Special Topics in Computer Science- CSC 4992

Sort Algorithms

Sorting

- One of the most fundamental and widespread ways of organizing information
- Can be a very costly process (time performance), so much ingenuity is devoted to the development of sorting strategies

Sorting

89	56	63	72	41	34	95
0	1	2	3	4	5	6

Puts the elements of a list into some kind of *order*, usually ascending

Elements must be *comparable* using the operators **<**, **>**, and **==**

34	41	56	63	72	89	95
0	1	2	3	4	5	6

Simple Sorting

- The **`sort()`** method modifies a list by sorting its elements into ascending order
- The elements must be comparable using the operators **`<`**, **`>`**, and **`==`**
- Returns **`None`**

Examples

```
>>> names = ['Hillary', 'Donald', 'Tim', 'Mike']

>>> names.sort()

>>> names
['Donald', 'Hillary', 'Mike', 'Tim']

>>> numbers = [88, 44, 99, 33, 44, 22]

>>> numbers.sort()

>>> numbers
[22, 33, 44, 44, 88, 99]
```

How Does `sort` Work?

- `sort` uses a fast but very sophisticated algorithm
- Let's examine two simpler but slower algorithms called *selection sort* and *bubble sort*
- There is often a tradeoff between the complexity of an algorithm and its performance

Bubble Sort Strategy

- Compare the first two elements and if they are out of order, exchange them
- Repeat this process for the second and third elements, etc.
- At the end of this process, the largest element will have bubbled down to the end of the list
- Repeat this process for the unsorted portion of the list, etc.

Formalize the Strategy

```
set n to the length of the list

while n > 1

    bubble the elements from position 0 to position n - 1

    decrement n
```

Formalize the Strategy

```
set n to the length of the list

while n > 1

    for each position i from 1 to n - 1

        if the elements at i and i - 1 are out of order

            exchange them

    decrement n
```

Define **bubbleSort**

```
def bubbleSort(lyst):
    n = len(lyst)
    while n > 1:                                # Do n - 1 bubbles









        n -= 1
```

Define **bubbleSort**

```
def bubbleSort(lyst):
    n = len(lyst)
    while n > 1:                                # Do n - 1 bubbles
        i = 1                                   # Start each bubble
        while i < n:




            i += 1
        n -= 1
```

Define **bubbleSort**

```
def bubbleSort(lyst):
    n = len(lyst)
    while n > 1:                          # Do n - 1 bubbles
        i = 1                             # Start each bubble
        while i < n:
            if lyst[i] < lyst[i - 1]:     # Exchange if needed
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
            i += 1
        n -= 1
```

Add a Trace Option

```
def bubbleSort(lyst, traceOn = False):
    n = len(lyst)
    while n > 1:                          # Do n - 1 bubbles
        i = 1                             # Start each bubble
        while i < n:
            if lyst[i] < lyst[i - 1]:     # Exchange if needed
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
                if traceOn: print(lyst)
            i += 1
        n -= 1
```

A Tester Function

```
def testSort(sortFunction, size = 1, traceOn = False):
    lyst = range(1, size + 1)
    random.shuffle(lyst)
    print(lyst)
    sortFunction(lyst, traceOn)
    print(lyst)
```

A Tester Function

```
def testSort(sortFunction, size = 1, traceOn = False):
    lyst = range(1, size + 1)
    random.shuffle(lyst)
    print(lyst)
    sortFunction(lyst, traceOn)
    print(lyst)
```

```
>>> testSort(sortFunction = bubbleSort, size = 5)
[1, 4, 5, 3, 2]
[1, 2, 3, 4, 5]
```

A Tester Function

```
def testSort(sortFunction, size = 1, traceOn = False):
    lyst = range(1, size + 1)
    random.shuffle(lyst)
    print(lyst)
    sortFunction(lyst, traceOn)
    print(lyst)
```

```
>>> testSort(sortFunction = bubbleSort, size = 5)
[1, 4, 5, 3, 2]
[1, 2, 3, 4, 5]

>>> testSort(sortFunction = bubbleSort, size = 5, traceOn = True)
[1, 4, 2, 5, 3]
[1, 2, 4, 5, 3]
[1, 2, 4, 3, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

Selection Sort Strategy

- Search for the smallest element in the list
- Exchange that element with the element in the first position
- Repeat this process with the rest of the list after the first position, etc.

Formalize the Strategy

```
for each position i in a list of length n

    select the smallest element from positions i to n - 1

    exchange that element with the ith element
```

Define **selectionSort**

```
def selectionSort(lyst):
    i = 0
    while i < len(lyst) - 1:        # Do n - 1 searches









        i += 1
```

Define **selectionSort**

```
def selectionSort(lyst):
    i = 0
    while i < len(lyst) - 1:          # Do n - 1 searches
        minPos = i                    # Start each search
        probe = i + 1
        while probe < len(lyst):
            if lyst[probe] < lyst[minPos]:
                minPos = probe
            probe += 1




        i += 1
```

Define **selectionSort**

```
def selectionSort(lyst):
    i = 0
    while i < len(lyst) - 1:          # Do n - 1 searches
        minPos = i                    # Start each search
        probe = i + 1
        while probe < len(lyst):
            if lyst[probe] < lyst[minPos]:
                minPos = probe
            probe += 1
        if minPos != i:               # Exchange if needed
            temp = lyst[i]
            lyst[i] = lyst[minPos]
            lyst[minPos] = temp
        i += 1
```

Add **traceOn**

```
def selectionSort(lyst, traceOn = False):
    i = 0
    while i < len(lyst) - 1:          # Do n - 1 searches
        minPos = i                    # Start each search
        probe = i + 1
        while probe < len(lyst):
            if lyst[probe] < lyst[minPos]:
                minPos = probe
            probe += 1
        if minPos != i:               # Exchange if needed
            temp = lyst[i]
            lyst[i] = lyst[minPos]
            lyst[minPos] = temp
            if traceOn: print(lyst)
        i += 1
```

Testing **selectionSort**

```
>>> testSort(selectionSort, 5, True)
[4, 5, 1, 2, 3]
[1, 5, 4, 2, 3]
[1, 2, 4, 5, 3]
[1, 2, 3, 5, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]

>>> selectionSort([1,2,3,4], traceOn = True)

>>>
```

Using Sort Criteria

- We might want to sort elements into descending order
- We might want to sort them by a criterion, such as account name, account balance, etc.
- We can use the same strategy that we employed in searching: package the comparison as a specialized function and pass it as an extra argument to the sort function

Simple **bubbleSort**

```
def bubbleSort(lyst):
    n = len(lyst)
    while n > 1:
        i = 1
        while i < n:
            if lyst[i] < lyst[i - 1]:
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
            i += 1
        n -= 1
```

The default test uses the **<** operator

Generic **bubbleSort**

```
def bubbleSort(lyst, test):
    n = len(lyst)
    while n > 1:
        i = 1
        while i < n:
            if test(lyst[i], lyst[i - 1]):
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
            i += 1
        n -= 1
```

Pass the test as a function argument

Generic **bubbleSort**

```
def bubbleSort(lyst, test = lambda x, y: x < y):
    n = len(lyst)
    while n > 1:
        i = 1
        while i < n:
            if test(lyst[i], lyst[i - 1]):
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
            i += 1
        n -= 1
```

Better still, retain the default by using a **lambda** function

Generic **bubbleSort**

```
def bubbleSort(lyst, test = lambda x, y: x < y):
    n = len(lyst)
    while n > 1:
        i = 1
        while i < n:
            if test(lyst[i], lyst[i - 1]):
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
            i += 1
        n -= 1
```

```
>>> bubbleSort(lyst)                          # Default order

>>> bubbleSort(lyst, lambda x, y: x > y)      # Descending order
```

Generic **bubbleSort**

```
def bubbleSort(lyst, test = lambda x, y: x < y):
    n = len(lyst)
    while n > 1:
        i = 1
        while i < n:
            if test(lyst[i], lyst[i - 1]):
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
            i += 1
        n -= 1
```

```
>>> bubbleSort(accounts,
               lambda x, y: x.getBalance() > y.getBalance())
```

By balance, from highest to lowest

Generic **bubbleSort**

```
def bubbleSort(lyst, test = lambda x, y: x < y):
    n = len(lyst)
    while n > 1:
        i = 1
        while i < n:
            if test(lyst[i], lyst[i - 1]):
                temp = lyst[i]
                lyst[i] = lyst[i - 1]
                lyst[i - 1] = temp
            i += 1
        n -= 1
```

```
>>> bubbleSort(accounts,
               lambda x, y: x.getName() < y.getName())
```

By name, alphabetically