CPE202 Spring 2019

Lab 6 : Sorting Algorithms

Sorting Algorithm Comparisons on Sorted and not Sorted List

In this lab **you will work with one of your classmates** and compare **six different sorting algorithms**. Review the lecture slides and the textbook and write six functions that use six sorting algorithms (selection sort, insertion sort, quicksort, merge sort, bubble sort, and heap sort). (Before doing any comparison make sure, your code works!) Distribute the work evenly among your team members: Each of you should write at least one of quicksort and merge sort, and in case you end up with working with two other classmates (this could happen if we have odd number of students in the class), the third person should implement heap sort. Put the name of the author in the docstring of each function so that we know who implemented the function (to determine who has slacked off in case that indeed happens).

(Note: You are testing these sorting algorithms on integer numbers)

Comparison Sorting Visualizations link can help you to have some perspective on sort comparison: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Create a python file as **comparison_sort.py**, write your functions for sorting sorted and unsorted list in this file. Your goal for this lab is to implement simple versions of Insertion Sort - insertion_sort(alist), Selection Sort - selection_sort(alist), Merge Sort - merge_sort(alist), Quick Sort - quick_sort(alist), bubble sort - (bubble_sort), and heap sort - (heap_sort) that will sort an array of integers and **count the number of comparisons**. Each function takes a list of integers as input, sorts the list, counts the number of comparisons at the same time, **and returns the number of comparisons**. After the function completes, the "alist" should be sorted.

Do not do any improvement in the sort algorithms, which can cause reducing the number of comparison. Use the version of algorithms presented in the lecture. Make sure you are using the same list for all sort algorithms.

OPTIONAL: Implement one other version of bubble sort with a slight modification to stop sorting as soon as the list is sorted. To do so, you will set a flag to indicate that no swaps have happened in the previous pass so that the program can detect if the list has been already sorted and end. Name this version of bubble sort as "bubble_sort2".

The runtime complexity is $O(n^2)$ for selection and insertion sort; and O(n*log(n)) for merge and quick sort, Why? You will submit your answers and results of your code in a pdf file. Write out the summation that represents the number of comparisons. Note that you will need to run test cases of different sizes to show that Bubble, Insertion and Selection sorts are $O(n^2)$ and Merge, Quick, and Heap sorts are

O(n*log(n)). To do this, plot the number of comparisons (y-axis) vs. the size of the list (x-axis). Since the plot is shaped like a parabola for selection and insertion sort this indicates that it is quadratic. In the pdf file, for each sort, submit a table showing list size and number of comparisons from your test cases and a plot of that data.

Time the runtime of Python sort( ) function for given list. Python uses **Timsort algorithm** (it is a hybrid sort that derived from merge and insertion sort algorithms with runtime complexity of O(n) in best case and O(n*log n) in average and worst cases. (just use alist.sort() in your code and set the start and end time). The source code of the Timsort can be viewed here.

For generating the list you can create a list of random numbers.

```
import random

random.seed(1) #in order to generate the same sequence of numbers each time.

alist = random.sample(range(10000),10)
```

Which produce a list of 10 random number between 1 and 10000 [5445, 8692, 6915, 8637, 4848, 9408, 1744, 171, 4315, 2949]

For timing, you can use time class of Python

```
import time

start_time = time.time()


#Now call sort function


end_time = time.time()


sort_time = end_time – start_time
```

Submission:

Zip the following two files into one. Make sure to have the names of all members in your team on the files.

1) comparison_sort.py 2) Comparison_analysis.pdf

The pdf file must include:

1) Two tables for each sort algorithm.

2) Answer to the questions in the last page

3) Plot the number of comparisons (y-axis) vs. the size of the list (x-axis). Since the plot is shaped like a parabola for selection and insertion sort this indicates that it is quadratic. It may take several hours (4 ~ 9 hours) for some sorting algorithms to complete sorting 500,000 items.

Algorithm Name:	Type of list (sorted or unsorted):	
List Size	Comparisons	Time (seconds)
1,000 (observed)		
2,000 (observed)		
4,000 (observed)		
8,000 (observed)		
16,000 (observed)		
32,000 (observed)		
100,000 (observed)		
500,000 (observed)		

Answer the following questions:

1. Which sort do you think is better? Why?

2. Which sort is better when sorting a list that is already sorted (or mostly sorted)? Why?

3. You probably found that insertion sort had about half as many comparisons as selection sort. Why? Why are the times for insertion sort not half what they are for selection sort?