

Team Project: Comparisons of Sorting Methods

- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quick Sort

Naive Sorting: Selection Sort

```
def selSort(nums):
    # sort nums into ascending order

    n = len(nums)

    # For each position in the list (except the very last)
    for bottom in range(n-1):
        # find the smallest item in nums[bottom]...nums[n-1]


        mp = bottom                # bottom is smallest initially
        for i in range(bottom+1, n): # look at each position
            if nums[i] < nums[mp]:    # this one is smaller
                mp = i               # remember its index

        # swap smallest item to the bottom
        nums[bottom], nums[mp] = nums[mp], nums[bottom]
```

Merge Sort in Python

```
def msort(list):
    if len(list) == 0 or len(list) == 1: # base case
        return list[:len(list)] # copy the input
    # recursive case
    halfway = len(list) // 2
    list1 = list[0:halfway]
    list2 = list[halfway:len(list)]
    newlist1 = msort(list1) # recursively sort left half
    newlist2 = msort(list2) # recursively sort right half
    newlist = merge(newlist1, newlist2)
    return newlist
```

```
def merge(a, b):
    index_a = 0 # the current index in list a
    index_b = 0 # the current index in list b
    c = []
    while index_a < len(a) and index_b < len(b):
        if a[index_a] <= b[index_b]:
            c.append(a[index_a])
            index_a = index_a + 1
        else:
            c.append(b[index_b])
            index_b = index_b + 1
    # when we exit the loop
    # we are at the end of at least one of the lists
    c.extend(a[index_a:])    c.extend(b[index_b:])
    return c
```



Depending on whether we are at the end of the list a or the list b, we execute one of the extend statements and return c.

Bubble Sort

http://en.wikipedia.org/wiki/Bubble_sort

Bubble sort is one of the most basic sorting algorithm that is the simplest to understand. It's basic idea is to bubble up the largest(or smallest), then the 2nd largest and the the 3rd and so on to the end of the list. Each bubble up takes a full sweep through the list.

Bubble Sort in Python

```
1  def bubble_sort(items):
2      """ Implementation of bubble sort """
3      for i in range(len(items)):
4          for j in range(len(items)-1-i):
5              if items[j] > items[j+1]:
6                  items[j], items[j+1] = items[j+1], items[j]      # Swap!
```

Insertion Sort

http://en.wikipedia.org/wiki/Insertion_sort

Insertion sort works by taking elements from the unsorted list and inserting them at the right place in a new sorted list. The sorted list is empty in the beginning. Since the total number of elements in the new and old list stays the same, we can use the same list to represent the sorted and the unsorted sections.

Insertion Sort in Python

```
1  def insertion_sort(items):
2      """ Implementation of insertion sort """
3      for i in range(1, len(items)):
4          j = i
5          while j > 0 and items[j] < items[j-1]:
6              items[j], items[j-1] = items[j-1], items[j]
7              j -= 1
```


Quick Sort

<http://en.wikipedia.org/wiki/Quicksort>

Quick sort works by first selecting a pivot element from the list. It then creates two lists, one containing elements less than the pivot and the other containing elements higher than the pivot. It then sorts the two lists and join them with the pivot in between. Just like the Merge sort, when the lists are subdivided to lists of size 1, they are considered as already sorted.

Quick Sort in Python

```
1  def quick_sort(items):
2      """ Implementation of quick sort """
3      if len(items) > 1:
4          pivot_index = len(items) // 2
5          smaller_items = []
6          larger_items = []
7
8          for i, val in enumerate(items):
9              if i != pivot_index:
10                 if val < items[pivot_index]:
11                     smaller_items.append(val)
12                 else:
13                     larger_items.append(val)
14
15             quick_sort(smaller_items)
16             quick_sort(larger_items)
17         items[:] = smaller_items + [items[pivot_index]] + larger_items
```

Test data and Time Measurement

```
1 import random
2
3 random_items = [random.randint(-50, 100) for c in range(32)]
4
5 print('Before: ', random_items)
6 insertion_sort(random_items)
7 print('After : ', random_items)
```

import time

startTime = time.time()

sort_function_x(random_items)

endTime = time.time()

elapsedTime = endTime - startTime

Print("The elapsed time for sort_function_x is: ", elapsedTime)

보고서 PPT에 있어야 하는 내용

- 5개의 sorting algorithm의 작동원리를 unsorted data item 6개를 가지고 알기쉽게 그림으로 보여주고
- Sorting algorithm 의 algorithm complexity를 설명하고
- 규모가 큰 데이터를 5개의 실제 sorting program에 돌려서 실행시간을 측정하여 algorithm complexity와 비교해서 설명