CS 102 – Data Structure and Algorithm
Habib University – Spring' 21
Assignment # 02
Due on : March 21 , 2021

Objective:

The objective of these assignments is to provide students an insight of Divide and Conquer techniques to sort a real-world dataset. The students will also analyze order of growth of different functions and compare time complexities of different code snippets.

Q 1 – Sorting Question: Document Distance Problem [30 points]

The document distance problem has applications in finding similar documents, detecting duplicates (Wikipedia mirrors and Google) and plagiarism, and in web search (D2 = query). The idea is to define distance in terms of shared words. Think of document D as a vector: D[w] = # occurrences of word W. Here, 'word' is a sequence of alphanumeric characters and 'document' is a sequence of words (ignore space, punctuation, etc.).

The distance between any two text files can be calculated as the angle between their word frequency vectors (in radians). For each input file, a word-frequency vector is computed as follows:

1. The specified file is read in.

2. It is converted into a list of alphanumeric "words". Here a "word" is a sequence of consecutive alphanumeric characters. Non-alphanumeric characters are treated as blanks. Case is not significant.

3. For each word, its frequency of occurrence is determined.

4. The word/frequency lists are sorted into order alphabetically.

Questions:

a) Write a function readFile(filename) that reads the text file with the given filename and returns a list of the lines of text in the file.

b) Write a function getWordsFromLineList(list) that parse the list of the lines of text into words and returns a list of all words found.

c) Write a function countFrequency(list) that computes the frequency of each word in the list and returns a list of tuples in the form: (word, frequency).

d) Implement a function mergeSort(list, column, ascending=True) that takes a list of tuples already loaded and uses Merge Sort to sort the data by the given column in ascending or descending order. For example, considering the list of tuples of the form: (word, frequency)

a. mergeSort(lst, 0, True) will sort the list by words in alphabetical order.
b. mergeSort(lst, 1, False) will sort the list by frequency in descending order.

e) Implement a function quickSort(list, column, ascending=True) that takes a list of tuples and uses Quick Sort to sort the data by the given column in ascending or descending order. The pivot element can be randomly selected.

f) Perform an experimental analysis of word-frequency vector computatkion to compare the effectiveness of different sorting algorithms. Use the implemented mergeSort(…) and quickSort(…) with three different pivot selection schemes for comparison on the provided text files file1.txt and file2.txt. **(Study textbook section 3.1 for experimental analysis. Visualize the running times with different sorting schemes for each provided text files; you may generate graphs for similar text files as a dataset for the performance analysis of your implementation).**

Q 2 – Count Inversions [15 Points]

Sometimes you are interested in finding how far an array is from being sorted. This can be found by counting number of inversions in an array. If the array is already sorted, then inversion count is 0. If the array is sorted in reverse order then inversion count is the maximum. Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

For example, the sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3) and its inversion count is 3.

Write a function CountInversions(lst) that takes an array of positive integers and returns number of inversions in it. You can certainly do this for $O(n^2)$ time. Think of a better solution that solves the problem in not more than O(nlogn) time.

Q 3 – Complexity [25 Points]

a) [10 points] Using Excel, plot all the functions below for different values of n from 1 to 100. Then arrange the following functions from the highest to the lowest order of growth in a table. You have to give both the plot and the table in this question.

$n-n^2+n^3$	$\log n^4$
2^{n-1}	6000
$\log n$	$(\log n)^2$
n^3	n
$n2^{\log n}$	$(3/2)^n$
$n\log n$	$\log\log n$

b) [15 points] Give the worst case Big-O runtime of code snippets given in part (a) – (e)? Explain your working.

Note: You may need to use some of the following formulas to compute complexity.

- Sum of first n natural numbers:

$$\sum_{\square=1}^{\square} \square = \frac{\square(\square+1)}{2}$$

- Sum of powers of 2:

$$2^0 + 2^1 + 2^2 + \cdots + 2^{\square} = 2^{\square+1} - 1$$

- Sum of first N even numbers

$$2 + 4 + 6 + 8 + \cdots + (2N - 1) = N\,(N + 1)$$

- Sum of first N odd number

$$1 + 3 + 5 + 7 + \cdots + (2N - 1) = N^2$$

- Sum of geometric Series

$$a + ar + ar2 + ar3 + \cdots + arn{-}1 = \text{a}\,\frac{1-r^n}{1-\text{r}}$$

a)

```
def exercise1(N):
     for i in range(0,N):
         for j in range(N,i,-1):
             a = a + i + j

         for j in range(0,N/2):
             b = b + i + j
```

b)

```
def exercise2(N):
     count = 0
     i = N
     while ( i > 0 ):
         for j in range(0,i):
             count = count + 1
         i = i//2
```

c)

```
def exercise3(arr):
     N = len(arr)
     for i in range(0,N):
         for j in range(0,N):
             binarySearch(arr,j)
         selectionSort(arr)
```

d)

```
def exercise4(L):
    N = len(L)
    s = []

    for i in range(N**2):
        s.append(L[i % N])

    return mergeSort(s)
```

e)

```
def exercise5(arr,N):
     counter =  1
     while counter <N:
         binarySearch(arr, counter)
         counter = counter *2
```