An algorithm (pronounced AL-go-rith-um) is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850.”
Reminders

- HW1 due today
- Please take the survey if you haven’t already.
  - https://usc.qualtrics.com/jfe/form/SV_9ZRBPmDUirKGt2B
<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Assigned</th>
<th>Due</th>
<th>Quizzes/Midterm/Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-Jan</td>
<td>Introduction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14-Jan</td>
<td>Computer architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21-Jan</td>
<td>MLK Holiday</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28-Jan</td>
<td>Data structures</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29-Jan</td>
<td>Last day to drop a Monday-only class without a mark of “W” and receive a refund or change to Pass/No Pass or Audit for Session 001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-Feb</td>
<td>Data structures</td>
<td>Trees, Graphs and Traversals</td>
<td>HW2</td>
<td></td>
</tr>
<tr>
<td>11-Feb</td>
<td>More Algorithms/Data Structures, Complexity and Combinatorics</td>
<td>Recursion and run-time</td>
<td></td>
<td>Quiz 2 on material taught in class 1/28, 2/4</td>
</tr>
<tr>
<td>18-Feb</td>
<td>Presidents Day</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22-Feb</td>
<td>Last day to drop a course without a mark of “W” on the transcript</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25-Feb</td>
<td>Algorithms and programming</td>
<td>(Somewhat) More complicated algorithms and simple programming constructs, compilers</td>
<td>HW2</td>
<td>Quiz 3 on material taught in class 2/11</td>
</tr>
<tr>
<td>4-Mar</td>
<td>Midterm</td>
<td></td>
<td></td>
<td>Midterm on all material taught so far</td>
</tr>
<tr>
<td>11-Mar</td>
<td>Spring Break</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18-Mar</td>
<td>Operating systems</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25-Mar</td>
<td>Computer networks</td>
<td></td>
<td></td>
<td>Quiz 4 on material taught in class 3/18</td>
</tr>
<tr>
<td>1-Apr</td>
<td>Artificial intelligence</td>
<td></td>
<td></td>
<td>Quiz 5 on material taught in class 3/25</td>
</tr>
<tr>
<td>5-Apr</td>
<td>Last day to drop a class with a mark of “W” for Session 001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-Apr</td>
<td>The limits of computation</td>
<td></td>
<td></td>
<td>Quiz 6 on material taught in class 4/1</td>
</tr>
<tr>
<td>15-Apr</td>
<td>Robotics</td>
<td></td>
<td></td>
<td>Quiz 7 on material taught in class 4/8</td>
</tr>
<tr>
<td>22-Apr</td>
<td>Summary, recap, review</td>
<td></td>
<td></td>
<td>Quiz 8 on material taught in class 4/15</td>
</tr>
<tr>
<td>3-May</td>
<td>Final exam 11 am - 1 pm in SAL 101</td>
<td></td>
<td></td>
<td>Final on all material covered in the semester</td>
</tr>
</tbody>
</table>
A problem-solving view of computers and computing
Organizing information: sequences and trees
Organizing information: graphs
Abstract data types: recursion

Reading: St. Amant Ch. 4
◆ “The architecture level gives us a very detailed view of what happens on a computer. But trying to understand everything a computer does at this level would be...(insert analogy about perspective). If all we can see is fine detail, it can be hard to grasp what’s happening on a larger scale.”
“Here’s a different perspective: computers solve problems. Solving problems, in contrast to executing instructions, means not having to worry about all the details at once. Instead, we can think in more abstract terms. How should we represent a problem? Can we break a problem down into smaller pieces so that it’s easier to solve? What would a solution procedure look like, in the abstract?”
"Answering these questions is a matter of representation. We’ve already seen representation, in the encoding of data and instructions in a form that’s convenient for a computer. Now we need to think more generally about how to represent problems and their solutions.” – st. Amant pg. 52
When thinking about solving problems with computers (somewhat due to the nature of computers), three abstract data types are essential:

- Sequences
- Trees
- Graphs

Part of the course is essentially an extended vocabulary lesson

- So you’re prepared to understand and learn these topics in detail in other courses
Architecture puts the computer under the microscope
  - Imagine solving *all* problems by thinking about the computer at the architecture level

Early computer scientists *had* to do this
  - Luckily we don’t.
Problem Solving

- Computers are used to solve problems
- Abstraction for problems
  - How to represent a problem?
  - How to break down a problem into smaller parts?
  - What does a solution look like?
- Two key building blocks
  - Abstract data types
  - Algorithms
Algorithm: a step by step description of actions to solve a problem

- Typically at an abstract level
- Analogy: clearly written recipe for preparing a meal
- More in the next few lectures

“Algorithms are models of procedures at an abstract level we decided is appropriate.” [St. Amant, pp. 53]
Abstract Data Types

- Models of collections of information
  - Chosen to help solve a problem

- Typically at an abstract level
  - Don’t deal with implementation details: memory layout, pointers, etc.

“... describes what can be done with a collection of information, without going down to the level of computer storage.” [St. Amant, pp. 53]
**Sequence: a list**
- Items are called elements
- Item number is called the index

**Tree**

**Graph**
Motivation for Abstract Data Structures

- The nature of some data, and the way we need to accesses it often requires some structure, or organization to make things efficient (or even possible)
- Data: large set of names (maybe attendance data)
- Problems: did Jelena attend on 9/10? How many lectures did Mario attend? Which students didn’t attend 8/27?
Is ‘Jelena’ on this list?

- Byron
- Therese
- Alpha
- Christopher
- Jacquelyn
- Amada
- Araceli
- Deanna
- Mario
- Pamela
- Lin
- Hester

- Lenora
- Staci
- Emma
- Elsa
- Derrick
- Kelley
- Kathe
- Mohammad
- Julia
- Renda
- Kylee
- Keren

- Jayna
- Joy
- Sean
- Basilia
- Cassie
- Sharice
- Carina
- Liv
- Clara
- Bess
- Simone
- Michiko
- Elmer

- Jayna
- Jesusa
- Dion
- Orpha
- Denice
- Tad
- Geraldine
- Bradley
- Mariah
- Lyndsey
- Marcia
- Beatrice
- Keri
- Thu
Option #1 No Data Structure

- Store names in the computer with no organization
- Scan all of them every time a question is asked
Is ‘Lilly’ on this list?

- Allene
- Berenice
- Bernadine
- Candelaria
- Carli
- Carry
- Chau
- Cynthia
- Clement
- Davina
- Exie
- Ezequiel
- Filiberto
- Francisca
- Fred
- Gayle
- Gudrun
- Huey
- Isaiah
- Janey
- Jen
- Joanne
- Joanie
- Laney
- Lenora
- Lilliam
- Lilly
- Lina
- Lorinda
- Lulu
- Michelle
- Madelaine
- Marielle
- Mauro
- Mayola
- Mikaela
- Pamala
- Pinkie
- Princess
- Rocco
- Rosanne
- Sally
- Season
- Sidney
- Tamica
- Tilda
- Val
- Vinita
- Yaeko
- Yoshiko
Store names in sorted order
This implies **structure** to the data
Also, if the names start out un-sorted, how do we get to sorted state?
Sequences aka Lists

◆ Sequences are our first fundamental data structure
◆ Sequences hold items
  ❖ Items = whatever we need. It’s abstract.
◆ Sequences have the notion of order
  ❖ Items come one after another
◆ Sequences can be accessed by index, or relative
  ❖ Find the 5th item
  ❖ Or move to next or previous from current item
◆ The “how” (implementation) is abstract
  ❖ Arrays (C, C++), Vectors (C++), ArrayList (Java), Lists (Python)...
  ❖ These are all different implementations of this abstract data structure
Most “questions” (problems) that are solved using sequences are essentially one of two questions:

- Is item A in sequence X?
- Where in sequence Y is item B?
- Both of these are answered by searching the sequence
Sequences: Searching

- Sequential search: start at 1, proceed to next location...
- If names in the list are sorted (say in alphabetical order), then how to proceed?
  - Start in the ‘middle’
  - Decide if the name you’re looking for is in the first half or second
  - ‘Zoom in’ to the correct half
  - Start in the ‘middle’
  - Decide if the name you’re looking for is in the first half or second
  - ‘Zoom in’ to the correct half
  - ...
- Which is more efficient (under what conditions)?

brute force
divide-and-conquer
If searching a sorted sequence is more efficient (per search), this implies we need a way to sort a sequence!

Sorting algorithms are fundamental to CS
- Used A LOT to teach various CS and programming concepts

Computer Scientists like coming up with better more efficient ways to sort data
- Even have contests!

We’ll look at two algorithms with very different designs
- Selection Sort
- Quick Sort
Sorting: Selection Sort

- Sorting: putting a set of items in order
- Simplest way: selection sort
  - March down the list starting at the beginning and find the smallest number
  - Exchange the smallest number with the number at location 1
  - March down the list starting at the second location and find the smallest number (overall second-smallest number)
  - Exchange the smallest number with the number at location 2
  - ...

## Sorting: Selection Sort

<table>
<thead>
<tr>
<th>Original List</th>
<th>Sorted List</th>
<th>Processed List</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 4 3 5 12 6 20 10</td>
<td>3 6 7 9 10 20 1</td>
<td>13 4 3 5 12 6 20 10</td>
</tr>
<tr>
<td>3 4 13 5 12 6 20 10</td>
<td>1 6 7 9 10 20 3</td>
<td>3 4 13 5 12 6 20 10</td>
</tr>
<tr>
<td>3 4 13 5 12 6 20 10</td>
<td>1 3 7 9 10 20 6</td>
<td>3 4 13 5 12 6 20 10</td>
</tr>
<tr>
<td>3 4 5 13 12 6 20 10</td>
<td>1 3 6 9 10 20 7</td>
<td>3 4 5 13 12 6 20 10</td>
</tr>
<tr>
<td>3 4 5 6 12 13 20 10</td>
<td>1 3 6 7 10 20 9</td>
<td>3 4 5 6 12 13 20 10</td>
</tr>
<tr>
<td>3 4 5 6 10 13 20 12</td>
<td>1 3 6 7 9 20 10</td>
<td>3 4 5 6 10 13 20 12</td>
</tr>
<tr>
<td>3 4 5 6 10 12 20 13</td>
<td>1 3 6 7 9 10 20</td>
<td>3 4 5 6 10 12 20 13</td>
</tr>
<tr>
<td>3 4 5 6 10 12 13 20</td>
<td>1 3 6 7 9 10 20</td>
<td>3 4 5 6 10 12 13 20</td>
</tr>
</tbody>
</table>
Sorting: Selection Sort

- Sorting: putting a set of items in order
- Simplest way: selection sort
  - March down the list starting at the beginning and find the smallest number
  - Exchange the smallest number with the number at location 1
  - March down the list starting at the second location and find the smallest number (overall second-smallest number)
  - Exchange the smallest number with the number at location 2
  - ...

- How long does this take? Can we do it faster?
- Yes, use divide-and-conquer
How long does it take?

- We just asked an interesting question, did you notice?
- ”How long does it take?”
- This question might (should) bother some of you.
  - Why?
How long does it take?

- WTH are we even asking here?
- We’re working with an “abstract data type”
- What does ”time” even mean?
- We need to abstract time as well!
- Given some data “how long does it take” = how much “work” do we do

- “Work”
  - Operations like moving an item, copying and item, comparing two items
  - Abstract steps required

- We’ll spend a lot more time discussing this over the next few lectures
- Pick a ‘middle’ element in the sequence (this is called the pivot)
- Put all elements smaller than the pivot on its left
- Put all elements larger than the pivot on the right
- Now you have two smaller sorting problems because you have an unsorted list to the left of the pivot and an unsorted list to the right of the pivot
- Sort the sequence on the left (use Quicksort!)
- Sort the sequence on the right (use Quicksort!)
Sorting: Quicksort

- Pick a ‘middle’ element in the sequence (this is called the pivot)
- Put all elements smaller than the pivot on its left
- Put all elements larger than the pivot on the right
- Now you have two smaller sorting problems because you have an unsorted list to the left of the pivot and an unsorted list to the right of the pivot

- Sort the sequence on the left (use Quicksort!)
  - Pick a ‘middle’ element in the sequence (this is called the pivot)
  - Put all elements smaller than the pivot on its left
  - Put all elements larger than the pivot on the right
  - Now you have two smaller sorting problems because you have an unsorted list to the left of the pivot and an unsorted list to the right of the pivot
  - Sort the sequence on the left (use Quicksort!)
  - Sort the sequence on the right (use Quicksort!)

- Sort the sequence on the right (use Quicksort!)
  - Pick a ‘middle’ element in the sequence (this is called the pivot)
  - Put all elements smaller than the pivot on its left
  - Put all elements larger than the pivot on the right
  - Now you have two smaller sorting problems because you have an unsorted list to the left of the pivot and an unsorted list to the right of the pivot
  - Sort the sequence on the left (use Quicksort!)
  - Sort the sequence on the right (use Quicksort!)
This is an unsorted list (e.g., a list of numbers not in order)
Choose a pivot and put all elements smaller than the pivot to the left of the pivot and all elements larger than the pivot to its right.
Quicksort

This is an unsorted list of all elements smaller than the pivot

This is an unsorted list of all elements larger than the pivot

Pivot
Quicksort
Quicksort

Left ‘list’ (1)  Right list (3)  Left list (2)  Right ‘list’ (1)
Quicksort
Quicksort
Quicksort
13 4 3 5 12 6 20 10
Pivot = 6
4 3 5 6 13 12 20 10
Quicksort (4 3 5)
Quicksort (13 12 20 10)
13 4 3 5 12 6 20 10

Pivot = 6

4 3 5 6 13 12 20 10

Quicksort (4 3 5) 6 Quicksort (13 12 20 10)

Pivot = 4

3 4 5 6 10 12 13 20

Quicksort(3) 4 Quicksort(5) 6 Quicksort(10) 12 Quicksort(13 20)

3 4 5 6 10 12 13 20
Sorting: Quicksort

13 4 3 5 12 6 20 10

Pivot = 12

4 3 5 6 10 12 13 20

Quicksort (4 3 5 6 10) 12 Quicksort (13 20)

Pivot = 4

3 4 5 6 10 12 13 20

Quicksort (3) 4 Quicksort (5 6 10) 12 13 20

3 4 Pivot = 6 12 13 20

3 4 Quicksort (5) 6 Quicksort (10) 12 13 20

3 4 5 6 10 12 13 20
Sorting: Quicksort

- If list is size 1, return the list
- If list is size 2, and out of order, swap elements and return the swapped elements, else return the list
- Pick an element in the sequence (called the pivot)
- Put all elements smaller than the pivot on its left
- Put all elements larger than the pivot on the right
- Sort the sequence on the left (use Quicksort)
- Sort the sequence on the right (use Quicksort)