36, 76, 85, **09**, 42, 64]

Swap 09 with 23 -- A[1] = 09; A[2,…, 8] = [93,36,76,85,26,42,64]
Swap 26 with 93 -- A[1,2]= [09,26]; A[3,…,8] = [36,76,85,93,42,64]
No swapping -- A[1,2,3] = [09,26,36]; A[4,…,8] =[76,85,93,42,64]
Swap 42 with 76 -- A[1,…,4] =[09,26,36,42]; A[5,…,8] = [85,93,76,64]
Swap 64 with85 -- A[1,…,5] =[09,26,36,42,64]; A[6,7,8] = [93,76,85]
Swap 76 with 93 -- A[1,…,6]=[09,26,36,42,64,76]; A[7,8] = [93,85]
Swap 85 with 93 -- A[1,…,7]=[09,26,36,42,64,76,85]; A[8] = 93

Sorted list : A[1,…,8] = [09,26,36,42,64,76,85,93]

Procedure **RECURSIVE_SELECTION_SORT (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;
       RECURSIVE_SELECTION_SORT(A,i+1,n)
End
```


The two Algorithms

- **SELECTION SORT Algorithm (***Iterative method)*
- **Input : Unsorted array A**
- **Output : Sorted array A**
- **Procedure SELECTION_SORT (A [1,…,n])**
- **Input : unsorted array A**
- **Output : Sorted array A**

```
1.   for i ← 1 to n-1
2.      small ← i;
3.         for j ← i+1 to n
4.             if A[j] < A[small] then
5.                small ← j;
6.         temp ← A[small];
7.         A[small] ← A[i];
8.         A[i] ← temp;
9.   end
```

- Procedure **SELECTION_SORT (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;
RECURSIVE_SELECTION_SORT(A,i+1,n)
End
```

Analysis of Recursive selection sort algorithm

Basis: If $i = n$, then only the last element of the array needs to be sorted, takes Θ (1) time.

Therefore, $T(1) = a$, a constant

Induction : if $i < n$, then,

1. we find the smallest element in A[i,…,n], takes at most (n-1) steps

    swap the smallest element with A[i], one step

    recursively sort A[i+1, …, n], takes T(n-1) time

Therefore, T(n) is given by,

$T(n) = T(n-1) + b. n$ (1)

It is required to solve the recursive equation,

$T(1) = a$; for $n = 1$

$T(n) = T(n-1) + b n$; for $n > 1$, where b is a constant

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

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

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

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

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

Questions:

- What is an algorithm?
- Why should we study algorithms?
- Why should we evaluate running time of algorithms?
- What is a recursive function?
- What are the basic differences among O, Ω, and Θ notations?
- Did you understand selection sort algorithm and its running time evaluation?
- Can you write pseudocode for selecting the largest element in a given array? Please write the algorithm in the class.

Home work: Please read

Chapters 1 and 2, Algorithm Design *Kleinberg and Tardos*

Next Class: Wednesday (8/29)

- Overview of Mathematical Induction
- Complexities of problems
- Recursive equations
- Problems will be solved in the class on the board