Programming Assignment 1

INFO-5502 (Section 002):
Analytic Tools, Techniques and Methods

Ramandeep Harjai

February 7, 2021

Write a Python program that can take strings of different lengths - each string may include digits, characters, and special symbols - and then sort them - you can use any sorting algorithm that interests you. You need to define the rule for sorting and then implement the sorting function using Python - DO NOT use any existing sort function from either Python or an external module - that is, program the "sort" algorithm yourself. Using Python, visualize a list of input strings - the list must include at least 500 strings of differing lengths. Use a scatter plot to do the visualization - one dimension will be the length of each string - and the other dimension could be the order of the string in the list.

Your submission should include the following:

1) Submit the python program with a brief description of how the program works (3 points); 2) A description that specifically describes how the sorting rule works (3 points); 3) Visualization results and testing results with 5 test cases - the 5 test cases would consist of 5 lists of 500 strings of differing lengths (4 points).

These strings can come from any source that is of interest to you - e.g. favorite book, Wikipedia, Twitter data - you can use strings from any source that interests you - you just need string data from somewhere.

List of Random Phrases The python function: `randomStringGenerator`, generates a list of random phrases. Each phrase is generated using 1 to 5 words (randomly), where each word is made up of 1 to 12 alpha-numeric characters (randomly). The function accepts an argument: `numStrings` — total number of strings (or phrases) to be generated, and returns a `List[]` of size: `numStrings`.

```
def randomStringGenerator(numStrings:int)->[]:
  """ generates & return a list of random strings
  """
  lstStr = []         # list to hold random strings
  minPhraseLen = 1    # minimum number of words in Random Phrase
  maxPhraseLen = 5    # maximum number of words in Random Phrase
  minStrLen = 1       # minimum length for the random string
  maxStrLen = 12      # maximum length for the random string
  for _ in range(numStrings):
    rndPhraseLen = randint(minPhraseLen, maxPhraseLen)
    rndPhrase = ""
    for i in range(rndPhraseLen):
      rndLen = randint(minStrLen, maxStrLen)
      if _%5 == 0:
        # use alphabets and numbers
        rndPhrase += ''.join(choices(string.ascii_uppercase + string.digits,
                                     k = randint(minStrLen, maxStrLen)))
      elif _%9 == 0:
        # use alphabets, numbers, and special characters
        rndPhrase += ''.join(choices(string.ascii_uppercase + string.digits +
                                     string.punctuation,
                                     k = randint(minStrLen, maxStrLen)))
      else:
        # use alphabets only
        rndPhrase += ''.join(choices(string.ascii_uppercase,
                                     k = randint(minStrLen, maxStrLen)))

      rndPhrase += " "
    lstStr.append(rndPhrase)
  return lstStr
```

Output — List of Random Phrases:

```
['4FXMRH ', 'NHAYCZ V UVJOUWEU I INOOK ', 'C NRNAEGAUV VFFG HDJ DQMIDPY ',
'DGLYZ EKH RMOGMF JVUFP ', 'WMXLXNZSKJGP GD ', 'LJMEXHDFHTFJ 7ABCB FZ7Q39FT2 12X3JR ',
'KHMKA AL DG U ZFVO ', 'CALCFHCZX VRWZLTOPXX ', 'MDMYBR SSDER ', '=\$=? ', 'Z2 FB ',
'E ', 'ULGAIJXON JWRCDC ZBDHPTMZD ', 'GEVSGK TFRPRVUYUYM ', 'HBWHTSU UVAMVENCYWSQ ',
'5G A81 ', 'KLUQOZUANFI SPB AAFWZO GAUIBZIE EMXYSJVNA ', ...]
```

ASCII Score The python function: `stringAsciiScore`, returns the ASCII score of a string, which is a total of ASCII values of all characters in the string. The function accepts a parameter: `strg` — string for which the ASCII score needs to be computed, and returns `int` — numeric value of ASCII score.

```
def stringAsciiScore(strg:str)->int:
  """ compute and return the ascii score for a given string
      ascii score of a string = total of ascii scores
                                of all characters in the string
  """
  asciiScore = 0
  if not strg:
    return asciiScore
  for _ in range(len(strg)):
    asciiScore += ord(strg[_])
  return asciiScore
```

Output — ASCII Score:

```
>> 'A' (input)
65 (output)
```

Output — ASCII Score:

```
>> '$' (input)
36 (output)
```

Output — ASCII Score:

```
>> 'Hello World!' (input)
1085 (output)
```

H	e	l	l	o		W	o	r	l	d	!
72	101	108	108	111	32	87	111	114	108	100	33
1085											

Figure 1: Compute ASCII Score for 'Hello World!'

Sorting The python function: `bubbleSort`, returns a sorted string where all characters in the string are sorted in ascending order of their ASCII values. The function uses `bubble sort algorithm` to the sort the list. The function accepts a parameter: `strg` — string to be sorted, and returns `str` — sorted string.

```
def bubbleSort(strg:str)->str:
  """ sort list using bubble sort algorithm """
  if (not strg):
    return ""

  strgSortedList = list(strg)
  strgLen = len(strgSortedList)
  for i in range(strgLen):
    for j in range (strgLen-i-1):
      # compare the ascii values of two characters
      # character with smaller ascii value will precede the other character
      if stringAsciiScore(strgSortedList[j]) > stringAsciiScore(strgSortedList[j+1]):
        strgSortedList[j], strgSortedList[j+1] = strgSortedList[j+1], strgSortedList[j]

  return "".join(strgSortedList)
```

Output — Sorting:

```
>> 'LJMEXHDFHTFJ 7ABCB FZ7Q39FT2 12X3JR' (input: phrase)
'   12233779ABBCDEFFFFHHJJJLMQRTTXXZ' (output: sorted string)
```

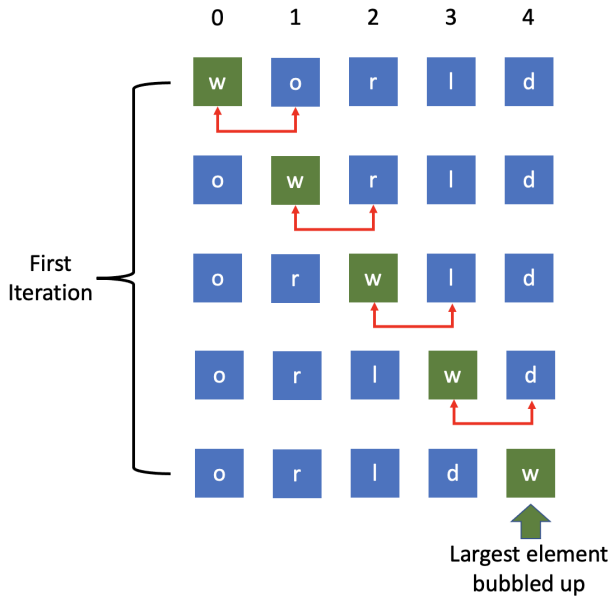

Figure 2: Bubble Sort Algorithm — First Iteration

Sorting (List) The python function: `bubbleSortList`, returns a sorted list where all elements in the list are sorted in ascending order of their Length and ASCII values. The function uses `bubble sort algorithm` to the sort the list. The function accepts a parameter: `lst` — list of strings (or phrases), and returns `List[]` — sorted list.

```
def bubbleSortList(lst:list)->list:
  """ sort list using bubble sort algorithm """
  if (not lst) or (len(lst)<=0):
    return []

  sortedList = lst.copy()
  lstLen = len(sortedList)
  for i in range(lstLen):
    for j in range (lstLen-i-1):
      # compare by string length
      # shorter length string should preceed longer length string
      if len(sortedList[j]) > len(sortedList[j+1]):
        sortedList[j], sortedList[j+1] = sortedList[j+1], sortedList[j]

      # for same length strings
      # compare the total ascii values of two string
      # string with smaller total ascii value will preceed the other string
      if len(sortedList[j]) == len(sortedList[j+1]):
        if stringAsciiScore(sortedList[j]) > stringAsciiScore(sortedList[j+1]):
          sortedList[j], sortedList[j+1] = sortedList[j+1], sortedList[j]

  return sortedList
```

Output — Sorting (List):

```
>> ['P 1K3SV', 'VEAPH HMBOGGWBW ZXZJC OJTWXCEDYRGM BVE',
'ZPQKFEDD DIAUAZFNW J CAAEPI LVAIISGVHBR', 'FDUEZUNB', 'KFEA',
'QVCDIA5OL', 'S LBGX', 'GY XCYFG VKWPGIICBYD', 'NT VPQT XCPLQH DA',
'*W4P +$C>4P[9'] (input: list)

['KFEA', 'S LBGX', 'P 1K3SV', 'FDUEZUNB', 'QVCDIA5OL',
'*W4P +$C>4P[9', 'NT VPQT XCPLQH DA', 'GY XCYFG VKWPGIICBYD',
'VEAPH HMBOGGWBW ZXZJC OJTWXCEDYRGM BVE',
'ZPQKFEDD DIAUAZFNW J CAAEPI LVAIISGVHBR'] (output: sorted list)
```

Program Execution The python program starts with creating a list of 500 random phrases, using `randomStringGenerator()`. Program then iterates through each element of the list. Each element is sorted with the `bubble sort algorithm`, using the `bubbleSort()` function. Next, the program sorts the list itself, with the `bubble sort algorithm`, using the `bubbleSortList()` function. Finally, the program creates two scatter charts — (1) for list with sorted elements, and (2) for sorted list of sorted elements. Scatter charts are created using `matplotlib` module.

```
# generate list of random strings
maxStrings = 500
lstRndStrg = randomStringGenerator(maxStrings)

print(len(lstRndStrg), "Random Strings: ", lstRndStrg)
```

```
# sort each element of the list
lstRndStrgSorted = []
for _ in range(len(lstRndStrg)):
  lstRndStrgSorted.append(bubbleSort(lstRndStrg[_]))

print(len(lstRndStrgSorted), "Random Strings (Sorted Elements): ", lstRndStrgSorted)
```

```
# sort entire list
lstRndStrgSorted2 = bubbleSortList(lstRndStrgSorted)
print(len(lstRndStrgSorted2), "Random Strings (Sorted List & Elements): ",
        lstRndStrgSorted2)
```

```
# Plot
fig, (ax1, ax2) = plt.subplots(1,2,constrained_layout=True)
fig.set_size_inches(12, 6)
fig.set_dpi(100)
fig.suptitle('Scatter Plots of Unsorted & Sorted List of Random Strings')

# Scatter (sub)Plot for Unsorted List
ax1.set_title('List with Sorted Elements')
ax1.set(xlabel='Position of Random String in the List',
        ylabel='Length of Random String')
ax1.scatter([range(len(lstRndStrgSorted))],
            [len(strg) for strg in lstRndStrgSorted])

# Scatter (sub)Plot for Sorted List
ax2.set_title('Sorted List with Sorted Elements')
ax2.set(xlabel='Position of Random String in the List',
        ylabel='Length of Random String')
ax2.scatter([range(len(lstRndStrgSorted2))],
            [len(strg) for strg in lstRndStrgSorted2])
```

Output: 5 test cases

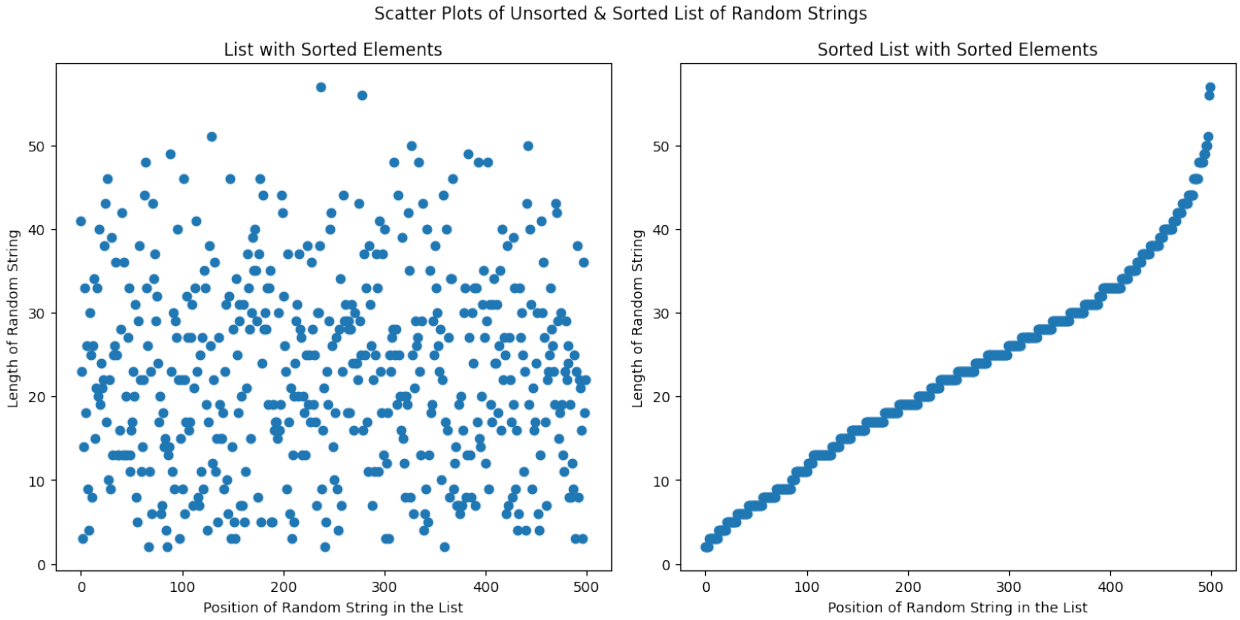

Figure 3: Output 1

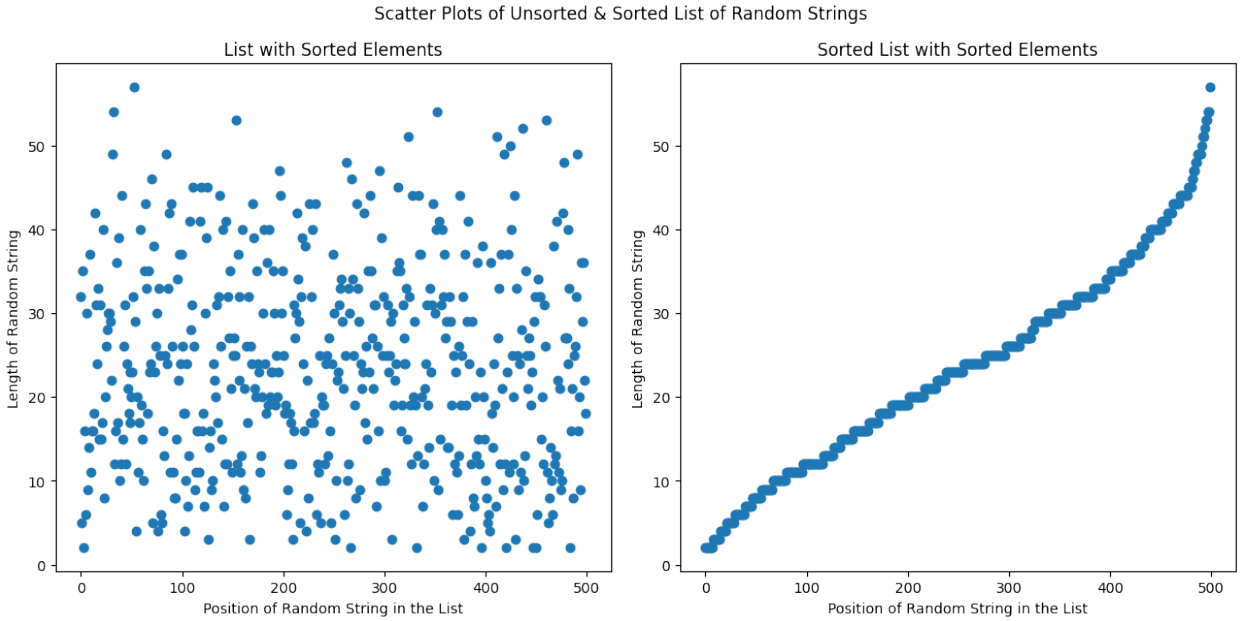

Figure 4: Output 2

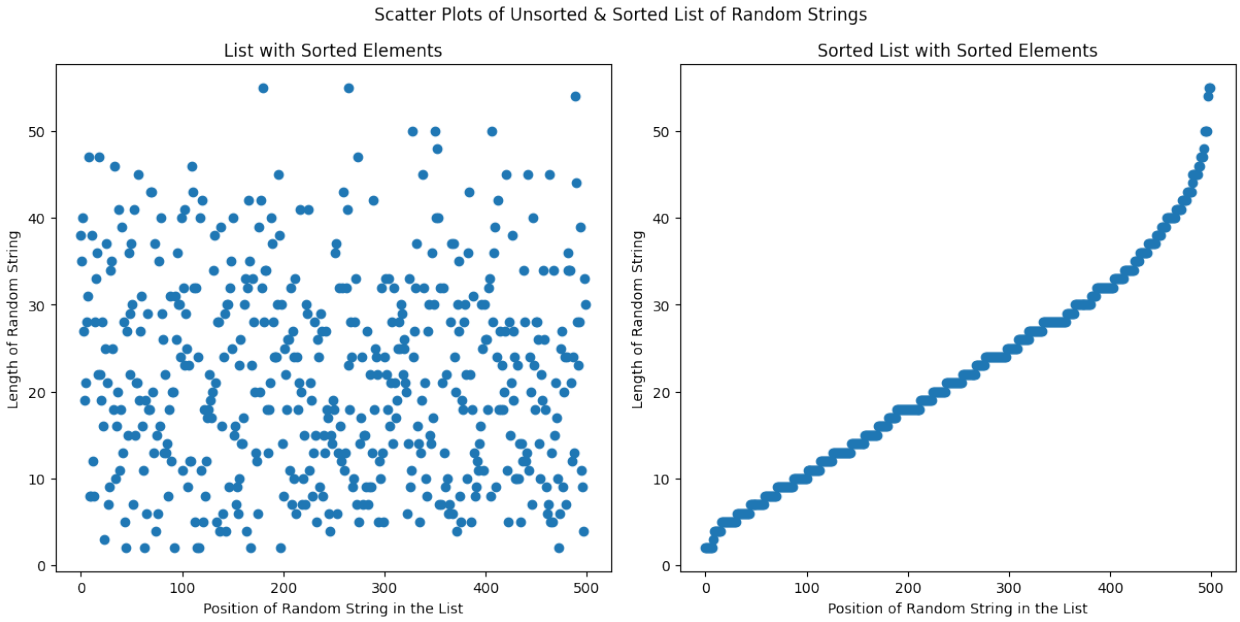

Figure 5: Output 3

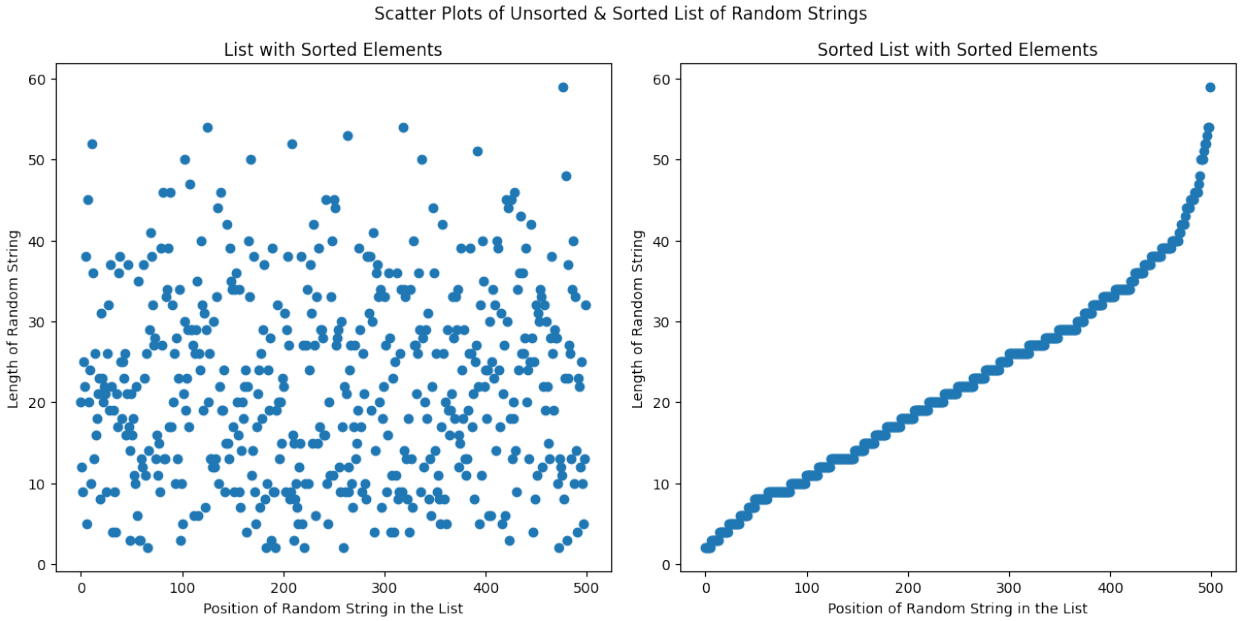

Figure 6: Output 4

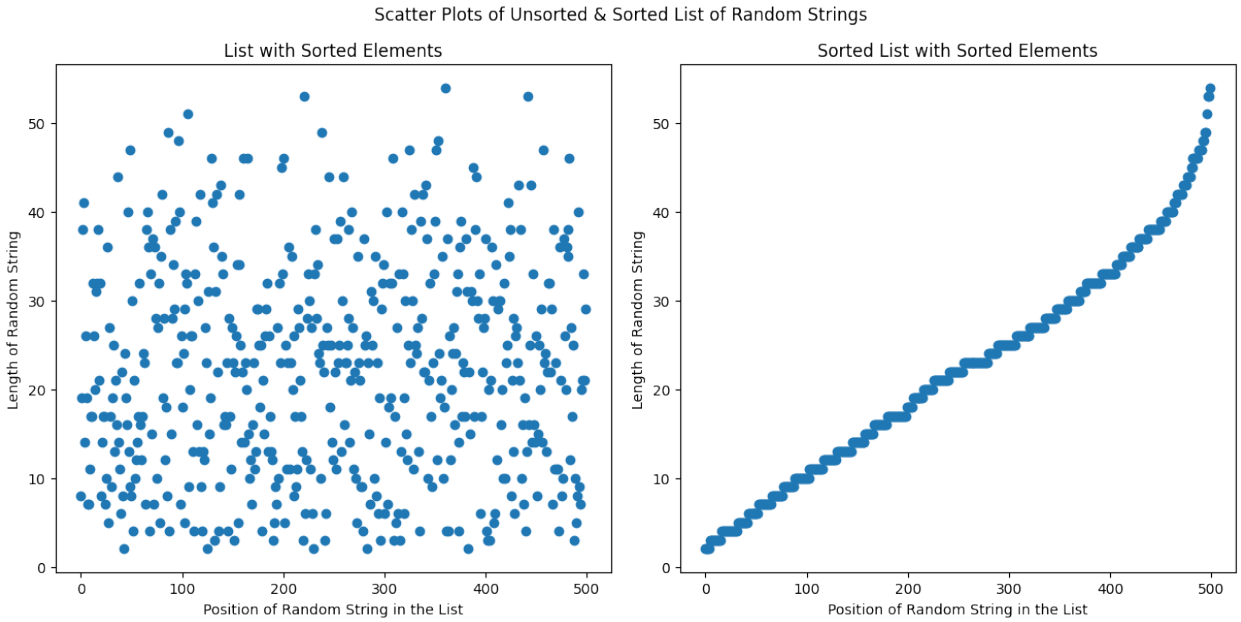

Figure 7: Output 5