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**
- 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

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.

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

- **Algorithms:**

An algorithm to sort a sequence of numbers into 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.

- **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 outerloop 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) required to implement these algorithms. However, both these are expensive and a computer scientist should endeavor 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**

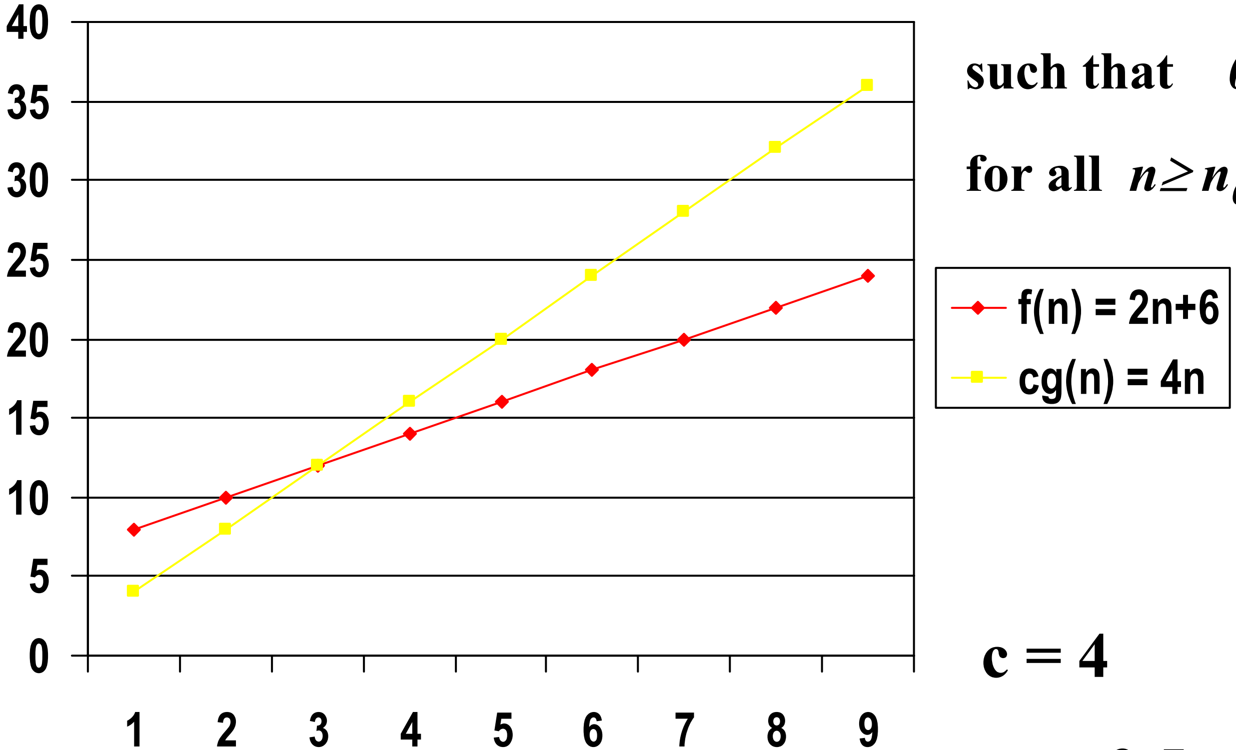
» ***Asymptotic upper bound***

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

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

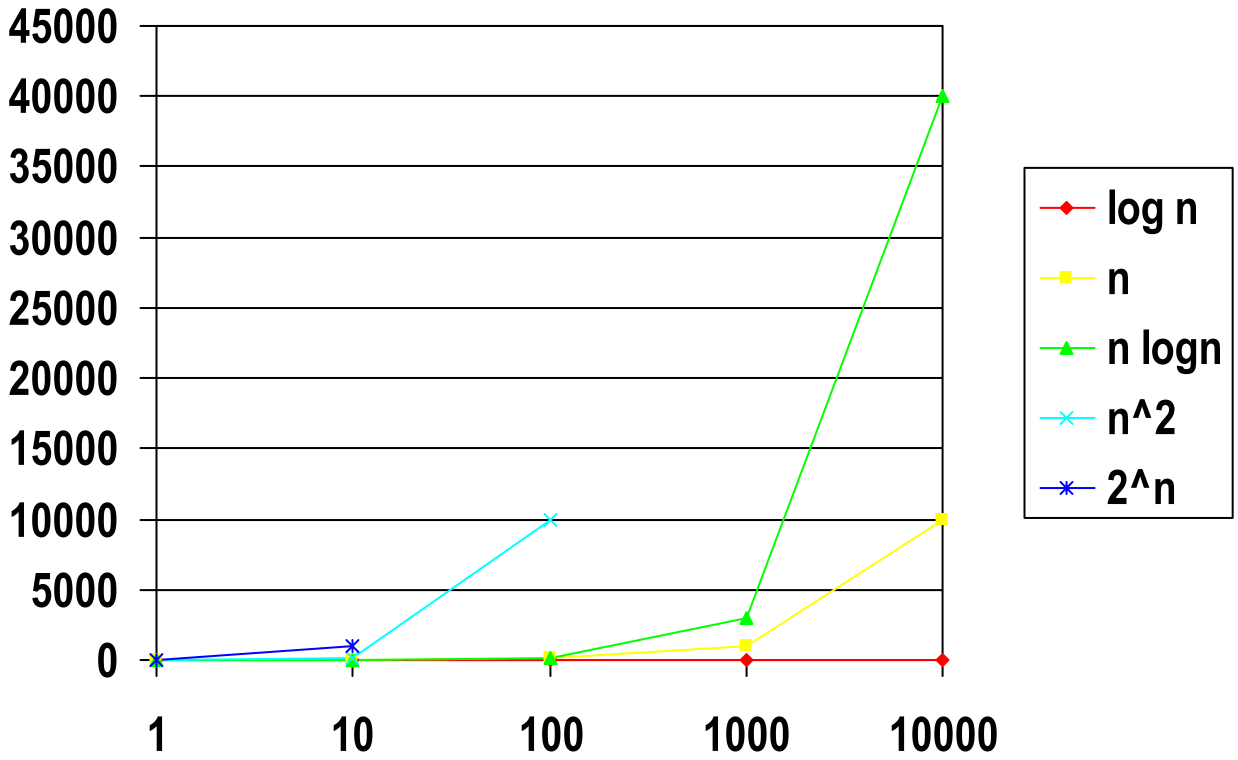
O Notation

***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


45000
40000
35000
30000
25000
20000
15000
10000
5000
0
1
10
100
1000
10000
log n
n
n logn
n^2
2^n

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 $0 \leq c_1 g(n) \leq f(n) \leq c_2\, g(n)$ 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 "≤", "≥", and "=".

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
```

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
```

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

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

$T(n-2) = T(n-3) + (n-2)\, b \quad (3)$

. . .

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

Using (2) in (1)

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

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

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

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

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*