ITEC 136

Business Programming Concepts

Week 12, Part 01

Overview

FRANKLIN UNIVERSITY

FOUNDED 1902

Week 12 Overview

- Week 11 review
  - Associative Arrays
  - Common Array Operations
    - Inserting – shifting elements right
    - Removing – shifting elements left
    - Copying – deep vs. shallow
    - Searching – linear vs. binary
  - Appending to an Array

FRANKLIN UNIVERSITY

Week 12 Overview

- Week 11 review
  - Deep Copy vs. Shallow Copy
  - Linear Search – no sort required
  - Binary Search – needs sorted array

Week 12 Overview

- Outcomes
  - Demonstrate the execution of selection, insertion and bubble sorts.
  - Profile and analyze the performance of selection, insertion, and bubble sorts.
  - Work with multi-dimensional arrays.

ITEC 136
Business Programming Concepts

Week 12, Part 02
Sorting

FRANKLIN UNIVERSITY
FOUNDED 1902

Array Sorting

- Sorting an array
  - Many different array sorting algorithms with many different tradeoffs.
  - *Quadratic* sorting algorithms: selection sort, insertion sort, and bubble sort.

Array Sorting

- Sorting an array
  - Many different array sorting algorithms with many different tradeoffs.
  - *Quadratic* sorting algorithms: selection sort, insertion sort, and bubble sort.

Quadratic: the time that it takes to sort an array is proportional to the *square* of the number of elements.

FRANKLIN UNIVERSITY

Array Sorting

- Selection sort
  - Divide array into two parts: sorted left and unsorted right.
  - From the unsorted right, find the smallest element. Swap it with the "border" element.
  - Repeat until all is sorted.

FRANKLIN UNIVERSITY

Array Sorting

- Selection sort

Pass	0	1	2	3	4	5	6	7
0	16	11	21	32	41	20	3	9
1								
2								
3								
4								
5								
6								
7								

border	X
unsorted	X
sorted	X
selected	**X**

Array Sorting

- Selection sort

Pass	0	1	2	3	4	5	6	7
0	16	11	21	32	41	20	**3**	9
1	3	11	21	32	41	20	16	**9**
2	3	9	21	32	41	20	16	**11**
3	3	9	11	32	41	20	**16**	21
4	3	9	11	16	41	**20**	32	21
5	3	9	11	16	20	41	32	**21**
6	3	9	11	16	20	21	**32**	41
7	3	9	11	16	20	21	32	41

border	X
unsorted	X
sorted	X
selected	**X**

Array Sorting

- Selection sort

```
function selectionSort(arr, left, right) {
  for (var i = left; i < right; ++i)
  {
    var min = i;
    for (var j = i; j < right; ++j)
      if (arr[min] > arr[j])
        min = j;

    var temp = arr[min];
    arr[min] = arr[i];
    arr[i] = temp;
  }
}
```

Finds the index of the smallest element

Swaps the border element with the smallest element

Array Sorting

- Selection sort
  - Efficiency: how many comparisons take place?

```
// a portion of the selection sort
for (var i = 0; i < arr.length; ++i)
{
  var min = i;
  for (var j = i; j < arr.length; ++j)
    if (arr[min] > arr[j])
      min = j;
}
```

Array Sorting

- Selection sort
  - Efficiency: how many comparisons take place?

```
// a portion of the selection sort
for (var i = 0; i < arr.length; ++i)
{
  var min = i;
  for (var j = i; j < arr.length; ++j)
    if (arr[min] > arr[j])
      min = j;
}
```

Outer loop executes n times where n is the array length.

Inner loop executes n, n-1, n-2, … 1 times depending on i.

Array Sorting

- Selection sort
  - Efficiency: how many comparisons take place?
    - The orange and yellow boxes in the previous slide show the number of comparisons.

16	11	21	32	41	20	3	9
3	11	21	32	41	20	16	9
3	9	21	32	41	20	16	11
3	9	11	32	41	20	16	21
3	9	11	16	41	20	32	21
3	9	11	16	20	41	32	21
3	9	11	16	20	21	32	41
3	9	11	16	20	21	32	41

Array Sorting

- Selection sort
  - Efficiency: how many comparisons take place?
    - The orange and yellow boxes in the previous slide show the number of comparisons.
    - This is half of an n*n grid. Thus the number of comparisons grows as the square of the array length.

FRANKLIN UNIVERSITY

Array Sorting

- Selection sort
  - Efficiency: how many comparisons take place?
    - If *n* is the length of the array, then the number of comparisons (f(*n*)) mathematically is approximately:

$$f(n) = n^2$$

FRANKLIN UNIVERSITY

Array Sorting

- Selection sort
  - Efficiency: how many comparisons take place?
    - If *n* is the length of the array, then the number of comparisons (f(*n*)) mathematically is approximately:

$$f(n) = n^2$$

A "quadratic" equation. Thus, selection sort is a quadratic algorithm.

Array Sorting

- Insertion sort
  - Divide array into two parts: sorted left and unsorted right.
  - Insert the "border" element into the sorted left where it belongs, shifting elements to the right as needed.
  - Repeat until all is sorted.

Array Sorting

- Insertion sort

Pass	0	1	2	3	4	5	6	7
0	16	11	21	32	41	20	3	9
1								
2								
3								
4								
5								
6								
7								

border	X
unsorted	X
sorted	X

Array Sorting

- Insertion sort

Pass	0	1	2	3	4	5	6	7
0	16	11	21	32	41	20	3	9
1	11	16	21	32	41	20	3	9
2	11	16	21	32	41	20	3	9
3	11	16	21	32	41	20	3	9
4	11	16	21	32	41	20	3	9
5	11	16	20	21	32	41	3	9
6	3	11	16	20	21	32	41	9
7	3	9	11	16	20	21	32	41

border	X
unsorted	X
sorted	X

Array Sorting

- Insertion sort

```
function insertionSort(arr, left, right)
{
  for (var i=left + 1; i<right; ++i)
  {
    var j;
    var temp = arr[i];

    for (j = i - 1; j >= left && arr[j] > temp; --j)
      arr[j + 1] = arr[j];

    arr[j + 1] = temp;
  }
}
```


Array Sorting

- Insertion sort

```
function insertionSort(arr, left, right)
{
  for (var i=left + 1; i<right; ++i)
  {
    var j;
    var temp = arr[i];

    for (j = i - 1; j >= left && arr[j] > temp; --j)
      arr[j + 1] = arr[j];

    arr[j + 1] = temp;
  }
}
```


The index "i" is the border between sorted and unsorted.

Moves every element greater than the border one index right.

Place the border element where it belongs

Array Sorting

- Insertion sort
  - Efficiency: how many element shifts could take place?

Array Sorting

- Bubble sort
  - Divide array into two parts: sorted *right* and unsorted *left*.
  - Compare the leftmost element against its neighbor to the right. If the two are out of order, swap them. Do this for each pair in the row.
  - Repeat for each row until all is sorted.

Array Sorting

- Bubble sort

Pass	0	1	2	3	4	5	6	7
0	16	11	21	32	41	20	3	9
1								
2								
3								
4								
5								
6								
7								

unsorted	X
sorted	X

Array Sorting

- Bubble sort

Pass	0	1	2	3	4	5	6	7
0	16	11	21	32	41	20	3	9
1	11	16	21	32	20	3	9	41
2	11	16	21	20	3	9	32	41
3	11	16	20	3	9	21	32	41
4	11	16	3	9	20	21	32	41
5	11	3	9	16	20	21	32	41
6	3	9	11	16	20	21	32	41
7	3	9	11	16	20	21	32	41

unsorted	X
sorted	X

Array Sorting

- Bubble sort

```
function bubbleSort(arr, left, right) {
  var didSwap = true;
  var last = right - 1;
  while (didSwap) {
    didSwap = false;
    for (var i=left; i<last; ++i)
      if (arr[i] > arr[i + 1]) {
        didSwap = true;
        // swap arr[i] with arr[i + 1] (omitted)
      }
    --last;
  }
}
```


Array Sorting

- Bubble sort

```
function bubbleSort(arr, left, right) {
  var didSwap = true;
  var last = right - 1;
  while (didSwap) {
    didSwap = false;
    for (var i=left; i<last; ++i)
      if (arr[i] > arr[i + 1]) {
        didSwap = true;
        // swap arr[i] with arr[i + 1] (omitted)
      }
    --last;
  }
}
```


Loop terminates when we don't swap anything in a pass (i.e. is sorted).

If they're out of order, swap them and set the flag to true

Always correctly place the largest element at index "last."

Array Sorting

- Bubble sort
  - Efficiency: how many swaps could take place?

Array Sorting

- Profiling sorting algorithms
  - Timing each sorting algorithm on the same set of data multiple times.
  - Plot the data and draw some conclusions.

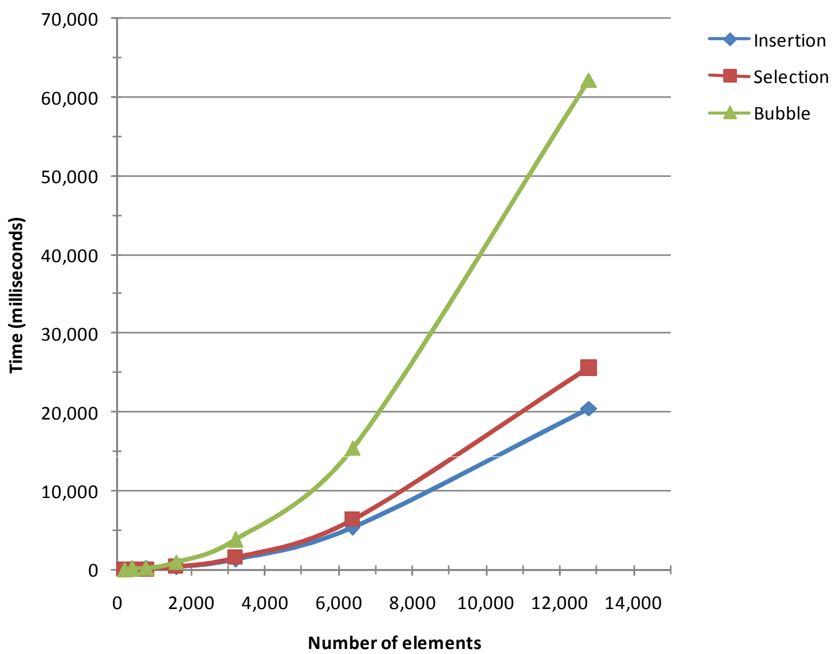
Array Sorting

- Profiling sorting algorithms
  - Code in the course: SortTimings.html

Elements	Insertion	Selection	Bubble
200	16	14	31
400	62	47	125
800	94	78	250
1600	297	375	953
3200	1266	1562	3782
6400	5297	6422	15453
12800	20517	25626	62160

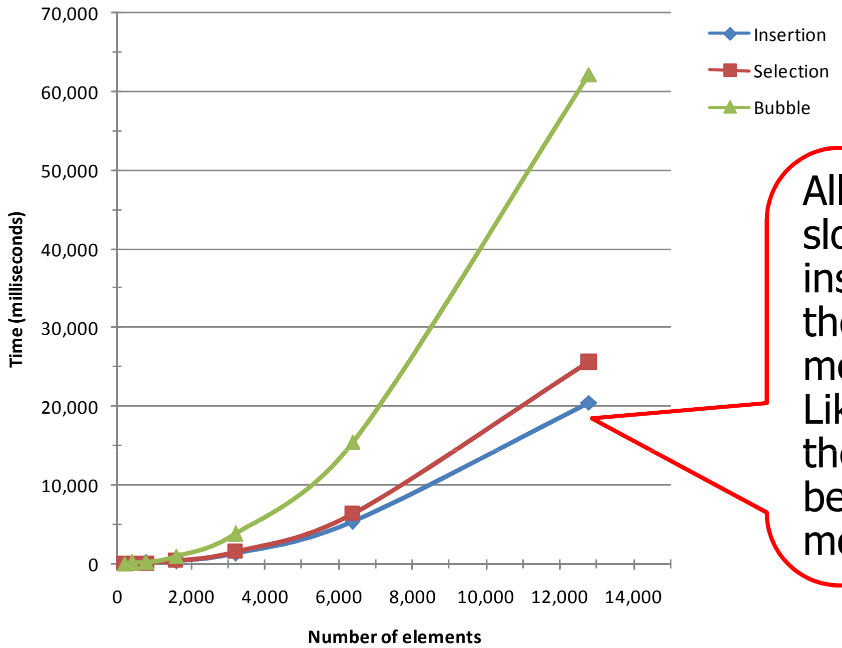
FRANKLIN UNIVERSITY

Array Sorting



FRANKLIN UNIVERSITY

Array Sorting



All these algorithms are slow, however, insertion sort is best of the worst because it moves the least data. Likewise, bubble sort is the worst of the worst because it moves the most data.

FRANKLIN UNIVERSITY

ITEC 136

Business Programming Concepts

Week 12, Part 03

Multi-dimensional arrays

FRANKLIN UNIVERSITY

FOUNDED 1902

Multi-dimensional Arrays

- Multi-dimensional arrays
  - Higher dimensions
    - 1-D: consists of columns
    - 2-D: rows and columns
    - 3-D: depth, rows, and columns
    - 4-D: No 3-space corollary
  - Key idea: 1D arrays are processed with a single loop. 2D arrays are processed with nested loops

Multi-dimensional Arrays

- Multi-dimensional arrays
  - Multi-dimensional arrays are arrays that contain other arrays as data.
  - Ex: matrix in mathematics

```
var matrix = [
    [ 1,  2,  3,  4,  5],
    [ 6,  7,  8,  9, 10],
    [11, 12, 13, 14, 15]
];
alert(matrix.length + " by " + matrix[0].length);
```

Multi-dimensional Arrays

- Multi-dimensional arrays

```
function makeMatrix(rows, cols)
{
    var result = new Array(rows);
    for (var i = 0; i < rows; ++i)
    {
        result[i] = new Array(cols);
    }
    return result;
}
```

FRANKLIN UNIVERSITY

Multi-dimensional Arrays

- Multi-dimensional arrays


```
function makeMatrix(rows, cols)
{
    var result = new Array(rows);
    for (var i = 0; i < rows; ++i)
    {
        result[i] = new Array(cols);
    }
    return result;
}
```


An array gets placed inside another array.

FRANKLIN UNIVERSITY

Multi-dimensional Arrays

- Multi-dimensional arrays

```
function multiplicationTable(rows, cols)
{
    var matrix = makeMatrix(rows, cols);
    for (var i = 0; i < matrix.length; ++i)
    {
        for (var j = 0; j < matrix[i].length; ++j)
        {
            matrix[i][j] = (i + 1) * (j + 1);
        }
    }
    return matrix;
}
```


Multi-dimensional Arrays

- Multi-dimensional arrays

```
function multiplicationTable(rows, cols)
{
    var matrix = makeMatrix(rows, cols);
    for (var i = 0; i < matrix.length; ++i)
    {
        for (var j = 0; j < matrix[i].length; ++j)
        {
            matrix[i][j] = (i + 1) * (j + 1);
        }
    }
    return matrix;
}
```

This is the number of rows

Use double subscripts to access the element

This is the number of columns within a row

Multi-dimensional Arrays

- Multi-dimensional arrays
  - Key idea: 1D arrays are generally processed with a single loop. 2D arrays are generally processed with nested loops.

Questions?

Next Week

- Object-based programming
  - Constructors!
  - Properties!
  - Methods!
  - Oh my!
- Exception handling

FRANKLIN UNIVERSITY

ITEC 136

Business Programming Concepts

Week 12, Part 04

Self Quiz

FRANKLIN UNIVERSITY

FOUNDED 1902

Self Quiz

- Write a function `isSorted` that receives an array as a parameter and returns true if the array is in sorted order, and false otherwise.
- Write a function `randomize` that takes an array as a parameter and swaps randomly selected elements in the array.

Self Quiz

- Write a function `addMatrix` that takes two 2D arrays of the same dimension as parameters. It should return a new array with matching dimensions containing the sum i.e.

4	1
5	2
3	8
10	6

14	3
9	7
4	4
0	1

18	4
14	9
7	12
0	7

Self Quiz

- Write a function `transpose` that takes a 2D array as a parameter and returns an new array with the rows and columns exchanged. i.e.

11	12	13	14
21	22	23	24

→

11	21
12	22
13	23
14	24

Self Quiz

- Demonstrate on paper the sequence of passes through bubble sort, insertion sort, and selection sort for the following array of data

0	1	2	3	4	5	6	7	8	9	10	11
10	2	3	6	9	1	5	4	12	8	7	11

Self Quiz

- How would you reverse the order (ascending vs. descending) of the sorting functions studied this week?
- If an insertion sort of 100,000 elements takes 14 seconds, how long would a sort of 300,000 elements take?

FRANKLIN UNIVERSITY

ITEC 136

Business Programming Concepts

Week 12, Part 05

Upcoming deadlines

FRANKLIN UNIVERSITY

FOUNDED 1902

Upcoming Deadlines

- Due March 30
  - Pre-class exercise 13
  - Homework 10