Every computer program uses **data structures and algorithms**

**Data Structures**
- **Why?**: organize your data in computer's memory to efficiently store and retrieve **information**
- **How?**: using arrays, linked lists, stacks, queues, trees, matrix, etc.

**Algorithms**
- **Why?**: manipulate the data in various way to perform **computation**
- **How?**: using strategy/method, operations and analysis → Importance of Big O notation

**Data-centric view of the world**
- Complex data structures (from arrays to trees)
- Basic 'non-numerical' algorithms act on database (insert, remove, search, sort, traverse)

**Scientific Computing**
- Basic data structure
- Complex numerical algorithms (computational modeling, HPC)
Data Structures & Algorithms: The big picture

**DATA STRUCTURES**

- **Array**
  - (1d, 2d, etc.)
- **Linked List**
  - (simple, doubly)
- **Unsorted list**
- **Sorted list**
- **Trees**
  - (BST, CBT, RBT)
- **Hash-Tables**
- **Stacks**
- **Queues**
- **Heap**
  - (Priority Q)
- **Graphs**

**ALGORITHMS**

- Insert
- Remove
- Search
- Sort
- Traverse
- Pop, Push, dequeue, enqueue, etc.
General Purpose Data Structures

- **Motivations**
  - Store real-world data: personal records, inventories, media library, etc.

- Some comparisons between *general purpose* data structures

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Search</th>
<th>Insertion</th>
<th>Deletion</th>
<th>Traversal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>—</td>
</tr>
<tr>
<td>Ordered array</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Linked list</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>—</td>
</tr>
<tr>
<td>Ordered linked list</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Binary tree (average)</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Binary tree (worst case)</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Balanced tree (average and worst case)</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Hash table</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>—</td>
</tr>
</tbody>
</table>
General Purpose Data Structures

- Tentative 'Guideline' on When to use What?

  - **Array**
    - Easy to implement, data locality
    - Static size
  - **Linked-list**
    - Still easy to implement
    - Dynamic size
  - **BST**
    - Balanced (difficult to implement)
    - Unbalanced (great if data inserted in random order)
  - **Hast-Tables**
    - Optimal for search, insert
    - No Traversal
    - Array-based (static size)
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Time Complexity</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bubble Sort</strong></td>
<td>O(N^2)</td>
<td>Slow, slow, slow</td>
</tr>
<tr>
<td></td>
<td>O(N^2)</td>
<td>comp.+ O(N^2) swaps</td>
</tr>
<tr>
<td><strong>Selection Sort</strong></td>
<td>O(N^2)</td>
<td>Intuitive but still slow</td>
</tr>
<tr>
<td></td>
<td>O(N^2)</td>
<td>comp.+ O(N) swaps</td>
</tr>
<tr>
<td><strong>Insertion Sort</strong></td>
<td>O(N^2)</td>
<td>half #comp. than Bubble</td>
</tr>
<tr>
<td></td>
<td>O(N^2)</td>
<td>comp.+ O(N^2) copies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O(N) total if data almost sorted</td>
</tr>
<tr>
<td><strong>Enhanced Insertion Sort</strong></td>
<td>O(N^2)</td>
<td>Use binary search rather than linear search</td>
</tr>
<tr>
<td></td>
<td>O(NlogN)</td>
<td>comp.+ O(N^2) copies</td>
</tr>
<tr>
<td><strong>List Insertion Sort</strong></td>
<td>O(N^2)</td>
<td>Only 2N copies</td>
</tr>
<tr>
<td></td>
<td>O(N^2)</td>
<td>Does not work 'in-place'</td>
</tr>
<tr>
<td></td>
<td>comp.+ O(N) copies</td>
<td></td>
</tr>
<tr>
<td><strong>MergeSort</strong></td>
<td>O(NlogN)</td>
<td>Divide &amp; Conquer + Recursive</td>
</tr>
<tr>
<td></td>
<td>O(N)</td>
<td>Does not work 'in place'</td>
</tr>
<tr>
<td></td>
<td>copies by O(logN) levels</td>
<td></td>
</tr>
<tr>
<td><strong>ShellSort</strong></td>
<td>O(N(logN)^2)</td>
<td>'Insertion sort' using increment</td>
</tr>
<tr>
<td></td>
<td>In average</td>
<td>Worst case not far from average</td>
</tr>
<tr>
<td><strong>QuickSort</strong></td>
<td>O(NlogN)</td>
<td>Divide and Conquer</td>
</tr>
<tr>
<td></td>
<td>Comp.&gt;swaps</td>
<td>Uses partitioning recursively</td>
</tr>
<tr>
<td><strong>HeapSort</strong></td>
<td>O(NlogN)</td>
<td>Require a heap data-structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No worst case</td>
</tr>
</tbody>
</table>
Specialized Purpose Data Structures

- **Motivations**
  - Used as programming tools (conceptual aids)
- **ADT**
  - Stacks: LIFO
    - Array or Linked-list
    - Insert/delete in O(1)
  - Queue: FIFO
    - Array or Linked-List
    - Insert/delete in O(1)
  - Priority Queue: item with highest priority
    - Using Ordered array: O(1) deletion, O(N) insertion
    - Using Heap: O(logN) for both deletion/insertion
Specialized Purpose Data Structures: Graphs

- **Motivations**
  - Do not store general-purpose data
  - Use in Modeling
    - Model real-world situations
    - Tool for numerical modeling in scientific computing (via matrix representation)

- **Graph ↔ Matrix**

- Algorithms we have seen:
  - Search: DFS, BFS
  - Minimum Spanning Trees (MST)
  - Shortest Path Problem (Dijkstra)

- Efficiency of Algorithms depends on Dense vs Sparse matrix storage
Further Reading

Your Textbook ;-)  

R. Sedgewick textbook

Interview preparation (example G. Laakmann)
Parallelism: A change of scenery

The end of free speed-up...
Challenges for traditional algorithms
Logistic: next Week

- **Monday:**
  - Short intro to scientific computing, parallelism
- **Wednesday**
  - No class
- **Thursday**
  - TA will discuss solutions of projects 5, 6
  - HW reviews
- **Friday**
  - Final review session
- **Final: Tuesday December 15**
  - 7:45am-10am
  - **Room:** may change