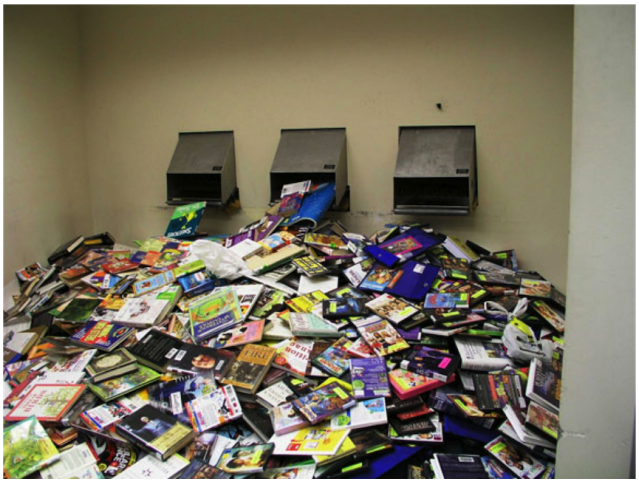
GLIS 691 | Karyn Moffatt

Computer Programming for Information Professionals

Sorting

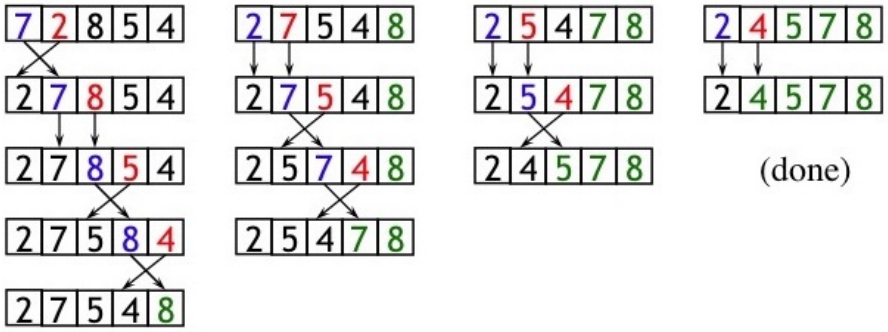
Ok, so pre-sorting helps... But how to sort them?



Bubble Sort

- Compare each element (except the last one) with its neighbour to the right
  - If they are out of order, swap them
  - This puts the largest element at the very end
  - The last element is now in the correct and final place
- Compare each element (except the last two) with its neighbour to the right
  - If they are out of order, swap them
  - This puts the second largest element at the very end
  - The last two elements are now in the correct and final place
- And so on...

Bubble Sort



Bubble Sort in Python

- You are not responsible for writing this code

In [ ]:

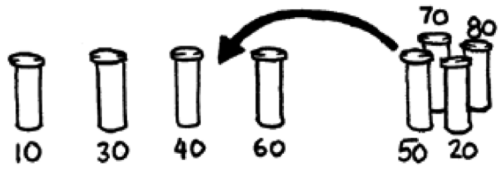
```
def bubbleSort(numList):
    n = len(numList)
    # i indicates how many items were sorted
    for i in range(n-1):
        # traverse the list from 0 to n-i-1
        # (last i elements are already in place)
        for j in range (0, n-i-1):
            # swap the element with next if out of order
            if numList[j] > numList[j+1]:
                numList[j], numList[j+1] = numList[j+1], numList[j]
```

In [ ]:

```
# Testing bubbleSort()
nums = [3, 1, 41, 59, 26, 53, 59]
bubbleSort(nums)
print(nums)
```

Insertion Sort

- Put the first two items in correct relative order
- Insert the third item relative to the first two
- Insert the fourth item relative to the first three
- And so on...



Insertion Sort in Python

- You are not responsible for writing this code

In [ ]:

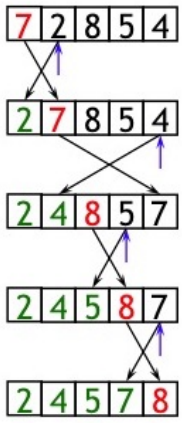
```
def insertionSort(numList):
    n = len(numList)
    # i indicates how many items were sorted
    for i in range(n):
        # get the first non-sorted value and its position from the list
        # currentValue is for convenience, it's the item we are currently s
orting
        currentValue = numList[i]
        # position we need because we don't want to lose the value of i
        position = i

        # until we find the spot for currentValue or reach the start
        while position>0 and numList[position-1]>currentValue:
            # shuffle items over one spot to make a space for currentValue
            numList[position] = numList[position-1]
            # update the position variable
            position -= 1
        # once we've found the spot for current value, put it in place
        numList[position] = currentValue
```

In [ ]:

```
# Testing insertionSort()
nums = [3, 1, 41, 59, 26, 53, 59]
insertionSort(nums)
print(nums)
```

Selection Sort

- Find the smallest value in the list.
  - Put it in the first spot, swapping with the current item in that spot.
- Find the next smallest value in the list.
  - Put it in the second spot, swapping it with the current item in that spot.



Selection Sort in Python

- You are not responsible for writing this code

In [ ]:

```
def selectionSort(numList):
    n = len(numList)
    # i indicates how many items were sorted
    for i in range(n-1):
        # To find the minimum value of the unsorted segment
        # We first assume that the first element is the lowest
        min_index = i
        # We then use j to loop through the remaining elements
        for j in range(i+1, n-1):
            # Update the min_index if the element at j is lower than it
            if numList[j] < numList[min_index]:
                min_index = j
        # After finding the lowest item of the unsorted regions,
        # swap with the first unsorted item
        numList[i], numList[min_index] = numList[min_index], numList[i]
```

In [ ]:

```
nums = [3, 1, 41, 59, 26, 53, 59]
selectionSort(nums)
print(nums)
```

Quick Sort — Uses Divide and Conquer

- Choose a pivot value
- Partition the array
  - Put all values less than the pivot in the left part of the array, and then the pivot itself
  - Then all values greater than the pivot in the right part
- Recursively, sort the values less than the pivot
- Recursively, sort the values greater than the pivot

Watch videos posted on MyCourses

Efficiency

- We'll talk more about this next week but as a general reference

	Insertion Sort			**Quick Sort**		
# of records	Thousand	Million	Billion	Thousand	Million	Billion
home computer	Instant	2.8 hrs	317 yrs	Instant	1 sec	18 mins
super computer	Instant	1 sec	1 week	Instant	Instant	Instant