CSC108H Lecture 33

Dan Zingaro

November 28, 2012

Selection Sort: the Code (selection_sort.py)

```
def find_min(L, i):
    '''(list, int) -> int
    Return the index of the smallest item in L[i:].
    '''
    smallest_index = i
    for j in range(i + 1, len(L)):
        if L[j] < L[smallest_index]:
            smallest_index = j
    return smallest_index

def selection_sort(L):
    '''(list) -> NoneType
    Sort the elements of L in non-descending order.
    '''
    for i in range(len(L) - 1):
        smallest_index = find_min(L, i)
        L[smallest_index], L[i] = L[i], L[smallest_index]
```

ConcepTest

Which of the following is true of **insertion** sort?

- A. Once a value is placed in the sorted part, it will never move again
- B. All values in the sorted part are always less than or equal to all values in the unsorted part
- C. Both of the above are true
- D. None of the above is true

ConcepTest

`[10, 20, 30, 40, 16, 94, 8, 22]`

The list above reflects the state of the list after 3 passes of insertion sort. What will be the list after the next (fourth) pass?

- A. `[8, 20, 30, 40, 16, 94, 10, 22]`
- B. `[10, 16, 20, 30, 40, 94, 8, 22]`
- C. `[10, 16, 30, 40, 20, 94, 8, 22]`
- D. `[8, 10, 20, 30, 40, 16, 94, 22]`
- E. `[10, 20, 30, 40, 8, 94, 16, 22]`

ConcepTest

`[5, 7, 14, 19, 16, 2, 32, 9]`

The list above reflects the state of the list after 3 passes of insertion sort. What will be the list after the next (fourth) pass?

- A. `[5, 7, 14, 16, 19, 2, 32, 9]`
- B. `[5, 7, 14, 19, 2, 16, 32, 9]`
- C. `[5, 7, 16, 19, 14, 2, 32, 9]`
- D. `[2, 5, 7, 14, 19, 16, 32, 9]`
- E. `[2, 7, 14, 19, 16, 5, 32, 9]`

Insertion Sort: Complication

When writing code for insertion sort, we run into a problem.
`[10, 20, 30, 40, 16]`

- We know that the 16 should go at index 1
- But we can't just put 16 there, because it would overwrite the 20
- What we do is shift each sorted element to the right until the place for 16 is found

```
[10, 20, 30, 40, x]
[10, 20, 30, x, 40]
[10, 20, x, 30, 40]
[10, x, 20, 30, 40]
```

Insertion Sort: the Code (insertion_sort.py)

```
def insert(L, i):
  '''(list, int) -> NoneType
  Move L[i] to where it belongs in L[:i].
  '''
  v = L[i]
  while i > 0 and L[i - 1] > v:
    L[i] = L[i - 1]
    i -= 1
  L[i] = v

def insertion_sort(L):
  '''(list) -> NoneType
  Sort the elements of L in non-descending order.
  '''
  for i in range(1, len(L)):
    insert(L, i)
```

Bubble Sort

- Bubble sort divides the list into an unsorted part (initially the whole list) and a sorted part (initially empty)
- Unlike selection and insertion, our bubble sort has the sorted part at the right (not the left)
- Then, while our sorted part is not the whole list
  - Scan the unsorted part from left to right
  - When an element is larger than the one to its right, swap them
- This scan-and-swap constitutes one pass of bubble sort
- This is a lot of work done on each pass!

Bubble Sort: Example

- List: **8 10 3 5 1**
- The work done on the first pass
  - - 8 10 - 3 5 1
  - 8 - 10 3 - 5 1
  - 8 3 - 10 5 - 1
  - 8 3 5 - 10 1 -
  - 8 3 5 1 10
- So, after one pass: **8 3 5 1** 10

Bubble Sort: Example...

- List: **8 3 5 1** 10
- The work done on the second pass
  - - 8 3 - 5 1 10
  - 3 - 8 5 - 1 10
  - 3 5 - 8 1 - 10
  - 3 5 1 8 10
- So, after two passes: **3 5 1** 8 10

ConcepTest

Which of the following is true of **bubble** sort?

- A. Once a value is placed in the sorted part, it will never move again
- B. There is never a value in the sorted part that is smaller than some value in the unsorted part
- C. Both of the above are true
- D. None of the above is true

Bubble Sort: the Code (bubble_sort.py)

```
def bubble_sort(L):
  '''(list) -> NoneType
  Sort the elements of L in non-descending order.
  '''
  for i in range(len(L) - 1):
    for j in range(len(L) - i - 1):
      if L[j] > L[j + 1]:
        L[j], L[j + 1] = L[j + 1], L[j]
```