

Data Structures

CSCI 2270

Sorting

Sprint 4

.....

Ready

.....

GO!

Struct

- ❖ Collection of data elements
- ❖ Composite object
- ❖ Give us more flexibility

Struct

```
struct WeatherData
{
    double temp;
    double humidity;
    double windVelocity;
}
```

Struct

```
struct WeatherData
{
    WeatherData();
}

// create an instance
WeatherData dp1;
```

Array

- ❖ Data Structure
- ❖ Static Data
- ❖ Contiguous data in memory

Array

- ❖ What if we need more flexibility?
- ❖ What if we don't want to waste space?
- ❖ What if we don't know the size we need to deal with?

Array

- ❖ Effectively use dynamic pointers
 - `int * myPtr = new int //pointing to int`
 - `int * myPtr = new int[3] //pointing to array of int`

- ❖ Remember to free the memory allocated!!!
 - We are in the Heap

Array

Example:

command line arguments

```
int main(int argc, char *argv[])
{
    ....
}
```

Array

Example:
command line arguments

```
int main(int argc, char *argv[])
{
    // how do we access the argument?
}
```

Array

Example:
command line arguments

```
int main(int argc, char *argv[])
{
    // how do we convert to something
    // meaningful?
}
```

Sorting

- ❖ BubbleSort
- ❖ InsertionSort
- ❖ QuickSort

Sorting:BubbleSort

- ❖ Bubble up
- ❖ Slow
- ❖ Does the job

Sorting:BubbleSort

Pre-conditions

A is an array.

Post-conditions

A is in ascending order.

Algorithm

```
bubbleSort(A){  
    1.    for i = 0 to A.end - 1  
    2.        for j = 0 to A.end - i - 1  
    3.            if (A[j] > A[j+1])  
    4.                swap = A[j]  
    5.                A[j] = A[j+1]  
    6.                A[j+1] = swap}
```

Sorting: InsertionSort

- ❖ Pivot down
- ❖ More efficient than bubble
- ❖ More coding

Sorting: InsertionSort

Pre-conditions

A is an array.

Post-conditions

The array A is in ascending order.

Algorithm

insertionSort(A)

1. for i = 1 to A.end

2. index = A[i]
3. j = i
4. while(j > 0 and A[j - 1] > index)
5. A[j] = A[j - 1]
6. j = j - 1
7. A[j] = index

Sorting: QuickSort

- ❖ Divide and conquer
- ❖ More efficient than others on average
- ❖ Complex coding/ recursion

Sorting: InsertionSort

Pre-conditions

A is an array.

left and *right* are valid integers that index array *A*.

Post-conditions

The array *A* is sorted in ascending order

Algorithm

quickSort(*A*, *left*, *right*)

1. *i* = *left*
2. *j* = *right*
3. *pivot* = *A*[*(left + right) / 2*]
4. while(*i* <= *j*)
5. while(*A*[*i*] < *pivot*)

6. *i* = *i* + 1
7. while(*A*[*j*] > *pivot*)
8. *j* = *j* - 1
9. if(*i* <= *j*)
10. tmp = *A*[*i*]
11. *A*[*i*] = *A*[*j*]
12. *A*[*j*] = tmp
13. *i* = *i* + 1
14. *j* = *j* - 1
15. if (*left* < *j*)
16. quickSort(*A*, *left*, *j*)
17. if (*i* < *right*)
18. quickSort(*A*, *i*, *right*)

Lecture Quiz

Key: nonna