MATLAB Other Data Structures & Sorting

Week 8

Loosely follows Chapter 10

Other Data Structures

Structures

- Up to now we’ve seen arrays of one element type
    - All numbers
    - All characters
- A ***Structure*** allows you to store multiple data types in various ***fields***
- Think of a structure like an object, objects have properties
- Structures are also known as ***Structs***
- Even though a struct is *like* an object, this is not Object Oriented Programming

Student Structure

- Structure name: student
- Structure fields:
    - name = ‘John Doe’
    - id = ‘jd001’
    - grades = [95, 87, 91]

```
student.name = 'John Doe';
student.id = 'jd001';
student.grades = [95, 87, 91];
```

Structure Output

- Viewing a structure’s contents is the same as any variable

```
student

student =

  struct with fields:

      name: 'John Doe'
        id: 'jd001'
    grades: [95 87 91]
```

Structures as Arrays

- We can easily create an array of structures
- Think of structures as arrays where each element has properties

```
student(2).name = 'Jane Doe';
student(2).id = 'jd002';
student(2).grades = [84, 81, 77];
```

```
student

student =

  1×2 struct array with fields:

    name
    id
    grades
```

Cell Arrays

- A cell is the most general data object in MATLAB
- A cell is like a “data container”
- Cells can hold any data type
  - Numbers
  - Characters
  - Even arrays

Cell Indexing

```
c(1, 1) = {rand(3)};
c(1, 2) = {char('Bologna', 'Salami')};
c(2, 1) = {13};
c(2, 2) = {student};

c

c =

  2×2 cell array

    {3×3 double}    {2×7 char  }
    {[        13]}    {1×2 struct}
```

```
c{1, 1} = rand(3);
c{1, 2} = char('Bologna', 'Salami');
c{2, 1} = 13;
c{2, 2} = student;

c

c =

  2×2 cell array

    {3×3 double}    {2×7 char  }
    {[        13]}    {1×2 struct}
```

Creating Cell Arrays

- Cell arrays can be created in one line like regular arrays

```
c = { rand(3), char('Bologna', 'Salami'); 13, student }

c =

  2×2 cell array

    {3×3 double}    {2×7 char  }
    {[        13]}    {1×2 struct}
```

Creating Cell Arrays

- The *cell* function allows you to preallocate empty cell arrays

```
a = cell(3, 2)

a =

  3×2 cell array

    {0×0 double}    {0×0 double}
    {0×0 double}    {0×0 double}
    {0×0 double}    {0×0 double}
```

Accessing Cell Data

- Remember to use curly braces on the cell array if you want the data
- Recall that c{1,1} is a rand(3).

```
c(1,1)

ans =

  1×1 cell array

    {3×3 double}
```

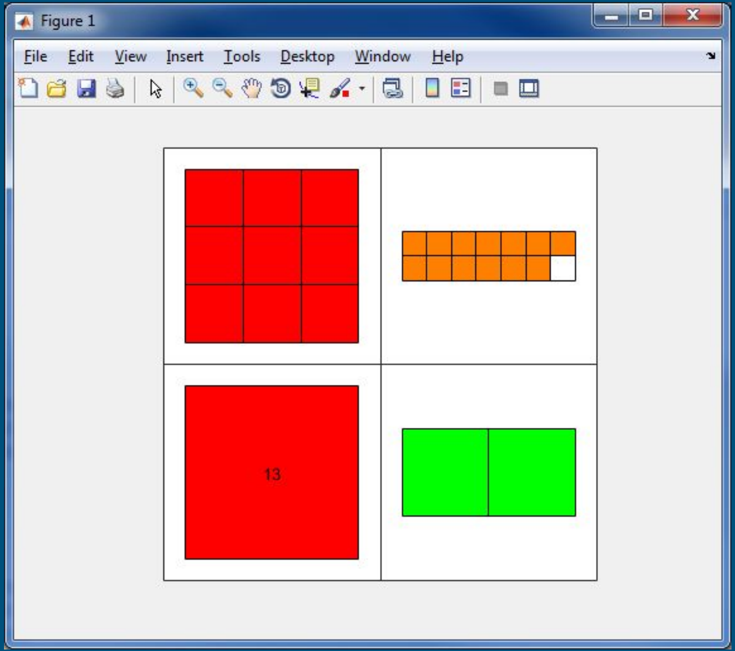
```
c{1, 1}

ans =

    0.7922    0.0357    0.6787
    0.9595    0.8491    0.7577
    0.6557    0.9340    0.7431
```

Visualizing Cell Data

- celldisp() recurses through the array (by column) displaying each cell’s contents.
- cellplot() shows a graphical representation
  - Shows empty content
  - Shows data types by color



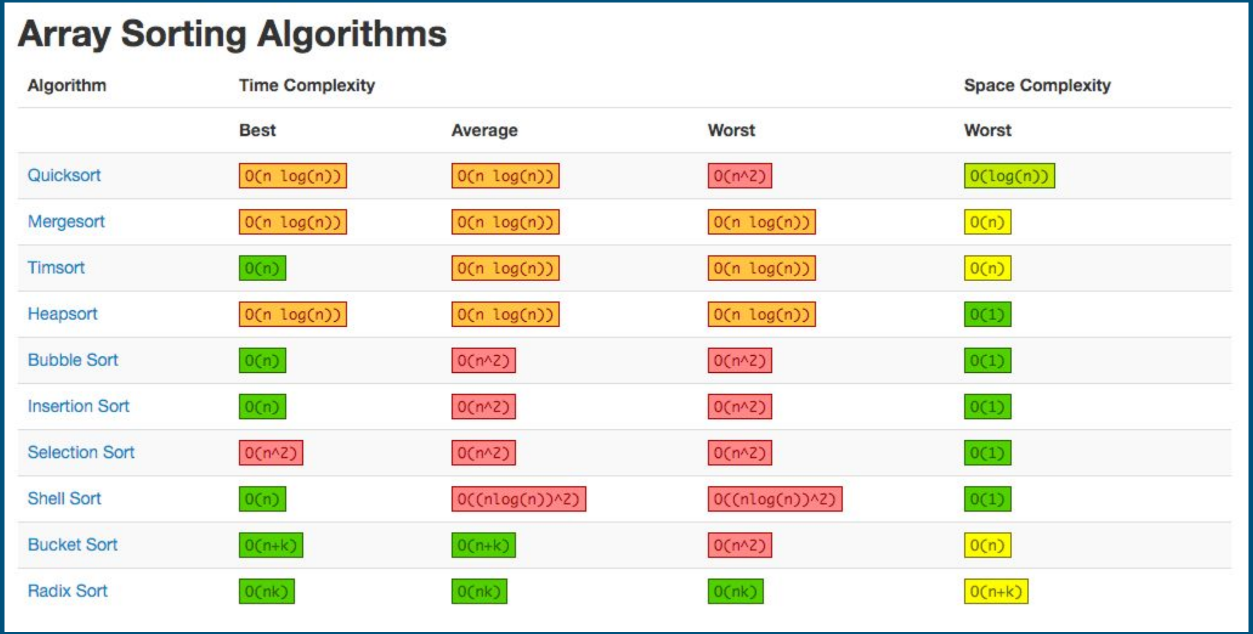
Key Takeaways

- Structs
  - Arrays where elements have properties
- Cell Arrays
  - Use c{x, y} to get raw contents
  - Use c(x, y) to get cell object
  - Create cell arrays like normal arrays { r1c1, r1c2; r2c1, r2c2 }
  - Preallocate with cell() or by assigning = [ ]
  - Visualize with dispcell or plotcell

Sorting

Sorting

- Many different sorting algorithms



Array Sorting Algorithms

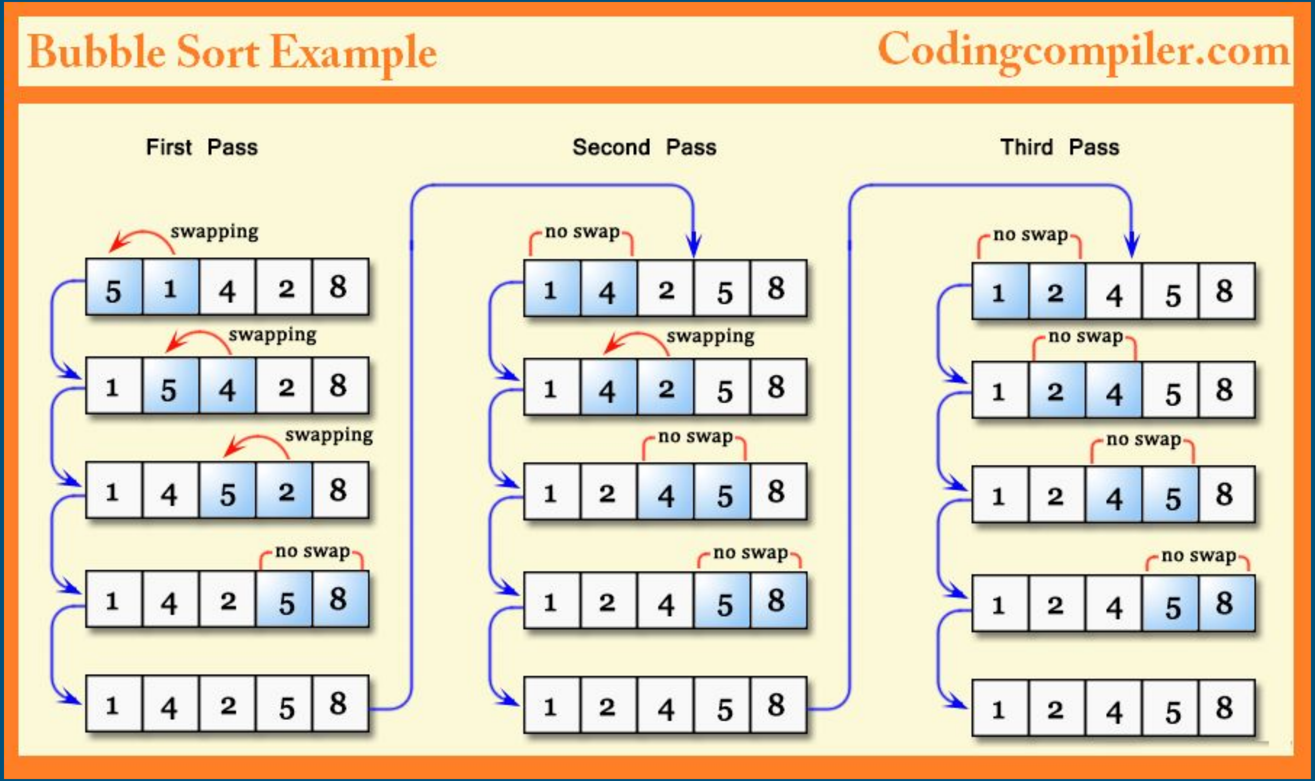
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

Bubble Sort

- Gets its name from the method of sorting
- Large values, like large bubbles, rise to the top faster

- While last iteration experienced a swap
    - swapOccurred = false
    - For each element n in an array up to length - 1
        - If element n > n + 1
            - Swap n and n+1
            - swapOccurred = true

Bubble Sort Example



MATLAB *sort*

- MATLAB offers the sort function for sorting arrays
- Arrays may be sorted by row or column
  - sort(A, 1); % Sorts elements of each column
  - sort(A, 2); % Sorts elements of each row
- Arrays may be sorted in descending or ascending order
  - sort(A, ‘ascend’); % Sorts elements of each column in ascending order
  - sort(A, ‘descend’); % Sorts elements of each column in descending order
- Sorting Explicitly with *sort(matrix, dim, direction)*
  - sort(A, 2, ‘ascend’); % Sorts elements of each row in ascending order