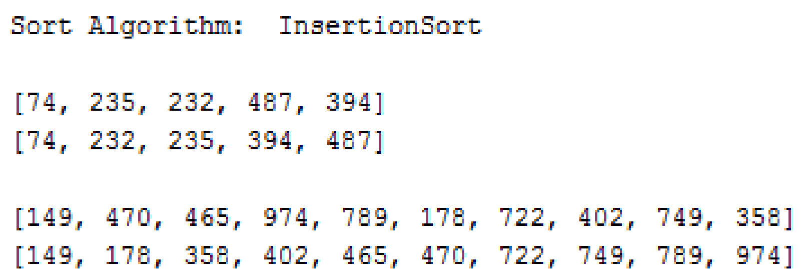
Praminda Mahesh Imaduwa-Gamage CMP3130-Fall 2017

Programming Assignment-2

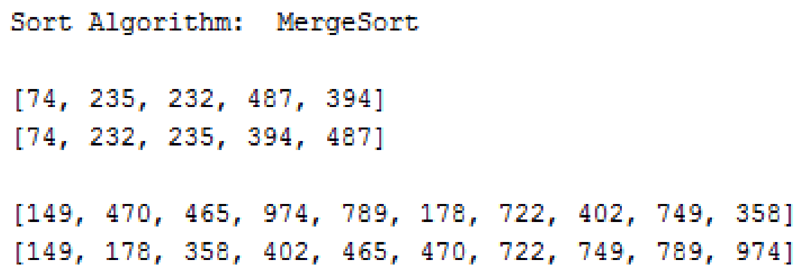
1. Write programs for quicksort, insertion sort and Merge sort. Compare the running time of these algorithms by taking random inputs/files of different sizes (at least 5). Do not use inbuild functions for implementing sorting algorithms.

Results:

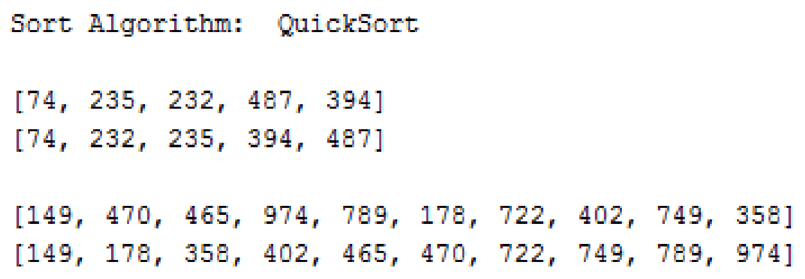
Sorted arrays of different size



```
Sort Algorithm:  InsertionSort

[74, 235, 232, 487, 394]
[74, 232, 235, 394, 487]

[149, 470, 465, 974, 789, 178, 722, 402, 749, 358]
[149, 178, 358, 402, 465, 470, 722, 749, 789, 974]
```

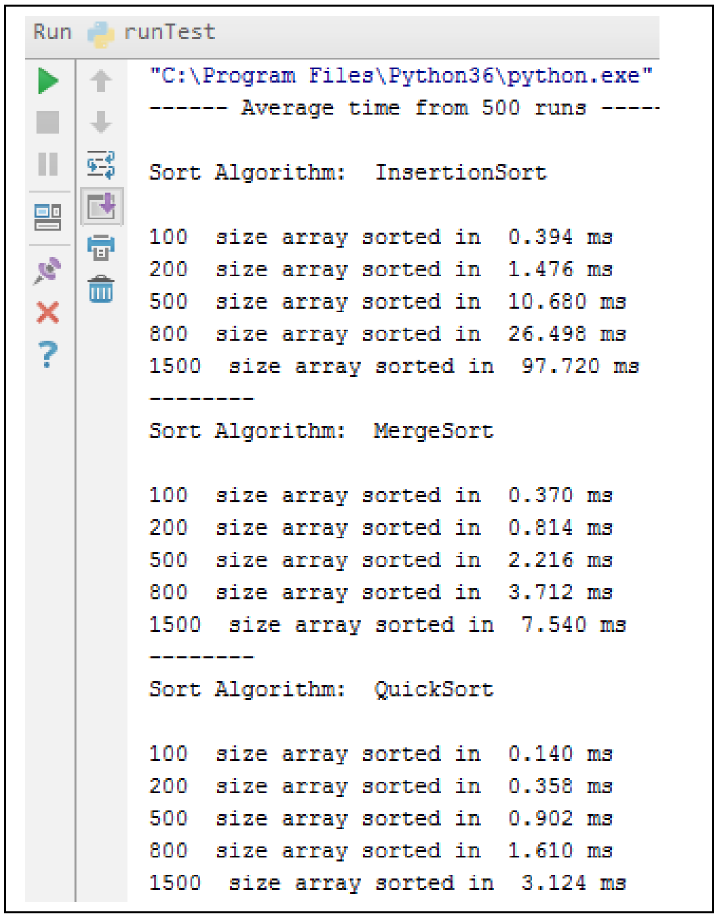


```
Sort Algorithm:  MergeSort

[74, 235, 232, 487, 394]
[74, 232, 235, 394, 487]

[149, 470, 465, 974, 789, 178, 722, 402, 749, 358]
[149, 178, 358, 402, 465, 470, 722, 749, 789, 974]
```



```
Sort Algorithm:  QuickSort

[74, 235, 232, 487, 394]
[74, 232, 235, 394, 487]

[149, 470, 465, 974, 789, 178, 722, 402, 749, 358]
[149, 178, 358, 402, 465, 470, 722, 749, 789, 974]
```

```
To get this results run runTest.py
```



```
"C:\Program Files\Python36\python.exe"
------ Average time from 500 runs -----

Sort Algorithm:  InsertionSort

100  size array sorted in  0.394 ms
200  size array sorted in  1.476 ms
500  size array sorted in  10.680 ms
800  size array sorted in  26.498 ms
1500  size array sorted in  97.720 ms
--------
Sort Algorithm:  MergeSort

100  size array sorted in  0.370 ms
200  size array sorted in  0.814 ms
500  size array sorted in  2.216 ms
800  size array sorted in  3.712 ms
1500  size array sorted in  7.540 ms
--------
Sort Algorithm:  QuickSort

100  size array sorted in  0.140 ms
200  size array sorted in  0.358 ms
500  size array sorted in  0.902 ms
800  size array sorted in  1.610 ms
1500  size array sorted in  3.124 ms
```

## runTest.py

```
import random
import sys
sys.setrecursionlimit(100000)
import time
import sort
#Author: Praminda Mahesh Imaduwa-Gamage, UMSL
#CMP3130: Algorithm Analysis and Design, Programming Assignment - 2 11/05/2017


def getMeanExecutionTime(sortAlgo, arraySize):

    size = arraySize
    ListRange = size * 100
    num_runs = 500
    timeList = list()

    for i in range(0, num_runs):

        random.seed(100)
        arr = random.sample(range(ListRange), size)
        start_time = time.time()

        if sortAlgo == 'QuickSort':
            sort.withQuick(arr)
        if sortAlgo == 'MergeSort':
            sort.withMerge(arr)
        if sortAlgo == 'InsertionSort':
            sort.withInsertion(arr)

        timeList.append(1000 * (time.time() - start_time))

    return sum(timeList) / len(timeList)


print("------ Average time from 500 runs -------", "\n")
sortAlgorithm = ['InsertionSort', 'MergeSort', 'QuickSort']

for sortType in sortAlgorithm:
    print("Sort Algorithm: ", sortType, '\n')
    for size in [100, 200, 500, 800, 1500]:
        print(size, " size array sorted in ", "%0.3f ms " %
getMeanExecutionTime(sortType, arraySize = size))
    print("--------")
```

```
## mergeSort.py

def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * (n1)
    R = [0] * (n2)

    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    i = 0
    j = 0
    k = l

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

def mergeSort(arr, l, r):
    if l < r:
        m = (l + (r - 1)) // 2
        mergeSort(arr, l, m)
        mergeSort(arr, m + 1, r)
        merge(arr, l, m, r)
    return arr
```

```
## insertionSort.py

def insertionSort(arr):

    for i in range(1, len(arr)):

        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

    return arr
```

```
## quicksort.py

def partition(arr, low, high):
    i = (low - 1)

    pivot = arr[high]

    for j in range(low, high):
        if arr[j] <= pivot:
            i = i + 1
            arr[i], arr[j] = arr[j],
arr[i]

    arr[i + 1], arr[high] = arr[high],
arr[i + 1]
    return (i + 1)


def quickSort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quickSort(arr, low, pi - 1)
        quickSort(arr, pi + 1, high)
    return arr
```

2.Write a program to design Huffman codes for a given sequence.

Try to write generalized program, where characters and their frequencies can be supplied by command line.

Test your program on following Example. Show saved bits

Information to be transmitted over the internet contains the following characters with their associated frequencies as shown in the following table:

Characters	a	e	l	n	o	s	t
Frequency	45	65	13	45	18	22	53

Results:

To get this results run huffman.py in command line

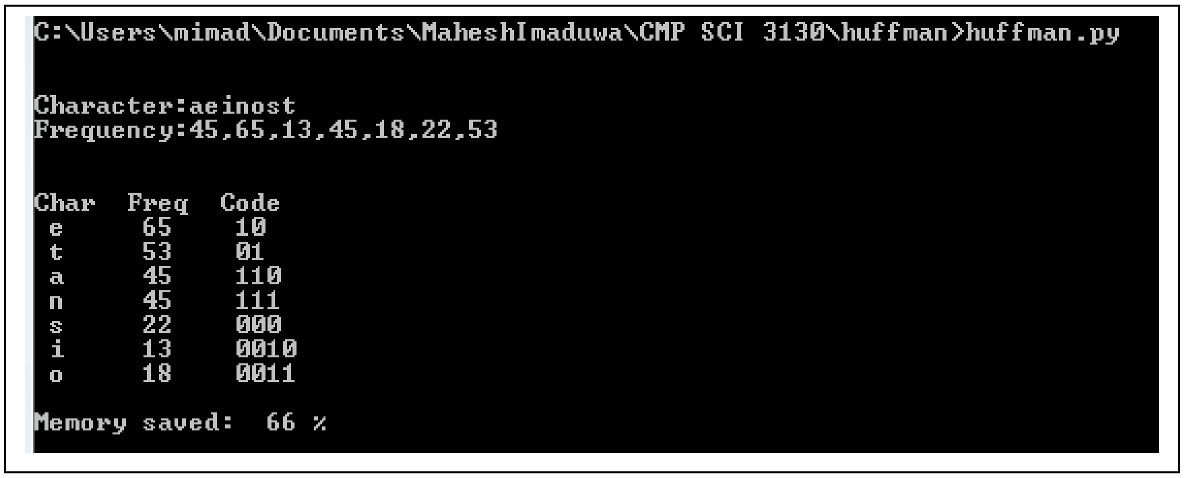
Enter characters and Frequency as follows.

cd huffman

characters: aeinost // as strings

Frequency: 45,65,13,45,18,22,53 //separated by comma with no spaces

Press Enter



```
C:\Users\mimad\Documents\MaheshImaduwa\CMP SCI 3130\huffman>huffman.py


Character:aeinost
Frequency:45,65,13,45,18,22,53


Char   Freq   Code
 e      65     10
 t      53     01
 a      45     110
 n      45     111
 s      22     000
 i      13     0010
 o      18     0011

Memory saved:   66 %
```

## huffman.py

```
from collections import defaultdict
from heapq import *
#Author: Praminda Mahesh Imaduwa-Gamage, UMSL
#CMP3130: Algorithm Analysis and Design, Programming Assignment - 2 11/05/2017
def encode(char2freq):

    heap = [[freq, [char, ""]] for char, freq in char2freq.items()]
    heapify(heap)
    while len(heap) > 1:
        low = heappop(heap)
        high = heappop(heap)
        for pair in low[1:]:
            pair[1] = '0' + pair[1]
        for pair in high[1:]:
            pair[1] = '1' + pair[1]
        heappush(heap, [low[0] + high[0]] + low[1:] + high[1:])
    return sorted(heappop(heap)[1:], key = lambda pos: (len(pos[-1]), pos))

def charIn(charList, freqList):

    text = ""
    for freq in range(len(freqList)):
        text += charList[freq] * freqList[freq]

    char2freq = defaultdict(int)
    for ch in text:
        char2freq[ch] += 1

    huff = encode(char2freq)
    print("Char  Freq  Code")
    numBits = 0
    originalBits = sum(freqList) * 8
    for pos in huff:
        print(" %s      %s     %s" % (pos[0], char2freq[pos[0]], pos[1]))
        numBits += char2freq[pos[0]]*len(pos[1])
    print("")
    print("Memory saved: ", 100 * (originalBits - numBits)//originalBits, "%")

print('\n')
inputStr = input('Character:')
inputNum = input('Frequency:')
characters = list()
for char in inputStr:
    characters.append(char)

#charIn(['a', 'e', 'i', 'n', 'o', 's', 't'], [45, 65, 13, 45, 18, 22, 53])

numbers = inputNum.split(",")
numbers = [int(num) for num in numbers]
print('\n')
charIn(characters, numbers)
```