Testing Algorithms

Announcements

- Lab 8 due tomorrow at midnight
  - Ninja session tonight, 7-9pm
- Lab 9 (sorting) posted on Sunday
- Quiz 5 next Friday
  - Includes searching and TDD, but not sorting

Today's Plan

- Review selection sort and bubble sort
- Insertion sort
- Testing, timing, and importing algorithms
- Compare sorting algorithms

Review: Bubble sort

- Makes a series of passes through the list, swapping pairs of consecutive values if the ‘left’ value is bigger than the ‘right’.

- Each pass ‘bubbles’ the biggest remaining unsorted value to its final position.

- Once a pass of the list yields no swaps, we know the list is sorted.

- $O(n^2)$ run time

```
def bubbleSort(L):
    n = len(L)
    made_swap = True
    while made_swap:
        made_swap = False
        for j in range(n-1):
            if L[j] > L[j+1]:
                swap(L, j, j+1)
                made_swap = True
```

Review: Selection sort

- For each index in the unsorted list, “select” the value that will end up there in the sorted list, and swap it into position.

- To do this selection look for the smallest value among those that haven’t yet been swapped into position.

- $O(n^2)$ run time

```
def findIndexOfMin(L, i):
  """
  Purpose: Find the index of the smallest value in L,
           not including values before index i
  Paramters: L - a list of values that can be ordered
             i - index where we start looking for minimum
  Returns: index of the minimum value in L, starting at i
  """
  indexOfMin = i
  for j in range(i+1, len(L)):
    if L[j] < L[indexOfMin]:
      indexOfMin = j
  return indexOfMin

def selectionSort(L):
  for i in range(len(L)-1):
    indexOfMin = findIndexOfMin(L, i)
    swap(L, i, indexOfMin)
```

Insertion sort

- For each index, `i`, from 1 to the end:
    - Compare `L[i]` with the value on its left, `L[i-1]`. If `L[i]` is smaller, swap these two values. Continue swapping `L[i]` to the left until it's bigger than the value on its left.

Insertion sort

- Insertion sort works because after `n` repetitions of the outer for loop, the first `n+1` values are sorted, even if they aren't in their final position.

- Then we "insert" the next value into its position in this sorted sublist.

Insertion sort implementation

Recap

- Test algorithms with many different kinds of input, use `assert` to verify that tests pass.

- "Protect" the call to main so the same code can be either run from the command line or imported:

```
if __name__ == "__main__":

    main()
```

- Use the `time()` function to get the current time in seconds.

Comparison

- If you know something about the inputs you're likely to get, it can influence your choice of algorithm, even if they all have the same big O run time.

- Selection: minimizes number of swaps

- Insertion: good for almost sorted lists and small lists

- Bubble: like insertion, but worse :(

- sorting-algorithms.com

Have a nice weekend!