# General Computer Science for Engineers CISC 106 Lecture 17

Roger Craig
Computer and Information Sciences
03/25/2009

# Lecture Overview

- Office hours today--> tomorrow
- Project 1
- Linear search vs. Binary search
- Selection sort

# Project 1

- Shark and goldfish simulator
- Teams of 3-5
- Form team by March 27, 11:55pm
- Complete project due April 20
- Functions that need to be implemented are listed
- Each team has 3 members designated as librarian, recorder, and team leader.
- Grading: 1/3 individual work, 1/3 team accomplishments (with blind rating), 1/3 questions on midterm/exam about the project

# Functions

%Make the board. Make the shark and position the shark.

%Calls play and makeBoard

function [] = startShark(rows, cols)

%Recursive function that implements top-level algorithm.

%oldShark is a shark with the previous position.

%Calls: print, plotBoard, getClosestGoldfish, moveToGoldfish, eatGoldfish

function [] = play(board, shark, oldShark)

%Goldfish disappears from board, shark moves into former goldfish location.

%Calls:

function [newBoard newShark] = eatGoldfish(board, goldfishCoords, shark)

%Prints an ascii version of board at interpreter prompt, showing all

%simulated items.

function [] = print(board, shark)

%Plots board, showing all simulated items.

%Calls: plotSquare

function [] = plotBoard(board, shark, oldShark)

# Functions cont.

```
%Plots a single square on board.
%Calls: patch
function [] = plotSquare(row, col, color)

%Shark moves one square at a time, horizontally or vertically, until
%shark is adjacent to goldfish. Returns updated copy of shark.
%Calls: isAdjacent, moveOneSquare, print, plotBoard
function newShark = moveToGoldfish(board, goldfishCoords, shark)

%Move shark one square closer to coords. New shark location must be a
%legal, unoccupied square.
%Calls: isLegalMove
function [canMove newShark] = moveOneSquare(board, coords, shark)

%Returns true iff board is empty at proposed position and position is
%valid for this board.
%Calls:
function legal = isLegalMove(board, proposedMove)
```

# Functions cont.

%Returns true iff coords and shark position are vertically or

%horizontally adjacent.

%Calls:

function flag = isAdjacent(coords, shark)

%For every element of the board matrix, if the element of

%the matrix is GOLDFISH, then calculate a distance. Keep track of

%the minimum distance so far AND the coords of the min distance. If no

%goldfish is found, isGoldfish is false. Otherwise, coords holds the row and col

%coordinates of the min distance goldfish.

%Calls: distance

function [isGoldfish coords] = getClosestGoldfish(board, shark)

%Takes the row, column coordinates of two points on the board and returns

%the distance between them.

%Calls:

function out = distance(x1, y1, x2, y2)

%Makes a matrix of the dimension parameters. Randomly places goldfish in

%about five percent of the board (that is, each square on the board has a

%five percent chance of being goldfish).

%Calls:

function board = makeBoard(rows, cols)

# Linear search vs. Binary Search

Linear search takes N iterations in the worst case

Binary search takes log2(N) iterations in the worst case

# Selection Sort

One of the easiest ways to sort

But computationally inefficient for a big dataset (see Chapman Chapter 5)

```
While not at end of array
  find the minimum value from…
    the current location (or loc)  to the…
    end of the array
  swap the current loc with the min's loc
End While
Return array
```

Number of steps?
Sum(N -1 + n-2 + n -3 + n -4 …. 1)
This is roughly n^2.
(The best sorting algorithms require about n*log(n) )

Selection sort pseudocode
In: unsorted array, array, of length n.
Out: sorted array, array.

```
Loop through from i = 0 to n-1
    minLoc = i
    min = array[i]
    Loop through from j = i+1 to n
        if min > array[j] then
            minLoc = j
            min = array[j]
    end inner loop
    %swap the values at the ith %
%...position and the minLoc position
    tmp = array[i]
    array[i] = array[minLoc]
    array[minLoc] = tmp
End outer loop
Return array
```