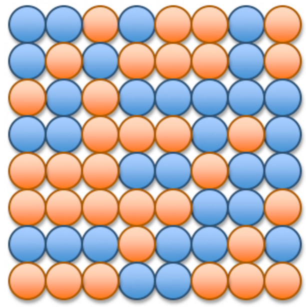
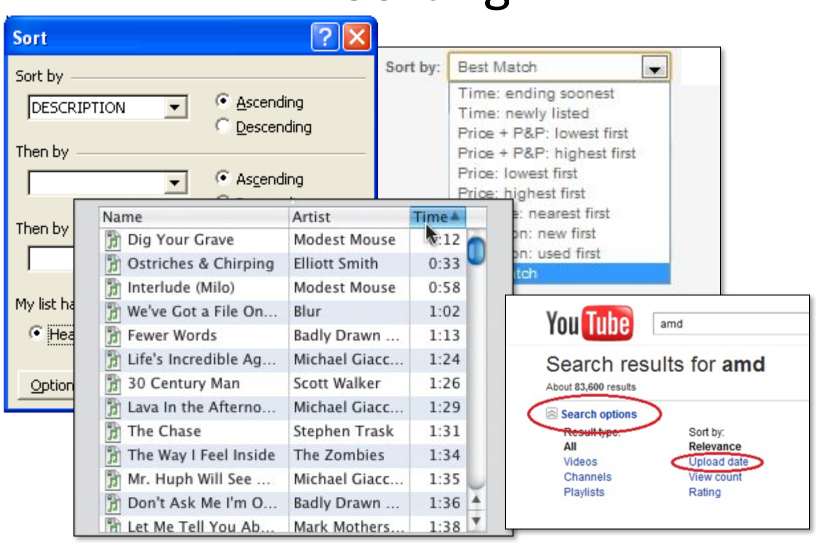


UNIT 4B
Iteration: Sorting

Sorting



Insertion Sort Outline

```
def isort(list)
  result = [ ]
  for val in list:
    # insert val in its proper
    #  place in result
    return result
```


insert function

```
list.insert(position, value)
```

```
>>> a = [10, 30, 20]
>>> a
[10, 30, 20]
>>> a.insert(0,"sna")
>>> a
["sna", 10, 30, 20]
```

insert function (cont'd)

```
>>> a.insert(2, “foo”)
>>> a
[“sna”, 10, “foo”, 30, 20]
>>> a.insert(5, “bar”)
>>> a
[“sna”, 10, “foo”, 30, 20, “bar”]
```


Insertion Sort, Refined

```
def isort (list)
  result = [ ]
  for val in list:
    # compute place to insert
    result.insert(place, val)
  return result
```

How do we find the right place to insert?

gr_index

```
# index of first element greater than item
def gr_index(list, item):
    index = 0
    while index < len(list) and   \
            list[index] < item:
        index = index + 1
    return index
```


Testing gr_index

```
>>> a = [10, 20, 30, 40, 50]
>>> a
[10, 20, 30, 40, 50]
>>> gr_index(a, 3)
0
>>> gr_index(a, 14)
1
>>> gr_index(a, 37)
3
>>> gr_index(a, 99)
5
```

Insertion Sort, Complete

```
def isort (list)
  result = [ ]
  for val in list:
    place = gr_index(result, val)
    result.insert(place, val)
  return result
```


Debugging Insertion Sort

```
def isort (list)
  result = [ ]
  print(result)     # for debugging
  for val in list:
    place = gindex(result, val)
    result.insert(place, val)
    print(result)   # for debugging
  return result
```

Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
```


Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
```

Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
[1, 3]
```


Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
[1, 3]
[1, 3, 4]
```

Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
[1, 3]
[1, 3, 4]
[1, 1, 3, 4]
```


Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
[1, 3]
[1, 3, 4]
[1, 1, 3, 4]
[1, 1, 3, 4, 5]
```

Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
[1, 3]
[1, 3, 4]
[1, 1, 3, 4]
[1, 1, 3, 4, 5]
[1, 1, 3, 4, 5, 9]
```


Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
[1, 3]
[1, 3, 4]
[1, 1, 3, 4]
[1, 1, 3, 4, 5]
[1, 1, 3, 4, 5, 9]
[1, 1, 2, 3, 4, 5, 9]
```

Testing isort

```
>>> isort([3, 1, 4, 1, 5, 9, 2, 6])
[]
[3]
[1, 3]
[1, 3, 4]
[1, 1, 3, 4]
[1, 1, 3, 4, 5]
[1, 1, 3, 4, 5, 9]
[1, 1, 3, 4, 5, 6, 9]
[1, 1, 2, 3, 4, 5, 6, 9]
=> [1, 1, 2, 3, 4, 5, 6, 9]
```


Can We Do Better?

- isort doesn't change its input list.
- Instead it makes a new list, called result.
- This takes twice as much memory.
- Can we write a destructive ("in place") version of the algorithm that doesn't use extra memory?
- That is the version shown in the book (see chapter 4).

Destructive (In Place) Insertion Sort

Given a list L of length n, n > 0.

1. Set i = 1.
2. While i is not equal to n, do the following:
   a. Insert L[i] into its correct position in L between indices 0 and i inclusive.
   b. Add 1 to i.
3. Return the list L which will now be sorted.

Example

```
L = [53, 26, 76, 30, 14, 91, 68, 42]
i =      1
```

Insert L[1] into its correct position in L between indices 0 and 1 inclusive and then add 1 to i:

53 moves to the right,

26 is inserted back into the list

```
L = [26, 53, 76, 30, 14, 91, 68, 42]
i =          2
```

Example

```
L = [26, 53, 76, 30, 14, 91, 68, 42]
i =             2
```

Insert L[2] into its correct position in L between indices 0 and 2 inclusive and then add 1 to i:

76 is already in the correct place

```
L = [26, 53, 76, 30, 14, 91, 68, 42]
i =                 3
```


Example

```
L = [26, 53, 76, 30, 14, 91, 68, 42]
i =                 3
```

Insert L[3] into its correct position in L between indices 0 and 3 inclusive and then add 1 to i:

76 moves to the right, then 53 moves to the right,

now 30 is inserted back into the list

```
L = [26, 30, 53, 76, 14, 91, 68, 42]
i =                     4
```

Look Closer at Insertion Sort

Given a list L of length n, n > 0.

1. Set i = 1.
2. While i is not equal to n, do the following:

   Precondition for each iteration: L[0..i) is sorted

   a. Insert L[i] into its correct position in L between index 0 and index i inclusive.

   b. Add 1 to i.

   Postcondition for each iteration: L[0..i) is sorted
3. Return the list L which will now be sorted.

Look Closer at Insertion Sort

Given a list L of length n, n > 0.

1. Set i = 1.
2. While i is not equal to n, do the following:

   Loop invariant: L[0..i) is sorted

   a. Insert L[i] into its correct position in L between index 0 and index i inclusive.

   b. Add 1 to i.
3. Return the list L which will now be sorted.

A loop invariant is a condition that is true at the start and end of each iteration of a loop.

Example (cont' d)

L = [26, 30, 53, 76, 14, 91, 68, 42]
i = 4

Insert L[4] into its correct position in L between indices 0 and 4 inclusive and then add 1 to i:

76 moves to the right, then 53 moves to the right,

then 30 moves to the right, then 26 moves to the right,

now 14 is inserted back into the list

L = [14, 26, 30, 53, 76, 91, 68, 42]
i = 5

Example

L = [14, 26, 30, 53, 76, 91, 68, 42]
i = 5

Insert L[5] into its correct position in L between indices 0 and 5 inclusive and then add 1 to i:

91 is already in its correct position

L = [14, 26, 30, 53, 76, 91, 68, 42]
i = 6

Example

```
L = [14, 26, 30, 53, 76, 91, 68, 42]
i =                           6
```

Insert L[6] into its correct position in L between indices 0 and 6 inclusive and then add 1 to i:

91 moves to the right,

76 moves to the right,

now 68 is inserted back into the list

```
L = [14, 26, 30, 53, 68, 76, 91, 42]
i =                               7
```


Example

```
L = [14, 26, 30, 53, 68, 76, 91, 42]
i =                               7
```

Insert L[7] into its correct position in L between indices 0 and 7 inclusive and then add 1 to i:

91 moves to the right, then 76 moves to the right,

then 68 moves to the right, then 53 moves to the right,

then 42 is inserted back into the list

```
L = [14, 26, 30, 42, 53, 68, 76, 91]
i =                                   8
```

Example

```
a = [14, 26, 30, 42, 53, 68, 76, 91]
i = 8
```

The list is sorted.

But how do we know that the algorithm always sorts correctly?

Reasoning with the Loop Invariant

The loop invariant is true at the end of each iteration, including the last iteration. After the last iteration, when we go to step 3:

L[0..i) is sorted AND i is equal to n

These 2 conditions imply that L[0..n) is sorted, but this range covers the entire list, so the list must always be sorted when we return our final answer!

Insertion Sort in Python

```
def isort(list):
     i = 1
     while i < len(list):
          move_left(list, i)
          i = i + 1
     return list
```

← insert list[i] into list[0..i] in its correct sorted position

15110 Principles of Computing, Carnegie Mellon University - CORTINA

33

Moving left

To move the element x at index i "left" to its correct position, start at position i-1, and search left until we find the first element that is less than x.

Then insert x back into the array to the right of the first element that is less than x when you searched from right to left in the sorted part of the array.

(The insert operation does not overwrite. Think of it as "squeezing into the array".)

Can you think of a special case for the step above?

15110 Principles of Computing, Carnegie Mellon University - CORTINA

34

Moving left: examples

Insert 68:

a = [14, 26, 30, 53, 76, 91, 68, 42]

Searching from right to left starting with 91, the first element less than 68 is 53. Insert 68 to the right of 53.

Insert 76:

a = [26, 53, 76, 30, 14, 91, 68, 42]

Searching from right to left starting with 53, the first element less than 76 is 53. Insert 76 to the right of 53 (where it was before).

Insert 14: SPECIAL CASE

a = [26, 30, 53, 76, 14, 91, 68, 42]

Searching from right to left starting with 76, all elements left of 14 are greater than 14. Insert 14 into the position 0.

move_left in Python

```
def move_left(list, i):
    x = list.pop(i)
    j = i - 1
    while j >= 0 and list[j] > x:
        j = j - 1
    list.insert(j + 1, x)
```

remove the item at position `i` in `list` and store it in `x`

insert `x` at position `j+1` of `list`, shifting all elements from `j+1` and beyond over one position

Insertion Sort, completed

```
def move_left(list, i):
      x = list.pop(i)
      j = i - 1
      while j >= 0 and list[j] > x:
            j = j – 1
      list.insert(j + 1, x)

def isort(list):
      i = 1
      while i < len(list):
            move_left(list, i)
            i = i + 1
      return list
```