CS200L Data Structures and Algorithms (Pr)

Lab Manual (Week 2)

Instructor:

- Mr. Muhammad Waseem

Registration No._____

Name: _____

Guide Lines/Instructions:

- Use of Spyder IDE/ Anaconda is must in this lab.
- Create meaningful variable names. Add comments for readability. Indent each line of your code.
- Plagiarism/Cheating is highly discouraged by penalizing to both those who tried and one who shared his/her code.

Today's Task:

- Comparison of Sorting Algorithms

Helping Content:

Example 1: Calculate the running time of a function

The following code snippet will help you how to calculate the time for the execution of the function.

```
def factorial(n):
    if(n==0):
        return 1
    else:
        return n* factorial(n-1)

import time
start_time = time.time()
n = 1500
ans= factorial(n)
end_time = time.time()

runtime = end_time - start_time

print("Runtime of factorial at",n,"is",runtime,"seconds")
```

Problems

Problem1:

- Implement a function that takes the array and populate the array with random numbers. RandomArray(size) – returns the randoms array of length size

Problem2:

- Implement a function InsertionSort that takes array A and sort the array. Function should have the following prototype.
  `InsertionSort(array,start, end)`
- In the Insertion.py create a random array of 30,000 integers and sort it using insertion sort. Use the above two functions to solve the problem.
- Calculate the time, that InsertionSort function takes. Print the answer in seconds.
- Save the sorted array in SortedInsertionSort.csv. One integer at one line.

Problem 3:

- Implement a function MergeSort that takes array A and sorts the 1d array. Function should have the following prototype.
  - `MergeSort(array, int start, int end)`
  - `void Merge(array, p, q, r)`
- In the MergeSort.py, create a random array of 30,000 integers and sort it using merge sort. Use the MergeSort, Merge functions in addition to RandomArray function created in Task 1.
- Also calculate the time, that MergeSort function takes. Print the answer in seconds.
- Save the sorted array in SortedMergeSort.csv. One integer at one line.

Activity 1:

- Calculate the value of n_o, for which running time of merge sort is better than insertion sort. Complete the following table as well for the calculation. n is the number of integers.

Value of n	Insertion sort (seconds)	Merge Sort(seconds)
100		
1000		
5000		
10000		
20000		
30000		
40000		
50000		
60000		
100000		
500000		
1000000		

This table will help you to find the value of n.

Problem 4:

Use the value of n to create the hybrid Merge sort in which instead of dividing arrays to single element array, divide the array to the value of n, so that small arrays can be sorted using insertion sort.

- Implement a function `HybridMergeSort` that takes array A and sort the array. Function should have the following prototype.
  `HybridMergeSort(array, start, end)`
- In the HybridMerge.py, create a random array of 30,000 integers and sort it using hybrid sort algorithm. Use the above `HybridMergeSort` and RandomArray function to solve the problem.
- Save the sorted array in SortedHybridSort.csv. One integer at one line.

Problem 5:

Implement a function BubbleSort that takes array A and sort the array. Function should have the following prototype.

```
BubbleSort(array,start, end)
```

- In the Bubble.py create a random array of 30,000 integers and sort it using bubble sort. Use the above BubbleSort and RandomArray function to solve the problem.
- Calculate the time, that BubbleSort function takes. Print the answer in seconds.
- Save the sorted array in SortedBubbleSort.csv. One integer at one line.

Problem 6:

Implement a function SelectionSort that takes array A and sort the array. Function should have the following prototype.

```
SelectionSort(array,start, end)
```

- In the Selection.py create a random array of 30,000 integers and sort it using selection sort. Use the above SelectionSort and RandomArray function to solve the problem.
- Calculate the time, that SelectionSort function takes. Print the answer in seconds.
- Save the sorted array in SortedSelectionSort.csv. One integer at one line.

Activity 2:

Compare the running time of three algorithms and fill the following table.

Value of n	Insertion sort (seconds)	Merge Sort (seconds)	Hybrid Merge Sort (Seconds)	Selection Sort (Seconds)	Bubble Sort (Seconds)
100					
1000					
5000					
10000					
20000					
30000					
40000					
50000					
60000					
100000					
500000					
1000000					

Problem 7:

- Read the value of n from the file Nvalues.txt. Each value per line.(Example of file is shown towards right)
- You are required to complete the above table programmatically and save the result in RunTime.csv
- Use the required functions from funcs.py
- Write your code in Problem7.py

Nvalues.txt

```
100
10000
50000
70000
```

Problem 8:

- You are provided with words.txt.
- Read the text file

- Run InsertionSort and MergeSort on words.txt. Display the runtime for both algorithms on InsertionSort and MergeSort.
- Write a function ShuffleArray(array, start, end) in the funcs.py. Function will shuffle the given array randomly.
- Use ShuffleArray to shuffle the words array randomly.
- Now run InsertionSort and MergeSort on words array. Display the runtime for both algorithms on InsertionSort and MergeSort.
- Do you feel any difference, write your answer below?
- Write your code in Prob8.py

Things to Remember (Not following the instruction will result in ZERO MARKS):

- You have to calculate to time the algorithm takes to sort the array. This time should not include the time for the generation of random numbers and writing the numbers in sorted array
- For all the function, strictly follow the definition of the functions.
- If you do not follow the definition, your assignment will not be checked.
- Properly commented code and clean code will get more marks.
- Plagiarism will never be tolerated.

What to Submit:

1. Only .py files are allowed.
2. You are required to submit the following files.
    a. funcs.py
    b. Insertion.py
    c. MergeSort.py
    d. HybridMerge.py
    e. Bubble.py
    f. Selection.py
    g. Prob7.py
    h. Prob8.py
3. Functions names, input and output should be exactly same.
4. Zip all files, and submit on eduko