Scientific Programming Practical 14

Introduction

Luca Bianco - Academic Year 2018-19
luca.bianco@fmach.it

Sorting algorithms

Given an input sequence (U) of un-sorted elements ***U=u*1,*u*2,...,*un*** produce a new sequence ***S=s*1,*s*2,...,*sn*** which is a **permutation of the elements in U** such that *s*1≤*s*2,...,≤*sn.*

Sorting algorithms

Given an input sequence (U) of un-sorted elements ***U*=*u*1,*u*2,...,*un*** produce a new sequence ***S*=*s*1,*s*2,...,*sn*** which is a **permutation of the elements in U** such that $s1 \leq s2,...,\leq sn.$

Today we will practice:

Selection sort
Insertion sort

Selection sort

The idea of **selection sort** is that given U=u1,u2,...,un the algorithm **loops through all the elements of U**, **finds the minimum** and **places it at the beginning of the sequence** U. The algorithm continues looking for the **minimum starting from u2** and so on.

Selection sort

If U has n elements, for each position $i = 0, \ldots, n-1$ in the list we need to perform the following two steps:

1. (argmin) Find index of the minimum element in the sublist $U[i+1:]$, let's call it m (i.e. $u_m = min(U[i:])$);
2. (swap) Swap u_m with u_i;

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=0$	7	4	2	1	8	3	5
$i=1$	1	4	2	7	8	3	5
$i=2$	1	2	4	7	8	3	5
$i=3$	1	2	3	7	8	4	5
$i=4$	1	2	3	4	8	7	5
$i=5$	1	2	3	4	5	7	8
$i=6$	1	2	3	4	5	7	8

Selection sort

If U has n elements, for each position $i = 0, \ldots, n-1$ in the list we need to perform the following two steps:

1. (argmin) Find index of the minimum element in the sublist $U[i+1:]$, let's call it m (i.e. $u_m = min(U[i:])$);
2. (swap) Swap u_m with u_i;

Argmin can be computed moving to the right
and then accumulating the minimum (and index)

Swap (m,j):

tmp = U[m]
U[m] = U[j]
U[j] = tmp

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=0$	7	4	2	1	8	3	5
$i=1$	1	4	2	7	8	3	5
$i=2$	1	2	4	7	8	3	5
$i=3$	1	2	3	7	8	4	5
$i=4$	1	2	3	4	8	7	5
$i=5$	1	2	3	4	5	7	8
$i=6$	1	2	3	4	5	7	8

Selection sort

If U has n elements, for each position $i = 0, \ldots, n-1$ in the list we need to perform the following two steps:

1. (argmin) Find index of the minimum element in the sublist $U[i+1:]$, let's call it m (i.e. $u_m = min(U[i:])$);
2. (swap) Swap u_m with u_i;

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=0$	7	4	2	1	8	3	5
$i=1$	1	4	2	7	8	3	5
$i=2$	1	2	4	7	8	3	5
$i=3$	1	2	3	7	8	4	5
$i=4$	1	2	3	4	8	7	5
$i=5$	1	2	3	4	5	7	8
$i=6$	1	2	3	4	5	7	8

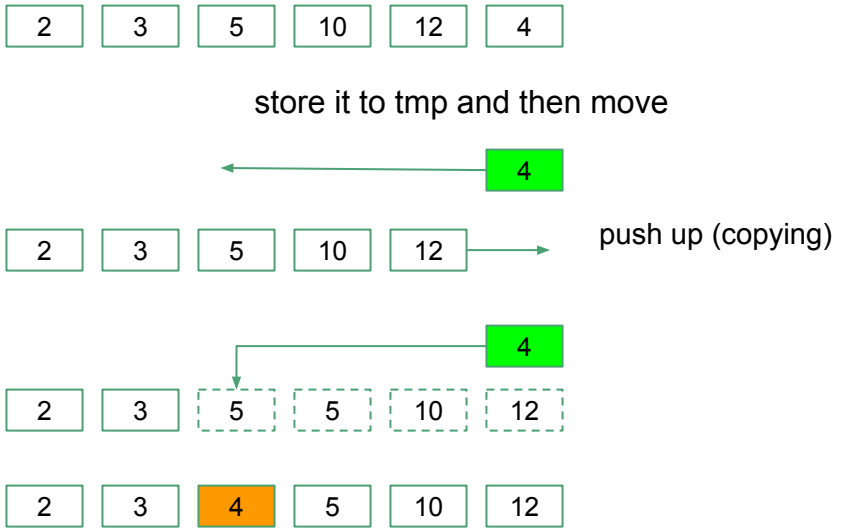
Simple algorithm, but complexity is O(n^2), where n is the length of the list.

Insertion sort

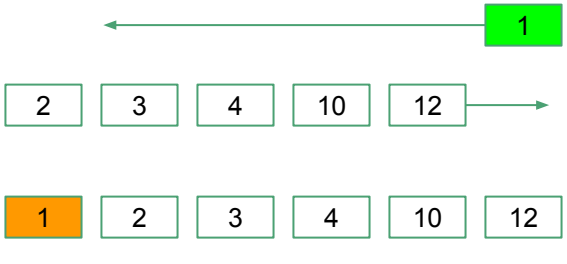
The idea of **insertion sort** is to build a sorted list step by step. In each step, one element is placed in its correct position on the left-side part of the array.

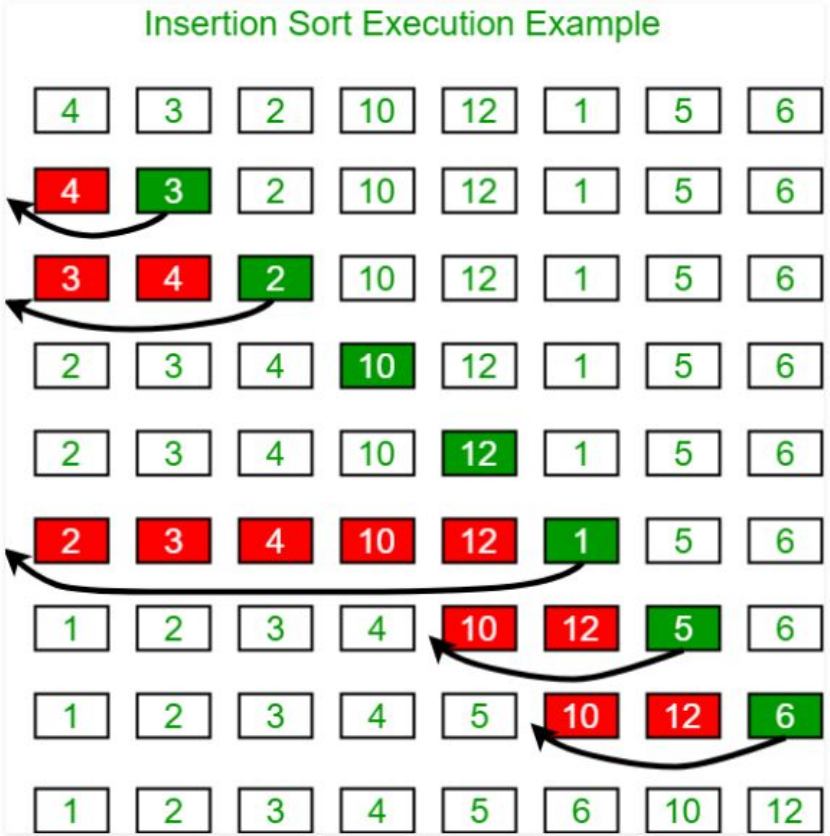
At each iteration (i):

U[i] → temp
for j: i-1,...,0
if U[j] < U[i]
copy U[j] →U[j+1]
U[j] ← U[i].

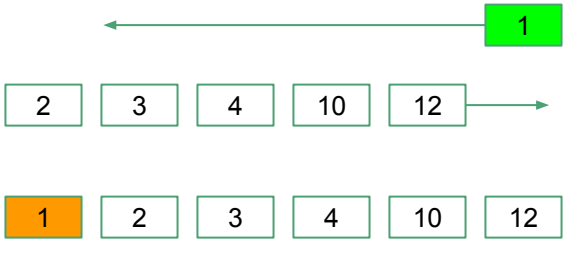


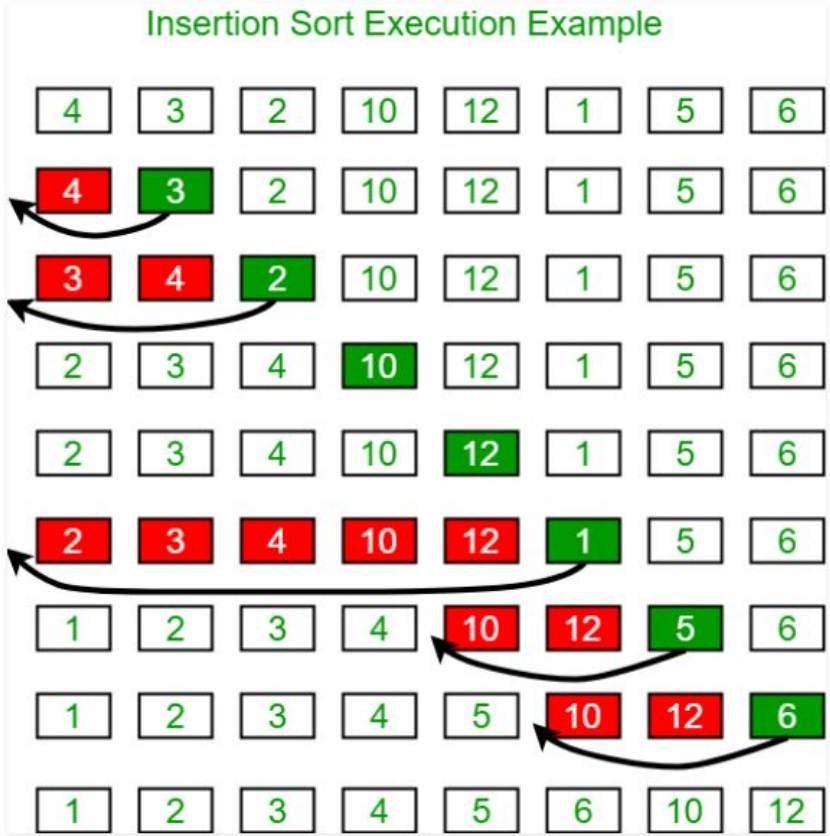
Insertion sort





Insertion sort





Complexity is O(n^2), where n is the length of the list.
If one element is no more than k places away from its correct place in the sorted array, complexity: O(kn).

http://qcbprolab.readthedocs.io/en/latest/practical14.html

Exercises

1. Implement a class SelectionSort (in a file called `SelSort.py`) that has one attribute called `data` (the actual data to sort), `operations` (initialized to 0) that counts how many swaps have been done to perform the sorting, `comparisons` (initialized to 0) that counts how many comparisons have been done and `verbose` a boolean (default= True) that is used to decide if the method should report what is happening at each step and some stats or not. The class has one method called `sort` that implements the selection sort algorithm (two more methods might be needed to compute `swap` and `argmin` – see description above).

Once you implemented the class you can test it with some data like:

```
[7, 5, 10, -11 ,3, -4, 99, 1]
```

or you can create a random list of N integers with:

```
import random
for i in range(0,N):
        d.append(random.randint(0,1000))
```

Test the class wit N = 10000 Add a private `__time` variable that computes the time spent doing the sorting. This can be done by:

```
import time
...
start_t = time.time()
...
end_t = time.time()
self._time = end_t - start_t
```

SelectionSort class

```
"""file: SelSort.py"""

import random
import time

class SelectionSort:
    def __init__(self,data, verbose = True):
        self.__data = data
        self.__comparisons = 0
        self.__operations = 0
        self.__verbose = verbose
        self.__time = 0

    def getData(self):
        return self.__data

    def getTime(self):
        return self.__time

    def getOperations(self):
        return self.__operations

    def getComparisons(self):
        return self.__comparisons
```

```
def swap(self, i, j):
    """
    swaps elements i and j in data.
    """
    ...

def argmin(self, i):
    """
    returns the index of the smallest element of
    self.__data[i:]
    """
    ...

def sort(self):
    self.__comparisons = 0
    self.__operations = 0

    start_t = time.time()
    ...
    end_t = time.time()
```

InsertionSort class

```
"""file: InsSort.py"""

import random
import time

class InsertionSort:
    def __init__(self,data, verbose = True):
        self.__data = data
        self.__comparisons = 0
        self.__operations = 0
        self.__verbose = verbose
        self.__time = 0

    def getData(self):
        return self.__data

    def getTime(self):
        return self.__time

    def getComparisons(self):
        return self.__comparisons

    def getOperations(self):
        return self.__operations
```

```
def sort(self):
    self.__comparisons = 0
    self.__operations = 0

    start_t = time.time()
    ...
    end_t = time.time()
```