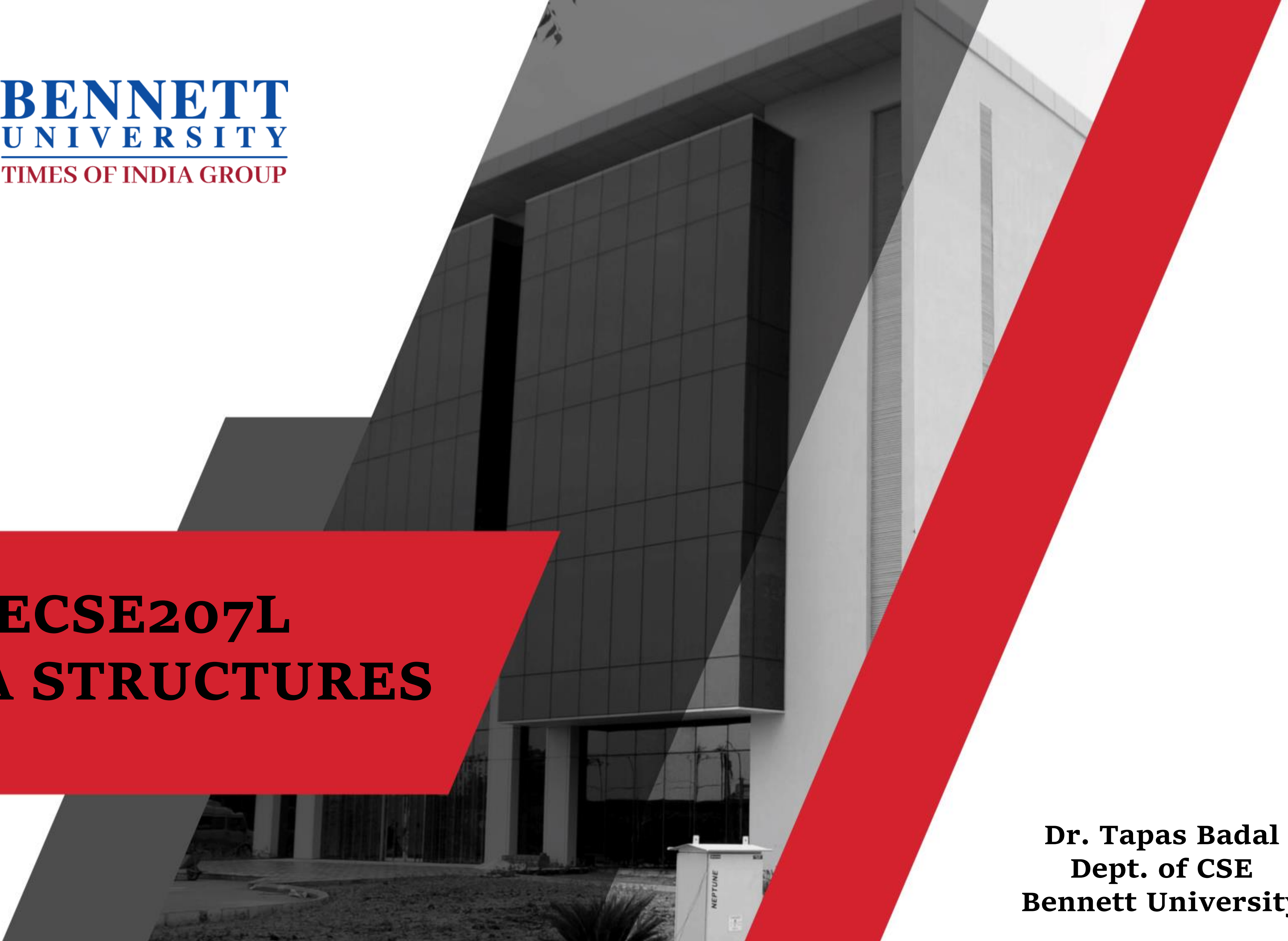
BENNETT
UNIVERSITY
TIMES OF INDIA GROUP



ECSE207L
DATA STRUCTURES

Dr. Tapas Badal
Dept. of CSE
Bennett University

Scenario 1: You are working on a word editor and wants to UNDO last ten operations.

- The operations must be stored somewhere in the system memory.
- The editor retrieves the last ten operations to perform UNDO.
- To carry out these operations, algorithm is followed.

Questions:

1. How these operations are stored in system memory?
2. Will the way of sorting the operation make any impact on the performance of UNDO operation?

Scenario 2: Faculties of CSE department are using a shared printer. They can send the request to print document at any time. Printer print these documents in the order the request are submitted.

- These print request must be stored in printer memory.
- The printer retrieve these request one by one for printing.
- To carry out these operations, algorithm is followed.

Questions:

1. How these operations are stored in printer memory?
2. Will the way of sorting these request make any impact on the performance of printing operation?

NOW

What do you think?
What is needed to solve a problem?

Is it only the selection of optimized algorithm?

NO

What else?

- ✓ The Editor scenario expects the operation(key strokes) to be stored in the memory.
- ✓ The printer scenario expects the requests from different users to be stored in memory.
- ✓ Hence the way of storing (organizing) these data is crucial for the operations to be performed.

DATA STRUCTURES: INTRODUCTION

- ✓ Data structure is an arrangement the Editor scenario expects the operation(key strokes) to be stored in the memory.
- ✓ Data structure represents the mathematical or logical model of organized data.
- ✓ For optimized handling of the organized data, knowledge of data structure is necessary.

DATA STRUCTURES: INTRODUCTION

- The task is to implement a flight trip planner.
- Flights are specified for various pairs of cities.
- You will be provided with flight number, departure time and arrival time of that flight.
- Note that there could be multiple flights between a pair of cities.
- There is some waiting time for the next flight once you reach an intermediate city.
- You need to find the best trip (taking least time) which starts at given time and given city and has to reach a specified city.

- Structured Data makes life easier.

Familiar Structures

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

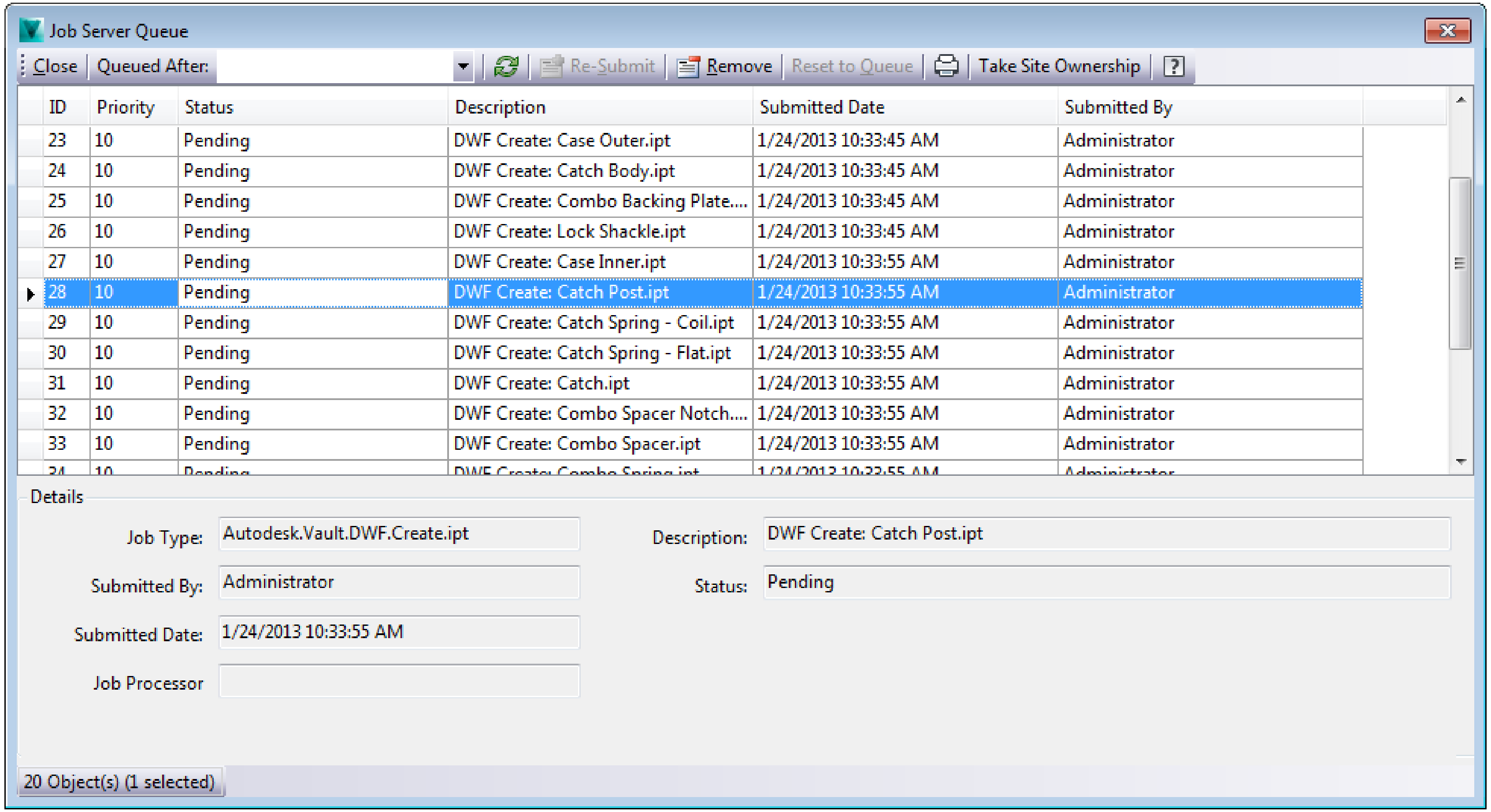
Dictionary: sorted data

plumber /ˈplʌmə(r)/ *n* [C] person whose job is to fit and repair water pipes
plumbing /ˈplʌmɪŋ/ *n* [U] **1** system of water pipes, tanks, etc in a building **2** work of a plumber
plume /pluːm/ *n* [C] **1** cloud of sth that rises into the air **2** large feather
plummet /ˈplʌmɪt/ *v* [I] fall suddenly and quickly from a high level: *House prices have ~ed.*
plump /plʌmp/ *adj* having a soft, round body; slightly fat ● **plump** *v* [T] ~ **(up)** make sth larger, softer and rounder: *~ up the pillows* [PV] **plump for sb/sth** (*infml*) choose sb/sth ▶ **plumpness** *n* [U]
plunder /ˈplʌndə(r)/ *v* [I,T] steal things from a place, esp during a war ● **plunder** *n* [U] **1** act of plundering **2** things that have been stolen, esp during a war
plunge /plʌndʒ/ *v* [I,T] (cause sb/sth to) move suddenly forwards and/or downwards: *The car ~d into the river.* ◇ *He ~d his hands into his pockets.* ● **plunge** *n* [C, usu sing] sudden movement downwards or away from sth; decrease [IDM] **take the plunge** (*infml*) finally decide to do sth important or difficult ▶ **plunger** *n* [C] part of a piece of equipment that can be pushed down
pluperfect /ˌpluːˈpɜːfɪkt/ *n* (*gram*) = THE PAST PERFECT (PAST[1])
plural /ˈplʊərəl/ *n* [usu sing] *adj*

p.m. /ˌpiː ˈem/ *abbr* after 12 o'clock noon
pneumatic /njuːˈmætɪk/ *adj* **1** filled with air: *a ~ tyre* **2** worked by air under pressure: *a ~ drill* ▶ **pneumatically** /-kli/ *adv*
pneumonia /njuːˈməʊniə/ *n* [U] serious illness affecting the lungs
PO /ˌpiː ˈəʊ/ *abbr* **1** = POST OFFICE (POST[1]) **2** = POSTAL ORDER (POSTAL) ■ **ˌP'O box** (*also* **ˈpost office box**) *n* [C] used as a kind of address, so that mail can be sent to a post office where it is kept until it is collected
poach /pəʊtʃ/ *v* **1** [T] cook fish or an egg without its shell in water that is boiling gently **2** [I,T] illegally hunt animals, birds or fish on sb else's property **3** [T] take from sb/sth dishonestly; steal sth ▶ **poacher** *n* [C] person who illegally hunts animals, birds or fish on sb else's property
pocket /ˈpɒkɪt/ *n* [C] **1** small bag sewn into a piece of clothing so that you can carry things in it **2** small bag or container fastened to sth so that you can put things in it, eg in a car door or handbag **3** [usu sing] amount of money that you have to spend: *He had no intention of paying out of his own ~.* **4** small separate group or area [IDM] **in/out of pocket** (*esp GB*) having gained/lost money as a result of sth ● **pocket** *v* [T] **1** put sth into your pocket **2** keep or take

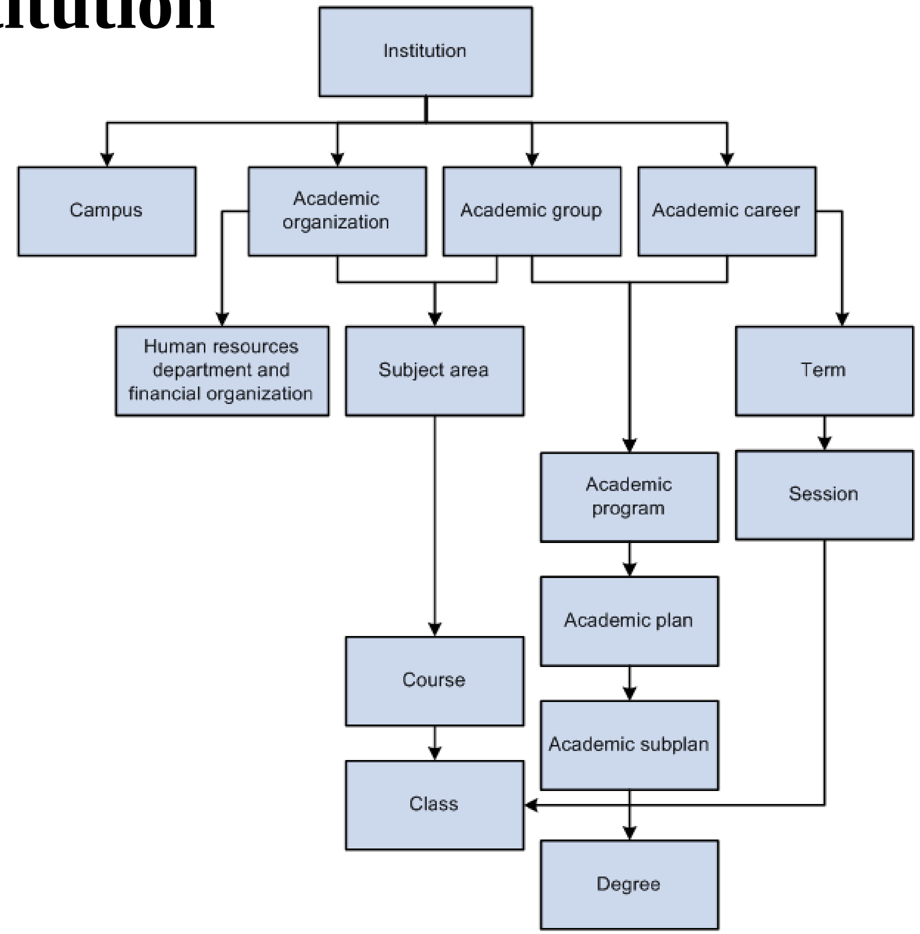
Familiar Structures

Data organized in columns



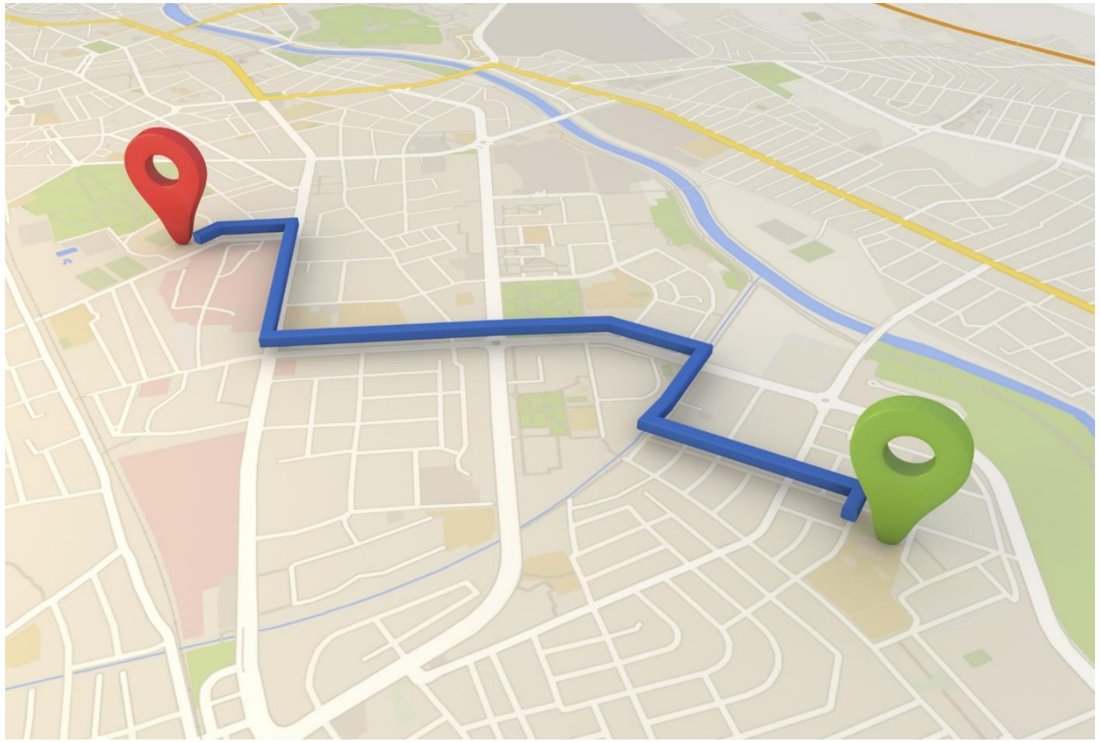
Familiar Structures

Organization structure of an institution



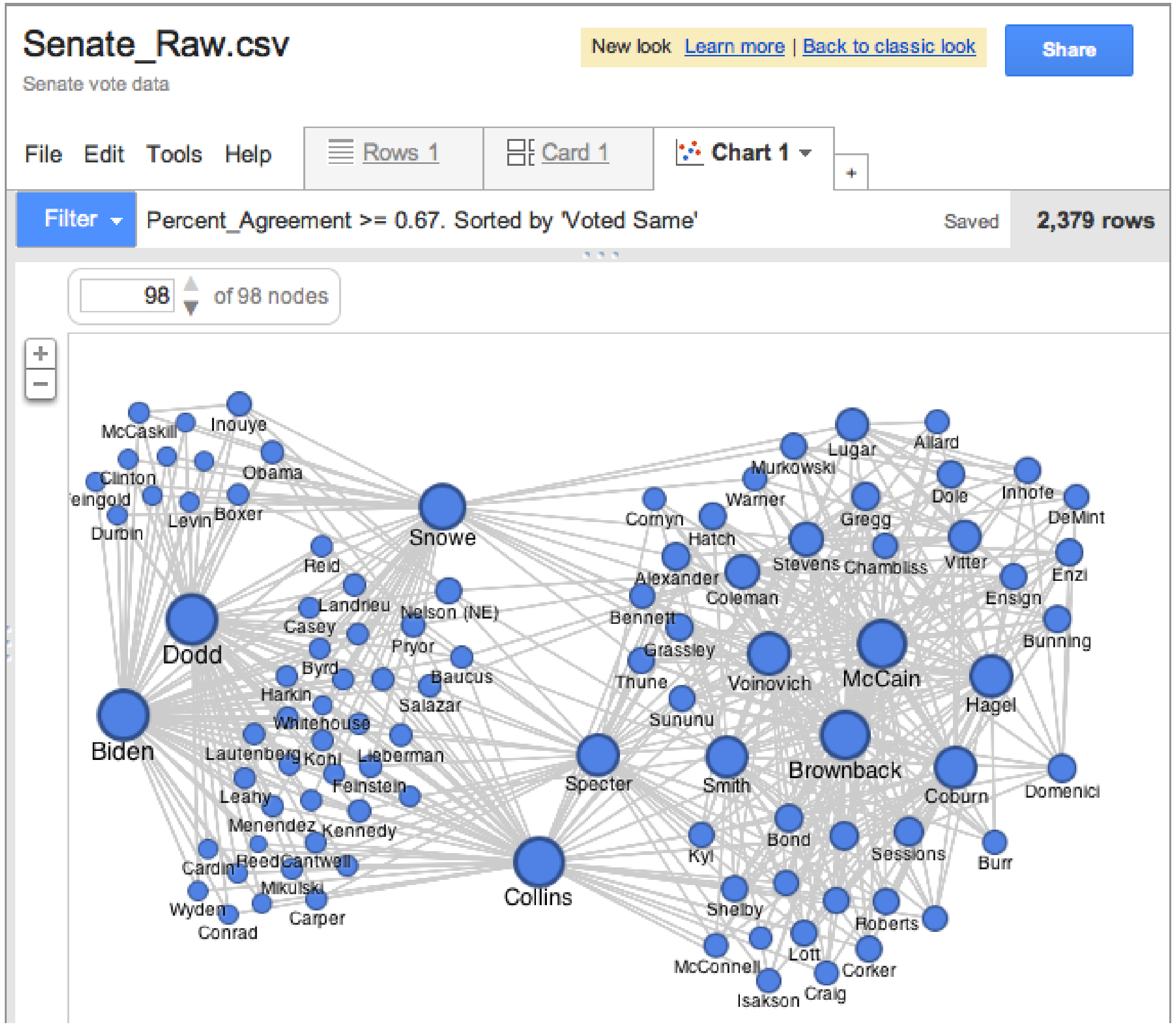
Familiar Structures

- **Road Map: geo-spatial data**
- Actual Storage: coordinates of end points of all road sections.
- calculates shortest path from A to B.



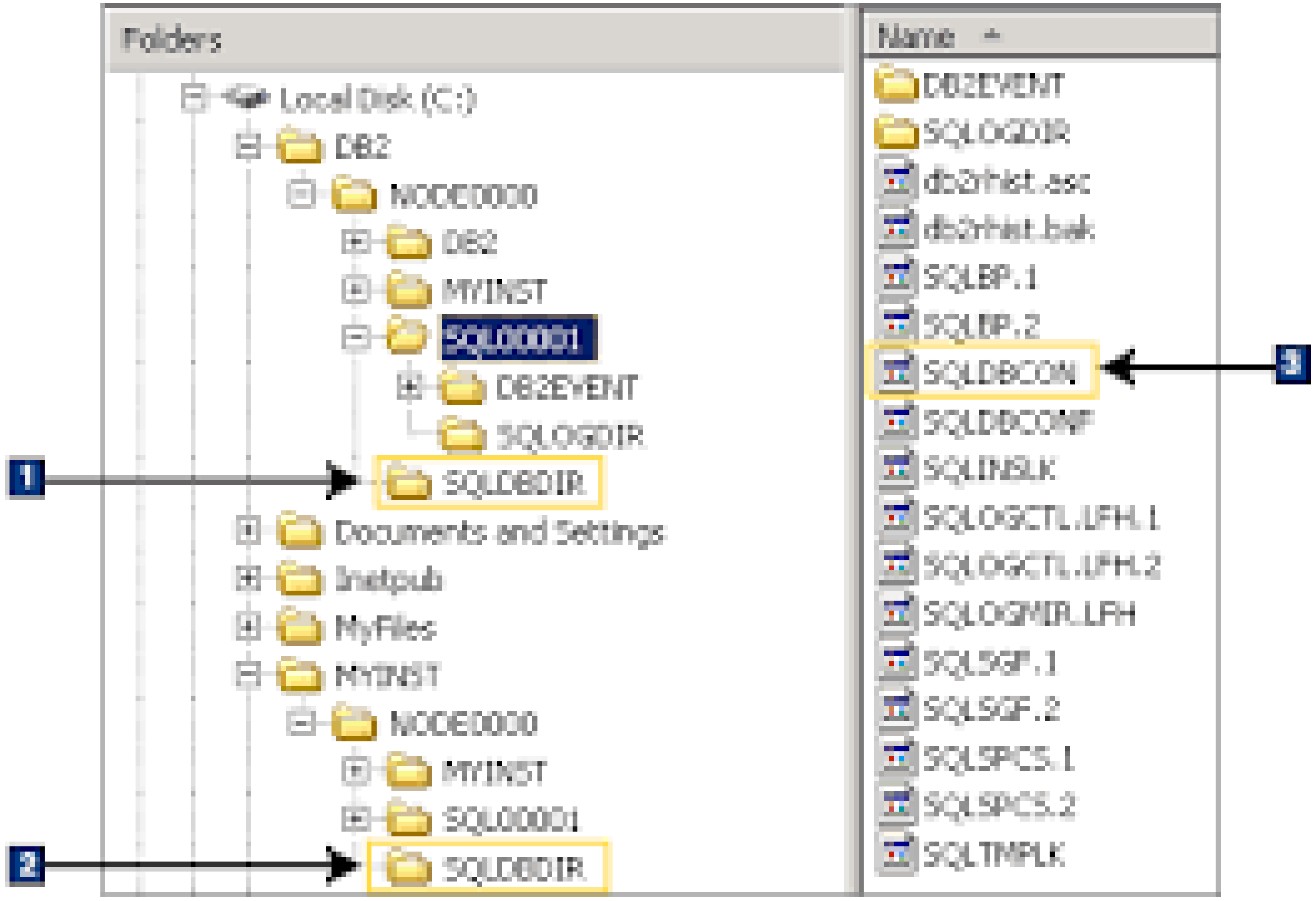
Familiar Structures

Social network



Familiar Structures

Hard disk directory



Why Data Structures?

- Data is just the raw material for information, analytics, business intelligence, advertising, etc.

- We need Computational efficient ways of analyzing, storing, searching, modeling data.

- Smart data structures offer an intelligent tradeoff.

What is this Course About?

- Clever ways to organize information in order to enable efficient computation

  - What do we mean by clever?
    - Given flight timings between various cities, work out the optimal (best path) route to reach a destination starting at a specific time from source.

  - What do we mean by efficient?
    - Efficient algorithm to solve the above problem in best time

DATA STRUCTURES:

- ✓ Can we conclude that
- ✓ **Program = Data Structure + Algorithm** operating on data?
- ✓ Yes
- ✓ For writing a program both algorithm and data structure should be considered.

DATA STRUCTURES:

Lists, Stacks, Queues
Heaps
Binary Search Trees
AVL Trees
Hash Tables
Graphs
Disjoint Sets

Data Structures

Insert
Delete
Find
Merge
Shortest Paths
Union

Algorithms

Analysis of Algorithms

Time and space

- To analyze an algorithm means:
  - developing a formula for predicting *how fast* an algorithm is, based on the size of the input (time complexity)
  - developing a formula for predicting *how much memory* an algorithm requires, based on the size of the input (space complexity)
- Usually time is our biggest concern
  - Most algorithms require a fixed amount of space

- In a higher-level language (such as Java), we *do not know* how long each operation takes
  - Which is faster, `x < 10` or `x <= 9` ?
  - We don’t know exactly what the compiler does with this.
  - The compiler probably optimizes the test anyway (replacing the slower version with the faster one).
- In a higher-level language we *cannot* do an exact analysis
  - Our timing analyses will use *major* oversimplifications.
  - Nevertheless, we can get some very useful results.

Algorithmic Analysis

- A technique used to characterize the execution behavior of algorithms in a manner *independent* of a particular platform, compiler, or language.

- Abstract away the minor variations and describe the performance of algorithms in a more theoretical, *processor independent* fashion.

- A method to compare speed of algorithms against one another depending on size of the input.

What does “size of the input” mean?

- If we are searching an array, the “size” of the input could be the size of the array.
- If we are merging two arrays, the “size” could be the sum of the two array sizes.
- If we are computing the n^{th} Fibonacci number, or the n^{th} factorial, the “size” is n.
- We choose the “size” to be the parameter that most influences the actual time/space required
  - It is *usually* obvious what this parameter is.
  - Sometimes we need two or more parameters.

Operations to count

- In computing time complexity, one good approach is to count characteristic operations
  - What a "characteristic operation" is depends on the particular problem
  - If searching, it might be comparing two values
  - If sorting an array, it might be:
    - comparing two values
    - swapping the contents of two array locations
    - both of the above
  - Sometimes we just look at how many times the *innermost loop* is executed

Constant time complexity

- *Constant time* means there is some constant k such that this operation always takes k nanoseconds.
- A Java statement takes constant time if:
  - It does not include a loop.
  - It does not include calling a method whose time is unknown or is not a constant.
- If a statement involves a choice (if or switch) among operations, each of which takes constant time, we consider the statement to take constant time.

Big O -- Big Oh

- The most common method and notation for discussing the execution time of algorithms is "Big O”.
- Operations involving constant time are said to require order 1 steps, that is O(1) steps.
- For an alphabetized dictionary the algorithm requires O(log N) steps to search for an item.
- For an unsorted list, the linear search algorithm requires O(N) steps.
- Big O is the *asymptotic execution time* of the algorithm.

Loops That Work on a Data Set

- Normally a loop operates on a data set which can vary in size.

```
public double minimum(double[] values, int n)
{
  double minValue = values[0];
  for(int i = 1; i < n; i++)
    if(values[i] < minValue)
      minValue = values[i];
  return minValue;
}
```

- The number of executions of the loop depends on the number of elements in the array .
- The run time is O(n).

Linear Search

```
static boolean member(int x, int[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        if (x == a[i]) return true;
    }   return false;
}
```

- If x is *not* in a, the loop executes n times, where n is number of elements in the array.
  - This is the worst case
- If x *is* in a, the loop executes n/2 times *on average.*
- Either way, it is order of n, O(n).

Linear time algorithm

```
for (i = 0, j = 1;    i < n;     i++) {
        j = j * i;
   }
```

- This loop takes time $k*n + c$, for some constants k and c

  k : How long it takes to go through the loop once
  (the time for $j = j * i$, plus loop overhead)

  n : The number of times through the loop
  (we can use this as the "size" of the problem)

  c : The time it takes to initialize the loop

- The total time $k*n + c$ is *linear in n*
- Execution time *O(n).*

Constant time is (usually) better than linear time

- Suppose we have two algorithms to solve a task:
  - Algorithm A takes 5000 time units
  - Algorithm B takes 100*n time units
- Which is better?
  - Clearly, algorithm B is better if our problem size is small, that is, if $n < 50$
  - Algorithm A is better for larger problems, with $n > 50$
  - So B is better on small problems that are quick anyway
  - But A is better for large problems, *where it matters more*
- *We usually care most about very large problems*

Nested Loops

```
public void bubbleSort(double[] data, int n)
{
   for(int i = n - 1; i > 0; i--)
      for(int j = 0; j < i; j++)
            if(data[j] > data[j+1])
         {   double temp = data[j];
                  data[j] = data[j + 1];
                  data[j + 1] = temp;
          }
}
```

Number of executions?

The array subset problem

- Suppose you have two sets, represented as unsorted arrays:
  - int[] sub = { 7, 1, 3, 2, 5 };
    int[] super = { 8, 4, 7, 1, 2, 3, 9 };
- and you want to test whether every element of the first set (sub) also occurs in the second set (super)
- If there are m elements in first set, and n elements in the second set, an algorithm will select one element from first set and go through n elements of the second set.
- Thus for m elements of first set, total number of operations are going to be order of m*n .
- We can say that the array subset problem has time complexity of O(mn), along with assorted constants

The array subset problem

- Suppose you have two sets, represented as unsorted arrays:
  - int[] sub = { 7, 1, 3, 2, 5 };
    int[] super = { 8, 4, 7, 1, 2, 3, 9 };
- and you want to test whether every element of the first set (sub) also occurs in the second set (super)
- If there are m elements in first set, and n elements in the second set, an algorithm will select one element from first set and go through n elements of the second set.
- Thus for m elements of first set, total number of operations are going to be order of m*n .
- We can say that the array subset problem has time complexity of O(mn), along with assorted constants
  - If m and n are similar in value, this is roughly quadratic time complexity

What about the constants?

- Forget the constants!
- An added constant, $f(n)+c$, becomes less and less important as n gets larger
- Suppose an algorithm takes $12n^3+4n^2+15$ steps and another takes $24n^2+8n+35$ how do we compare the two algorithms?
- We need to simplify the formulae for a quick comparison.

Simplifying the formula

- Consider a jungle with number of animals in it.
- Let there be n elephants, m tigers, p foxes, k squirrels, and g ants in the jungle. Is there a simple formula to represent the net weight of the animals?
- Consider 3 elephants, 5 tigers, 10 foxes, 200 squirrels, and 10,000 ants. The net weight is going to be governed mainly by the weight of the elephants.
- In another jungle with no elephants, the weight of the tigers would be most prominent.
- Thus it makes sense to keep only the highest order terms of the formula for calculating time complexity.

Simplifying the formulae

- Throwing out the constants is one of *two* things we do in analysis of algorithms
  - By throwing out constants, we simplify $12n^2 + 35$ to just n^2
- Our timing formula is a polynomial, and may have terms of various orders (constant, linear, quadratic, cubic, etc.)
  - We usually discard all but the *highest-order* term
    - We simplify $n^2 + 3n + 5$ to just n^2

Big O notation

- When we have a polynomial that describes the time requirements of an algorithm, we simplify it by:
  - Throwing out all but the highest-order term
  - Throwing out all the constants

- If an algorithm takes $12n^3+4n^2+8n+35$ time, we simplify this formula to just n^3

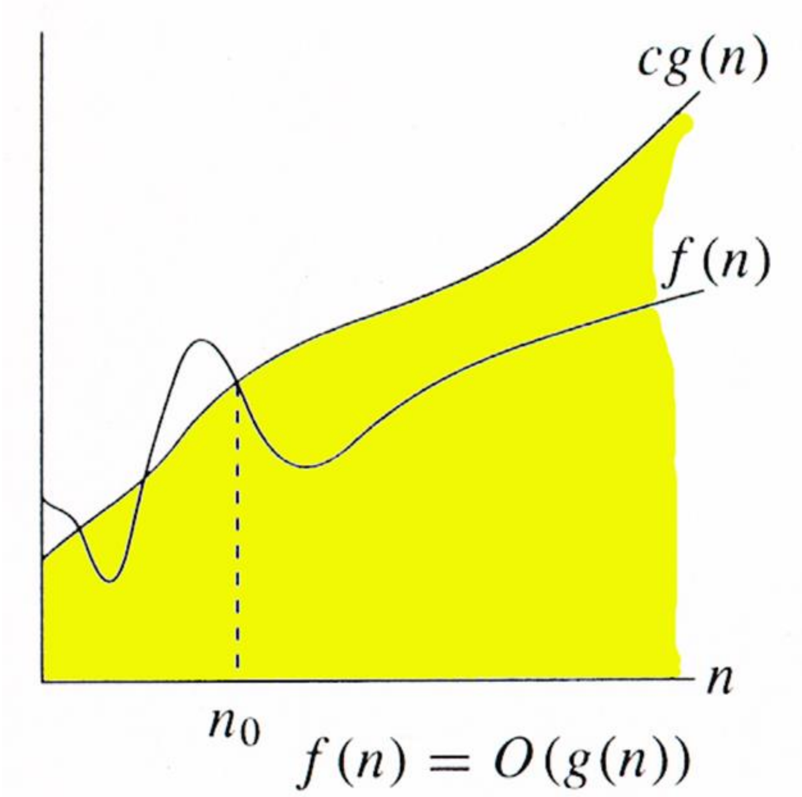
- We say the algorithm requires $O(n^3)$ time
  - We call this Big O notation

Formal Definition of Big O-notation

- Let f(n) denote the expression denoting the number of operations for an algorithm.
- If there are positive constants c and n_0,
- such that for all $n \geq n_0$,
- $f(n) \leq cg(n)$
- then we say that $f(n)$ is of order $O(g(n))$

- This covers set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.

Big O-notation



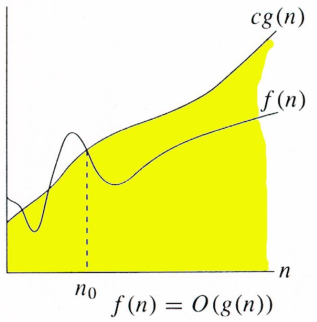
c g*(*n*)** **is an** ***asymptotic upper bound **for *f*(*n*).**

More on Big - O

- There is a point n_0 such that for all values of n that are past this point, f(n) is bounded by some multiple of g(n).

- Thus if f(n) of the algorithm is O(n^2) then, ignoring constants, at some point we can *bound* the running time by a quadratic function of the input size.

- Given a *linear* algorithm, it is *technically correct* to say the running time is O(n^2). O(n) is a more precise answer as to the Big O bound of a linear algorithm.



Can we justify Big O notation?

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

- Big O notation is a *huge* simplification; can we justify it?
  - It only makes sense for *large* problem sizes
  - **For sufficiently large problem sizes, the highest-order term swamps all the rest!**
- Consider $R = x^2 + 3x + 5$ as x varies:

$x = 0$	$x^2 = 0$	$3n = 0$	$5 = 5$	$R = 5$
$x = 10$	$x^2 = 100$	$3x = 30$	$5 = 5$	$R = 135$
$x = 100$	$x^2 = 10000$	$3x = 300$	$5 = 5$	$R = 10{,}305$
$x = 1000$	$x^2 = 1000000$	$3x = 3000$	$5 = 5$	$R = 1{,}003{,}005$
$x = 10{,}000$	$x^2 = 10^8$	$3x = 3*10^4$	$5 = 5$	$R = 100{,}030{,}005$
$x = 100{,}000$	$x^2 = 10^{10}$	$3x = 3*10^5$	$5 = 5$	$R = 10{,}000{,}300{,}005$

Big O Examples

- $3n^3 = O(n^3)$
- $3n^3 + 8 = O(n^3)$
- $8n^2 + 10n * \log(n) + 100n + 10^{20} = O(n^2)$
- $3 \log(n) + 2n^{1/2} = O(n^{1/2})$
- $2^5 = O(1)$
- $T_{linearSearch}(n) = O(n)$
- $T_{binarySearch}(n) = O(\log(n))$

Common time complexities

BETTER

• $O(1)$	constant time
• $O(\log n)$	log time
• $O(n)$	linear time
• $O(n \log n)$	log linear time
• $O(n^2)$	quadratic time
• $O(n^3)$	cubic time
• $O(2^n)$	exponential time

WORSE

Ranking of Algorithmic Behaviors

Function	Common Name
$N!$	factorial
2^N	Exponential
N^d, $d > 3$	Polynomial
N^3	Cubic
N^2	Quadratic
$N\sqrt{N}$	
$N \log N$	
N	Linear
$\sqrt{N}$	Root - n
$\log N$	Logarithmic
1	Constant

Running Times

Assume N = 100,000 and a computer with processor speed of 1,000,000 operations per second.

Function	Running Time
2^N	over 100 years
N^3	31.7 years
N^2	2.8 hours
N N	31.6 seconds
N log N	1.2 seconds
N	0.1 seconds
N	3.2×10^{-4} seconds
log N	1.2×10^{-5} seconds

THANKYOU

@csebennett cse_bennett