cycle_7_lecture_notes

February 27, 2021

Plan for today: - Discuss the idea of sorting - outline several sorting algorithms - BubbleSort - Selection Sort - consider big-O complexity of these

We will do some demonstrations of sorting algorithms, and then look at or produce code.

For extra demonstrations (and information on MergeSort, which is a more advanced sorting algorithm), Computerphile has a nice YouTube video: https://www.youtube.com/watch?v=kgBjXUE_Nwc

Not necessarily useful to you, but I found it fun: a visual and audio representation of lots of different sorting algorithms. https://www.youtube.com/watch?v=kPRA0W1kECg&t=14s (I find HeapSort to be very ominous)

Selection sort is my favourite simple-ish sorting algorithm, because I find it the easiest to remember, and the easiest to understand recursively.

Selection sort works like this: - traverse over the list. - For each position in the list, find the smallest element that comes after that position - swap that smallest with the current element You're done!

Why does this work? Let's do a demonstration with cards.

OK, now let's try and write this out recursively:

In pseudocode: Base case?

Recursive case: - find the smallest thing in the list, swap it to the front.
- then call the function again just on the rest of the list

Now let's look at some code:

[1]:
```
def swap(myList, i, j):
    tmp = myList[j]
    myList[j] = myList[i]
    myList[i] = tmp

def findMinimumInd(myList):
    if len(myList) <=0:
        return None
    smallestInd = 0
    for i in range(len(myList)):
        if myList[i] < myList[smallestInd]:
            smallestInd = i
    return smallestInd
```

```
def selectionRecursive(myList):
#     print ("\n\n\n sorting list: ")
#     print (myList)
    size = len(myList)
    if size <= 1:
        return myList
    else:
        smallestInd = findMinimumInd(myList)
        swap(myList, 0, smallestInd)
        return [myList[0]] + selectionRecursive(myList[1:])


myList = [6, 2, 3, 7, 9, 1, 4, 10, 8, 5]
myList = selectionRecursive(myList)
print(myList)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In your lab this week I'll ask you to implement an iterative version of this.

Let's move on now to talking about another sorting algorithm - BubbleSort.

BubbleSort is so-named because the values 'bubble' up through the list. BubbleSort always compares pairs of adjacent values.

We'll do an example with cards.

Now let's look at some code:

[2]:
```
def bubbleSort(myList):
    n = len(myList)

    # Iterate over each element
    for i in range(n):

        # Last i elements are already in order
        for j in range(0, n-i-1):
            # Swap if the element found is greater
            # than the next element
            if myList[j] > myList[j+1] :
                tmp = myList[j]
                myList[j] = myList[j+1]
                myList[j+1] = tmp

forSorting = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(forSorting)
print(forSorting)
```

```
[11, 12, 22, 25, 34, 64, 90]
```

Now let's talk about big-O complexity. Any ideas?

Think back to our strategies: can you recognise a pattern in the BubbleSort code? How much longer will the code take if we add one more item to the list?

Exercise: try to trace a couple small examples of the BubbleSort and SelectionSort, and convince yourself of their complexity.

[ ]: