CS218 - Data Structures
FAST NUCES Peshawar Campus
Dr. Nauman (recluze.net)

October 21, 2019

1 Sorting

Raster images of the notebook 15-sorts.

Bubble Sort

In [3]:
```
def bubble_sort(l):
    n = len(l)
    # print(n)

    # Outer loop. Goes over the whole thing `n' times
    # (because each time, one `highest' will have moved to the end)
    for i in range(n):

        # try to bubble the highest one up
        for j in range(0, (n-i)-1):

            # compare pairs, move higher one up (the highest will always reach the end this way!)
            if l[j] > l[j+1]:
                l[j], l[j+1] = l[j+1], l[j]
```

In [4]:
```
l = [1, 2, 4, 1, 2, 5, 5, 6, 1, 110, 15]
bubble_sort(l)
print(l)
```

```
[1, 1, 1, 2, 2, 4, 5, 5, 6, 15, 110]
```

Insertion Sort

In [5]:
```
def insertion_sort(l):

    # Go through all elements (except first).
    # Call it `key'
    # Each time, the key would be `inserted' in its place
    # At each iteration, stuff less than i would be sorted already

    for i in range(1, len(l)):

        key = l[i]   # hold this key

        # start comparing keys to things on its left!
        # stop when less or equal value found (or we reach left end)

        j = i-1
        while j >= 0 and key < l[j] :
                l[j+1] = l[j]   # move this to right. Slot left on j
                j -= 1

        l[j+1] = key  # Place key in free slot ... (j+1 because we decremented j above)
```

In [6]:
```
l = [1, 2, 4, 1, 2, 5, 5, 6, 1, 110, 15]
insertion_sort(l)
print(l)
```

```
[1, 1, 1, 2, 2, 4, 5, 5, 6, 15, 110]
```

Selection Sort

In [ ]:
```
def selection_sort(l):
    n = len(l)

    # for each element in the list (starting from left)
    for i in range(n):
        min_idx = i    # find the minimum ...

        # .... in the *rest* of the list
        for j in range(i+1, n):

            if l[j] < l[min_idx]:
                min_idx = j

        # swap the minimum with current element, now we have (sorted stuff till i)
        l[i], l[min_idx] = l[min_idx], l[i]
```

In [ ]:
```
l = [1, 2, 4, 1, 2, 5, 5, 6, 1, 110, 15]
selection_sort(l)
print(l)
```

Quick Sort

In [ ]:
```
import random

def qsort(l, fst, lst):
    if fst >= lst: return

    i, j = fst, lst
    pivot = l[random.randint(fst, lst)]

    while i <= j:
        while l[i] < pivot: i += 1
        while l[j] > pivot: j -= 1
        if i <= j:
            l[i], l[j] = l[j], l[i]
            i, j = i + 1, j - 1
    qsort(l, fst, j)
    qsort(l, i, lst)
```

In [ ]:
```
l = [1, 2, 4, 1, 2, 5, 5, 6, 1, 110, 15]
qsort(l, 0, len(l)-1)
```

In [ ]:
```
print(l)
```

Sorting in the Real World

Descending Sorts

In [ ]:
```
def selection_sort(l):
    n = len(l)

    # for each element in the list (starting from left)
    for i in range(n):
        idx = i    # find the replacement ...

        # .... in the *rest* of the list
        for j in range(i+1, n):

            if l[j] > l[idx]:    # changed to > for descending
                idx = j

        # swap the replacement with current element, now we have (sorted stuff till i)
        l[i], l[idx] = l[idx], l[i]
```

In [ ]:
```
l = [1, 2, 4, 1, 2, 5, 5, 6, 1, 110, 15]
selection_sort(l)
print(l)
```

But there's a better way!

In [ ]:
```
def less_than(a, b):
    return a < b
```

In [ ]:
```
def selection_sort(l, compare_with):
    n = len(l)

    # for each element in the list (starting from left)
    for i in range(n):
        min_idx = i    # find the minimum ...

        # .... in the *rest* of the list
        for j in range(i+1, n):

            if compare_with(l[j], l[min_idx]):      # now the "comparison" is out-sourced!
                min_idx = j

        # swap the minimum with current element, now we have (sorted stuff till i)
        l[i], l[min_idx] = l[min_idx], l[i]
```

In [ ]:
```
l = [1, 2, 4, 1, 2, 5, 5, 6, 1, 110, 15]
selection_sort(l, less_than)
print(l)
```

In [ ]:
```
def greater_than(a, b):
    return a > b
```

In [ ]:
```
l = [1, 2, 4, 1, 2, 5, 5, 6, 1, 110, 15]
selection_sort(l, greater_than)
print(l)
```

Taking it Even Further

Now we can do all sorts of stuff with this without making a single change to our selection sort code.

In [ ]:
```
all_tuples = [ (24, 25),
      (1, 2),
      (2, 4),
      (3, 5)]
```

In [ ]:
```
def tuple_less_than(a, b):
    return sum(a) < sum(b)
def tuple_greater_than(a, b):
    return sum(a) > sum(b)
```

In [ ]:
```
print("Ascending:\t", end="")
selection_sort(all_tuples, tuple_less_than)
print(all_tuples)

print("Descending:\t", end="")
selection_sort(all_tuples, tuple_greater_than)
print(all_tuples)
```

Sorting in Python

If you have a list of dictionaries -- each representing a student, for instance.

In [ ]:
```
d = [
     { 'name': 'khalid',  'age': 5  },
     { 'name': 'usman',   'age': 7  },
     { 'name': 'ali',     'age': 12 },
     { 'name': 'farooq',  'age': 3  },
]
```

In [ ]:
```
def d_less_than(a, b):
    return a['age'] < b['age']

selection_sort(d, d_less_than)
print(d)
```

In [ ]:
```
def student_age(a):
    return a['age']
```

In [ ]:
```
print(d)
d.sort(key=student_age, reverse=True)
print(d)
```

Sorting Objects of Custom Classes

In [ ]:
```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return self.name + ': ' + str(self.age)
```

In [ ]:
```
s1 = Student('Wajid', 5)
s2 = Student('Usman', 7)
s3 = Student('Ali', 3)

s = [s1, s2, s3]
```

You don't even have to give the function a name -- just use an anonymous function like so:

In [ ]:
```
for i in s:
    print(i)
```

Anonymous Function

In [ ]:
```
s.sort(key=lambda x: x.age)    # reverse
```

In [ ]:
```
for i in s:
    print(i)
```