Introduction to Python Programming

(7) Sorting Algorithms

S. Thater and A. Friedrich

Saarland University

Winter Semester 2011/2012
Sorting Algorithms

Agenda:
- Insertion Sort
- Selection Sort
- Bubblesort
- Mergesort
- Quicksort

Goals:
- Understand the above sorting algorithms.
- Get an idea that there are differences in efficiency.
- Get used to recursion.
- We won’t talk about complexity theory in this lecture.
Swapping two items of a list

How can we swap two items of a list?

```python
1 a = [5, 3]
2
3 <your code here>
```

Today, we always assume that our list is called \( a \).
Selection Sort

Until list is empty:

Find the smallest element of the remaining list.
Append it to a new list.

<table>
<thead>
<tr>
<th>i</th>
<th>Input List</th>
<th>Output List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>D</td>
<td>A B</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>A B C</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>A B C D</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>A B C D E</td>
</tr>
</tbody>
</table>
```python
def selection_sort(a):
    """ sorting algorithm, creates a NEW list """
    b = []
    # Why is the list a copied here?
    a = a[:]
    while len(a) > 0:
        # Find minimum of the list
        minimum = 0
        for i in range(1, len(a)):
            if a[i] < a[minimum]:
                minimum = i
        # Remove the minimum from the list
        # and append it to the new list
        b.append(a.pop(minimum))
    return b
```

Selection Sort - in place

For each $i$ from 0 to $N - 2$:

Find the smallest element of the remaining list.

If this element is smaller than $a[i]$, exchange it with $a[i]$. 

---

<table>
<thead>
<tr>
<th>i</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>D B A E C</td>
</tr>
<tr>
<td>1</td>
<td>A B D E C</td>
</tr>
<tr>
<td>2</td>
<td>A B D E C</td>
</tr>
<tr>
<td>3</td>
<td>A B C E D</td>
</tr>
<tr>
<td>4</td>
<td>A B C D E</td>
</tr>
</tbody>
</table>
```python
def selection_sort_in_place(a):
    """ sorting algorithm, works in-place"""
    # point to element at position i
    for i in range(0, len(a)):
        # Find the smallest element in the rest of the list
        minimum = i
        for j in range(i+1, len(a)):
            if a[j] < a[minimum]:
                minimum = j
        # exchange elements
        temp = a[i]
        a[i] = a[minimum]
        a[minimum] = temp
```
Insertion Sort

Look at one element after the other and insert it into the list at the correct position.
Insertion Sort - in-place

Insertion Sort

Look at one element after the other and insert it into the part of the list already dealt with at the correct position.
def insertion_sort_in_place(a):
    """ sorting algorithm, works in-place"""
    # go through list
    for i in range(1, len(a)):
        # Remember value of a[i]
        v = a[i]
        j = i
        # Go backwards in list and shift to
        # the right if element > value to be
        # inserted
        while a[j-1] > v and j>=1:
            a[j] = a[j-1]
            j -= 1
        # insert the value at its correct position
        a[j] = v
## Bubble Sort

Go through the list again and again and exchange neighboring elements (if necessary). Stop if no changes have been made while going through the list once.

**Implementation:** Your task. Try not to just copy it from the web. :)

<table>
<thead>
<tr>
<th>i</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>D  B  A  E  C</td>
</tr>
<tr>
<td>1</td>
<td>D  B  A  C  E</td>
</tr>
<tr>
<td>2</td>
<td>D  B  A  C  E</td>
</tr>
<tr>
<td>3</td>
<td>D  A  B  C  E</td>
</tr>
<tr>
<td>4</td>
<td>A  D  B  C  E</td>
</tr>
<tr>
<td>5</td>
<td>A  D  B  C  E</td>
</tr>
<tr>
<td>6</td>
<td>A  D  B  C  E</td>
</tr>
<tr>
<td>7</td>
<td>A  B  D  C  E</td>
</tr>
<tr>
<td>8</td>
<td>A  B  D  C  E</td>
</tr>
<tr>
<td>9</td>
<td>A  B  D  C  E</td>
</tr>
<tr>
<td>10</td>
<td>A  B  C  D  E</td>
</tr>
<tr>
<td>11</td>
<td>A  B  C  D  E</td>
</tr>
<tr>
<td>12</td>
<td>A  B  C  D  E</td>
</tr>
<tr>
<td>13</td>
<td>A  B  C  D  E</td>
</tr>
<tr>
<td>14</td>
<td>A  B  C  D  E</td>
</tr>
<tr>
<td>15</td>
<td>A  B  C  D  E</td>
</tr>
<tr>
<td>16</td>
<td>A  B  C  D  E</td>
</tr>
</tbody>
</table>
Recursion: Base Case

```
1   def factorial(x):
2       # base case
3       if (x <= 1):
4           return 1
5       # recursive call
6       return x*factorial(x-1)
7
8   print("factorial(4) =", factorial(4))
```

- ‘Execute’ above function call ON BOARD
- Don’t forget to code the base case in recursive functions.
- Otherwise, you create an infinite number of function calls.
Recursion: Base Case

Base cases for sorting algorithms:

- empty lists are sorted.
- lists with only one element are sorted.

All other lists may still need sorting.

```python
1   if len(a) <= 1:
2       # base case
3       # a is sorted by definition
4       return a
5   else:
6       # need to sort a (recursively)
```
Mergesort: Divide & Conquer

Divide lists & sort them recursively.

Merge sorted lists.
on board
Quicksort: Divide & Conquer

Divide
lists using a pivot
element from the list:
(a) items < pivot
(b) items > pivot
Sort (a) and (b)
recursively.

Merge
sorted lists:
(a) + [pivot] + (b)