CISC101 Reminders & Notes

- Test 3 this week in tutorial
- USATs at the beginning of next lecture
  - Please attend and fill out an evaluation
- School of Computing First Year Information Session
  - Thursday, March 24th from 5:30-7:00PM
  - Goodwin Hall, Room 254
    - Overview of programs including Computing and the Arts, Biomedical Computing, Cognitive Science and Software Design
- Remaining lecture topics have shifted
  - May not cover GUIs or other Python modules in-depth

Slides courtesy of Dr. Alan McLeod

Today

- From last time …
  - Finding minimums and maximums
    - Slides 31-37
  - Timing code execution
    - Slides 38-42
- Sequential Search
- Binary Search
- Selection Sort (likely …)
- Insertion Sort (perhaps …)

Slides courtesy of Dr. Alan McLeod

Searching in Python

- We already have searching methods as well as the keywords **`in`** and **`not in`**
  - **`count(…)`** and **`index(…)`** for lists
  - **`find(…)`**, **`count(…)`** and **`index(…)`** for strings
- A search could return different results
  - A count of occurrences
  - **`True`** or **`False`**
  - Just the location of the first match
- So, why do we need to write our own searching functions?

Slides courtesy of Dr. Alan McLeod

Searching in Python - Cont.

- You might need to search datasets in a programming language that does not have these methods or functions built-in
- Your dataset structure might not be amenable for use with the built-in methods
- So, you need to know these algorithms!

Slides courtesy of Dr. Alan McLeod

Sequential Search

- Sequential search pseudocode

- *Loop through the dataset starting at the first element until the value of the target matches one of the elements*
- *Return the location of the match*
- *If a match is not found, raise ValueError*

- Note that the ***aList*.index(…)** method also throws a ValueError exception if the value is not located

Slides courtesy of Dr. Alan McLeod

Sequential Search - Cont.

```
def sequentialSearch(numsList, target) :
    i = 0
    size = len(numsList)
    while i < size :
        if numsList[i] == target :
            return i
        i = i + 1
    raise ValueError("Target not found.")
```

Note how **len(numsList)** *is done outside loop*

Slides courtesy of Dr. Alan McLeod

Sequential Search -Version 2

```
def sequentialSearch2(numsList, target) :
    for i in range(len(numsList)) :
        if numsList[i] == target :
            return i
    raise ValueError("Target not found.")
```

Uses our trusty **for** *loop, but is it faster?*

Slides courtesy of Dr. Alan McLeod

Timing Our Search

- Demo: TimingSeqSearch.py

- Note how the exception is raised and caught
- The farther the target is from the beginning of the dataset, the longer the search takes
  - *Makes sense!*
- Our fastest sequential search is still 2X slower than ***aList*.index(…)**
  - *Why?*

Slides courtesy of Dr. Alan McLeod

Other Search Return Values

- **`True`** if a match exists and **`False`** otherwise
- A count of how many values match
- A list of locations that match
  - Not built-in to Python
- The location of the match searching from the end of the list, not the beginning

Slides courtesy of Dr. Alan McLeod

Searching an Ordered Dataset

- How do you find a name in a telephone book?
- How do you find a word in a dictionary?
- In the Week 5 lab, Exercise 3 involved coding a number guessing game
  - What is the most effective way of guessing the unknown number?

Slides courtesy of Dr. Alan McLeod

Binary Search

- Binary search pseudocode
  - Only works on ordered sets

a) Locate the midpoint of dataset to search
- *Test the midpoint to see if it matches*

b) Determine if target is in lower half or upper half of the dataset
- *If in lower half, make that half the dataset to search*
- *If in upper half, make that half the dataset to search*
- *Loop back to step a) unless you've exhausted all the possible values*
  - *Raise a ValueError exception*

Slides courtesy of Dr. Alan McLeod

Binary Search – Cont.

```
def binarySearch(numsList, target):
    low = 0
    high = len(numsList) - 1
    while low <= high :
        mid = (low + high) // 2
        if target < numsList[mid]:
            high = mid - 1
        elif target > numsList[mid]:
            low = mid + 1
        else:
            return mid
    raise ValueError("Target not found.")
```


Slides courtesy of Dr. Alan McLeod

Binary Search – Cont.

- What is the best case?
  - The element matches right at the middle of the dataset, and the loop only executes once
- What is the worst case?
  - **`target`** will not be found and the maximum number of iterations will occur
- Note that the loop will execute until there is only one element left that does not match
- Each time through the loop the number of elements left is halved

Slides courtesy of Dr. Alan McLeod

Binary Search – Cont.

- Number of elements to be searched (progression)

$$n, \frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}, \ldots, \frac{n}{2^m}$$

- The last comparison is for $n/2^m$, when the number of elements is one (worst case)
  - So, $n/2^m = 1$ or $n = 2^m$
  - $m = log_2(n)$
- So, the algorithm loops $log(n)$ times in the worst case

Slides courtesy of Dr. Alan McLeod

Binary Search - Cont.

- Binary search with $log(n)$ iterations for the worst case is much better than n iterations for the worst case with a sequential search!
- Major reason to sort datasets!

Slides courtesy of Dr. Alan McLeod

Binary Search – Timing

- Demo: TimingBothSearches.py
  - Much better time now!
- Does *`aList`*`.index(…)` work any faster with a `sorted(…)` list?
- Can *`aList`*`.index(…)` assume the list is sorted and thus switch to a binary search?
- Could *`aList`*`.index(…)` determine if the list is in order and then switch to binary search?
  - How would it do this?

Slides courtesy of Dr. Alan McLeod

Sorting Overview

- We will look at three simple sorts:
  - Selection sort
  - Insertion sort
  - Bubble sort
- We might get a quick peek at Quicksort, but you will not be responsible for knowing this one

Slides courtesy of Dr. Alan McLeod

Sorting Overview – Cont.

- The first step in sorting is to select the criteria used for the sort and the direction of the sort
- It could be ascending numeric order, or alphabetic order by last name, *etc.*

Slides courtesy of Dr. Alan McLeod

Choosing a Sorting Algorithm

- How large is the dataset?
- What is critical: memory usage or execution time?
- Will the algorithm be asked to sort …
  - a dataset that is already in order except for a few newly added elements
  - a completely disordered dataset?

Slides courtesy of Dr. Alan McLeod

Comparing Sorting Algorithms

- Sorting algorithms can be compared using …
  - the number of comparisons for a dataset of size n
  - the number of data movements ("swaps") necessary
  - how these measures change with n
    - Complexity analysis!
- Often need to consider these measures for best case (data almost in order), average case (random order), and worst case (reverse order)
  - Some algorithms behave the same regardless of the state of the data
  - Others do better depending on how well the data is initially ordered

Slides courtesy of Dr. Alan McLeod

Comparing Sorting Algorithms – Cont.

- What if you're sorting simple values like integers?
  - Comparisons are easy to carry out
  - Keep the number of data movements to a minimum
- What if you're sorting strings or objects?
  - Comparisons are more time-consuming
  - Keep the number of comparisons to a minimum
- The only real measure of what algorithm is the best is an actual measure of elapsed time
  - The initial choice can be based on theory alone
  - The final choice for a time-critical application must be made using actual experimental measurement

Slides courtesy of Dr. Alan McLeod

Sorting Overview – Cont.

- I will be presenting code samples that sort lists of integers into ascending order
  - This is easiest to understand
- However the logic of the algorithm can be applied directly to lists of strings or other objects
- Different orders often only require you to change the comparison

Slides courtesy of Dr. Alan McLeod

Before We Start …

- You need to learn these algorithms for the same reasons you needed to learn searching algorithms
- The `sort(…)` in Python is way faster
  - It uses Quicksort, which uses recursion
  - Both topics our outside the scope of this course but covered in CISC121
- Even if we coded Quicksort it would still be slower because of the interpreted *vs.* compiled issue

Slides courtesy of Dr. Alan McLeod

Selection Sort

- An "instinctive" sorting approach
  - Look for the smallest element in the list
  - Put it in at the beginning of the list
  - Repeat with the remaining elements as the list

- *Loop through the array from $i=0$ to one element short of the end of the array*
  - *Select the smallest element in the array range from $i + 1$ to the end of the array*
  - *Swap this value with the value at position i*

Slides courtesy of Dr. Alan McLeod

Swapping Elements

- First, a **swap**(…) function that will be used by this and other sorts:

```
def swap(numsList, pos1, pos2) :
    numsList[pos1], numsList[pos2] =
  numsList[pos2], numsList[pos1]
    # Alternate:
    #temp = numsList[pos1]
    #numsList[pos1] = numsList[pos2]
    #numsList[pos2] = temp
```


Slides courtesy of Dr. Alan McLeod

Selection Sort - Cont.

```
def selectionSort(numsList):
    i = 0
    size = len(numsList)
    while i < size - 1:
        smallestPos = i
        j = i + 1
        while j < size:
            if numsList[j] < numsList[smallestPos]:
                smallestPos = j
            j = j + 1
        if smallestPos != i:
            swap(numsList, i, smallestPos)
        i = i + 1
```


Slides courtesy of Dr. Alan McLeod

Aside - Sorting "in situ"

- Our code is sorting the list in place
- Saves the memory (and time) required to create a copy of the same list in memory
- However, this means that once it is sorted, and since it is passed by reference, it stays sorted!

Slides courtesy of Dr. Alan McLeod

Insertion Sort

- Another "instinctive" kind of sort
  - Start with the first element as a sorted sub-list
  - Take the first element from the unsorted sub-list and find its location in the sorted sub-list
  - Shift the sorted elements up and insert the element

- *Loop through the list from $i=1$ to $size-1$, selecting element $temp$ at position i*
  - *Locate position for $temp$ (position j, where $j <= i$), and move all elements above j up one location*
  - *Put $temp$ at position j*

Slides courtesy of Dr. Alan McLeod

Insertion Sort - Cont.

```
def insertionSort(numsList):
    for i in range(1, len(numsList)):
        temp = numsList[i]
        j = i
        while j > 0 and temp < numsList[j - 1]:
               numsList[j] = numsList[j - 1]
               j = j - 1
        numsList[j] = temp
```


Slides courtesy of Dr. Alan McLeod

Selection Sort *vs.* Insertion Sort

- Selection sort is “swap efficient”
- Insertion sort can be efficient for datasets that are mostly in order

Slides courtesy of Dr. Alan McLeod

Sorting Demo

- Demo: SortingTest.py
- How does our sort compare to *aList*`.sort()`?
  - *Not very well*
  - *How is* `.sort()` *so fast?*
- How different are the other two sorts?

Slides courtesy of Dr. Alan McLeod

Sorting Animations

- For a collection of animation links see:

`http://www.hig.no/~algmet/animate.html`

- Here are a couple that I liked:

`http://www.cs.pitt.edu/~kirk/cs1501/animations/Sort3.html`

`http://cs.smith.edu/~thiebaut/java/sort/demo.html`

Slides courtesy of Dr. Alan McLeod