Quick Sort

MAY 1ST, 2013

Quick Sort: Review

```
def generalQuickSort(L, first, last):
    # Base case: if first == last, then there is only one element in the
    # slice that needs sorting. So there is nothing to do.

    # Recursive case: if there are 2 or more elements in the slice L[first:last+1]
    if first < last:
        # Divide step: partition returns an index p such that
        # first <= p <= last and everthing in L[first:p] is <= L[p]
        # and everything in L[p+1:last+1] is >= L[p]
        p = partition(L, first, last)

        # Conquer step
        generalQuickSort(L, first, p-1)
        generalQuickSort(L, p+1, last)

        # Combine step: there is nothing left to do!
```

Quick Sort: Review
The partition function

```
def partition(L, first, last):
    # We pick the element L[first] as the "pivot" around which we partition the list
    p = first

    # We process the rest of the elements, one-by-one, in left-to-right order
    for current in range(p+1, last+1):

        # If L[current] is smaller than the pivot, it needs to move into the first block,
        # to the left of the pivot.
        if L[current] < L[p]:
            swap(L, current, p+1)
            swap(L, p, p+1)
            p = p + 1

    return p
```

Quick Sort: Review
The wrapper function and the swap function

```
def quickSort(L):
    generalQuickSort(L, 0, len(L)-1)


def swap(L, i, j):
    temp = L[i]
    L[i] = L[j]
    L[j] = temp
```

Partition Function in Action

- **Initial list:** [6, 2, 4, 1, 6, 10, 2, 11, 8, 7]
- 6 is selected as the pivot.

[ ] 6 [ ]
[2] 6 [ ]
[2, 4] 6 [ ]
[2, 4, 1] 6 [ ]
[2, 4, 1] 6 [6]
[2, 4, 1] 6 [6, 10]
[2, 4, 1, 2] 6 [10, 6]
[2, 4, 1, 2] 6 [10, 6, 11]
[2, 4, 1, 2] 6 [10, 6, 11, 8]
[2, 4, 1, 2] 6 [10, 6, 11, 8, 7]

- **Final list:** [2, 4, 1, 2, 6, 10, 6, 11, 8, 7]
- Function returns index 4

The partition may be skewed!

- **Initial list:** [16, 2, 4, 1, 6, 10, 2, 11, 8, 7]

  [ ] 16 [ ]
  [2] 16 [ ]
  [2, 4] 16 [ ]
  [2, 4, 1] 16 [ ]
  [2, 4, 1, 6] 16 [ ]
  [2, 4, 1, 6, 10] 16 [ ]
  [2, 4, 1, 6, 10, 2] 16 [ ]
  [2, 4, 1, 6, 10, 2, 11] 16 [ ]
  [2, 4, 1, 6, 10, 2, 11, 8] 16 [ ]
  [2, 4, 1, 6, 10, 2, 11, 8, 7] 16 [ ]

- **Final list:** [2, 4, 1, 6, 10, 2, 11, 8, 7, 16]
- Function returns index 9

Quick Sort in Action

The initial list is: [6, 2, 4, 11, 6, 10, 2]

[2, 4, 2] [6] [6, 10, 11] (partition on [6, 2, 4, 11, 6, 10, 2])

[ ] [2] [4, 2] (partition on [2, 4, 2])

[2] [4] [ ] (partition on [4, 2])

[ ] [6] [10, 11] (partition on [6, 10, 11])

[ ] [10] [11] (partition on [10, 11])

The sorted list is: [2, 2, 4, 6, 6, 10, 11]

Running Time Comparison

- On lists with 100,000 elements constructed at random. Selection sort took 5 minutes on lists of this size.

```
Finished constructing the lists...
Time for Merge Sort is:  0.678801059723
Time for Quick Sort is:  0.980933904648

Finished constructing the lists...
Time for Merge Sort is:  0.682029008865
Time for Quick Sort is:  0.987423181534

Finished constructing the lists...
Time for Merge Sort is:  0.67242193222
Time for Quick Sort is:  0.985061883926
```

Puzzle: A different Experiment

- The input list is [0, 1, 2, ...] of size 10,000.

```
Finished constructing the lists...
Time for Merge Sort is:  0.0404422283173
Time for Quick Sort is:  4.38273501396

Finished constructing the lists...
Time for Merge Sort is:  0.0395169258118
Time for Quick Sort is:  4.36549711227

Finished constructing the lists...
Time for Merge Sort is:  0.0384669303894
Time for Quick Sort is:  4.36951899529
```

- Why does quick sort take 100 times more time??

Solution

- For [0, 1, 2, 3, …, n-1], n units of work yields

  [ ] 0 [1, 2, 3, …, n-1]

- An additional n-1 units of work yields

  [ ] 1 [2, 3, …, n-1]

- An additional n-2 units of work yields

  [ ] 2 [3, 4, …, n-1]

So total work is n + (n-1) + (n-2) + … + 1, which is roughly $n^2/2$.

Ideal behavior

- n units of work yield

  [……] pivot [……]

- n units of work yield

  [……] pivot [……] [……] pivot [……]

- n units of work yield

  [...] pivot [...] [...] pivot [...] [...] pivot [...] [...] pivot [...]

- ….

We go down log(n) levels, for a total of *n log(n)* units of work.

How to pick a good pivot?

- Randomize! (Just pick a random element as the pivot, instead of the first element).

- Add this line of code at the beginning of partition:

```
swap(L, first, random.randint(first, last))
```

- Now the running times, even on a sorted input list are comparable:

```
Finished constructing the lists...
Time for Merge Sort is:  0.040990114212
Time for Quick Sort is:  0.0971350669861
```