Searching and Sorting (2)

Wait … we can even do better!

Do we really need to search one-by-one from the beginning?

The answer is NO. Binary search is much more faster.

```
def binarySearch(self, titleToSearch ):  # assume titles are sorted
    found = False
    left = 0
    right = len(self)
    mid = (left + right) // 2
    while found == False and left <= right:
      if self.songs[mid].title.lower() == titleToSearch.lower():
        found = True
        break    # leave the loop
      elif self.songs[mid].title.lower() > titleToSearch.lower():  # search the left half
        right = mid - 1
      else:     # search the right half
        left =  mid + 1
      mid = (left + right) // 2

    if found == True:
      return self.songs[mid]
    else:
      return None
```

How to sort a list? (We did it in hw3!)

```
def sort(aList):
    ''' sort returns a list of the elements of aList in ascending order.
        Input aList: a list '''
    if aList == []:
        return []
    else:
        sortedList = sort(aList[1:])
        return insertOne(aList[0], sortedList)


def insertOne(element, aList):
    ''' Inserts element into its proper place in a sorted list aList.
        Input: element is an item to be inserted.  aList is a sorted list.
        Output: A sorted list.'''
    if len(aList) == 0:
        return [element]
    elif element < aList[0]:
        return [element] + aList
    else:
        return aList[0:1] + insertOne(element, aList[1:])
```


Sorting Revisited

How to do a selection sort in an Imperative style?

Develop a plan:

Find index of smallest element in list.
Swap that with the first element.
Find index of 2nd smallest element in list.
Swap that with the 2nd element.
Repeat until we run out of elements.

Imperative style – we use loops, break into subtasks, and change values of variables "in-place."

Subtasks identified:

Find index of minimum of a list
Swap two elements in a list

Index of minimum in a list

```
def indexOfMinimum(aList, startIndex):
    ''' returns index of the minimum element
        at or after startIndex.
    '''
    minIndex = startIndex
    for i in range(startIndex, len(aList)):
        if aList[i] < aList[minIndex]:
            minIndex = i

    return minIndex
```


Swap two elements in a list

```
def swap(a, b):
    ''' swaps the values of a and b '''
    temp = a
    a = b
    b = temp
```

Try it with

```
def main():
    aList = [5, 3, 4, 2, 7]
    swap(aList[0], aList[3])
    print(aList)
```

Doesn't work! Why?

Swap two elements in a list – 2nd Try

```
def swap(aList, i, j):
    ''' swaps the values of aList[i] and aList[j] '''
    temp = aList[i]
    aList[i] = aList[j]
    aList[j] = temp
```

Try it with

```
def main():
    aList = [5, 3, 4, 2, 7]
    swap(aList, 0, 3)
    print(aList)
```

Works! Why?

7

Sorting Revisited

Put the pieces together

```
def selectionSort(aList):
    ''' sort aList in an imperative style:
        iteratively, subtasks, and in-place '''

    for start in range(len(aList)):
        minIndex = indexOfMinimum(aList, start)
        swap(aList, start, minIndex)
```

Demonstrate selction-sort.py and album-app.py

8

Some notes

- Import: note in album-app.py we used

from song **import** *

from album **import** *

which means reading the file from song.py and album.py, making all functions, objects in these files available for current application.

Handling csv files

- Now let's make the list of songs into a file. We then read the song list from a file and make it available for other applications.
- We will use the csv reading function we learned a few days ago.
- We then will create a dictionary using the artist as the **key** and a list of the songs that the artist sung as the **value**.

Demonstrate album-app-d.py