Assignment 1 of CP5602

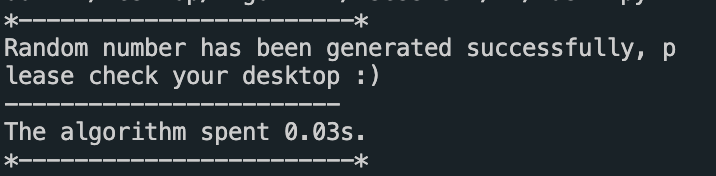
***Task1:***

*a). (All the text files generated can be found in the attachment. TestFile/TaskA0.txt--TaskA6.txt, TaskB0.txt--TaskB6.txt)*

**TestA0:** Generate n = 5000 non-unique random values from -35500 to 36600. (The comparison testing of TestA1)

**图片包含 瓶子

描述已自动生成**

****

**TestA1:** Generate n = 50000 non-unique random values from -35500 to 36600.

**图片包含 瓶子

描述已自动生成**

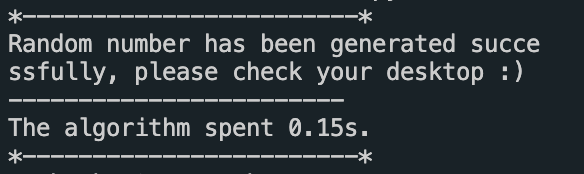
**图片包含 文字

描述已自动生成**

**TestA2:** Generate n = 55000 non-unique random values from -25500 to 46600.

**图片包含 瓶子

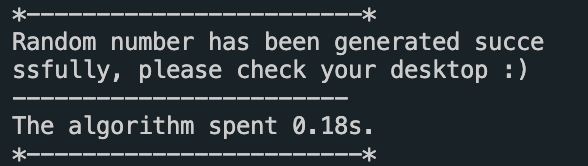
描述已自动生成**

****

**TestA3:** Generate n = 60000 non-unique random values from -15500 to 56600.

**图片包含 瓶子

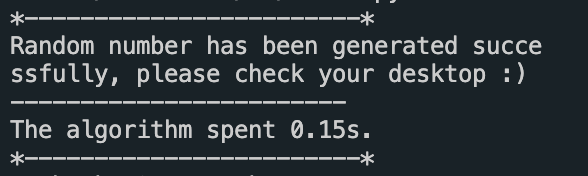
描述已自动生成**

****

**TestA4:** Generate n = 65000 non-unique random values from -13800 to 96800.

**图片包含 瓶子

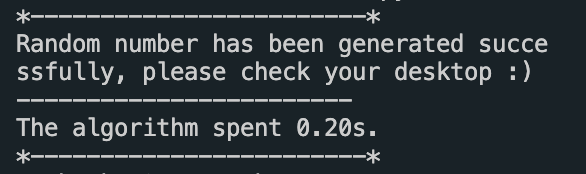
描述已自动生成**

****

**TestA5:** Generate n = 70000 non-unique random values from -13800 to 96800.

**图片包含 瓶子

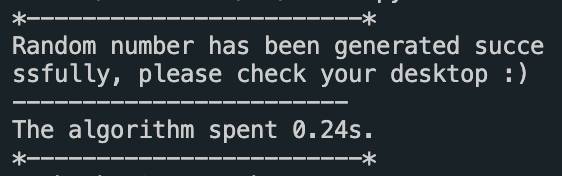
描述已自动生成**

****

**TestA6:** Generate n = 80000 non-unique random values from -13800 to 96800.

图片包含 瓶子

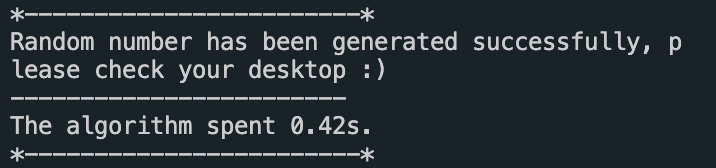
描述已自动生成



**TestB0:** Generate n = 5000 unique random values from -35500 to 36600. (The comparison testing of TestB1)

**图片包含 瓶子

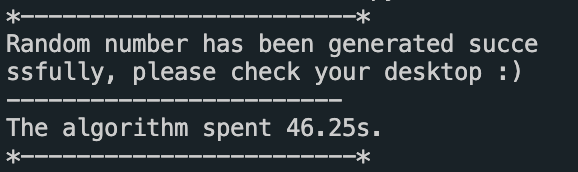
描述已自动生成**

****

**TestB1:** Generate n = 50000 unique random values from -35500 to 36600.

**图片包含 瓶子

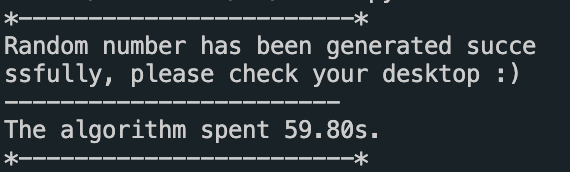
描述已自动生成**

****

**TestB2:** Generate n = 55000 unique random values from -25500 to 46600.

**图片包含 瓶子

描述已自动生成**

****

**TestB3:** Generate n = 60000 unique random values from -15500 to 56600.

**图片包含 瓶子

描述已自动生成**

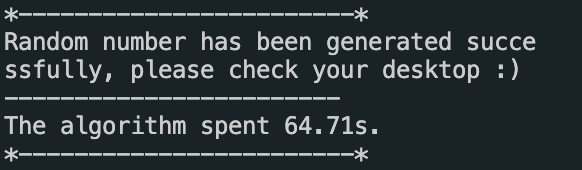
**图片包含 文字

描述已自动生成**

**TestB4:** Generate n = 65000 unique random values from -13800 to 96800.

**图片包含 瓶子

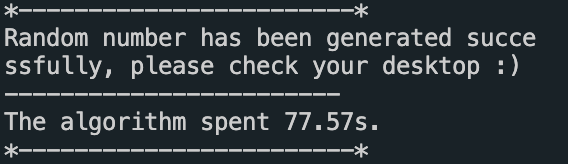
描述已自动生成**

****

**TestB5:** Generate n = 70000 unique random values from -13800 to 96800.

**图片包含 瓶子

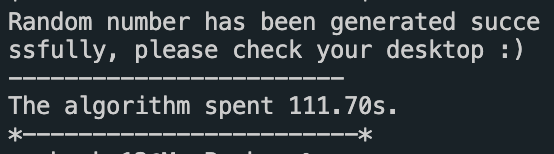
描述已自动生成**

****

**TestB6:** Generate n = 80000 unique random values from -13800 to 96800.

图片包含 瓶子

描述已自动生成



*b).*

**The source code:** (Or check the attachment “ZhangZibo\_13684506\_Assn1\_Task1.py”)

1. # This is the program for the Assignment1 Task 1, it contains rNGeneration function and saveToFile function. This program is used for generating non-unique ranmdom numbers or unique random numbers and save the result into a file.
3. **import** random # Prepare for the random generator
4. **import** re # Prepare for the validation
5. **import** time # Prepare for the time spend calculation
7. lowerLimit = -35500 # The variable of lower limit, users can chenge the value here
8. upperLimit = 36600 # The variable of upper limit, users can chenge the value here
9. numOfValues = 5000 # The variable of number of random value, users can chenge the value here
10. noDuplicates = "f" # The variable of the unique switch's status, users can change the value here ("T","t","F" or "f")
12. **def** saveToFile(\_resStr, \_fileName): # The function to save file on your desktop and takes 2 parameters whcich are the content to save and the file name
13. file = open(\_fileName, mode = "w", encoding = "utf-8") # Mode is "w", which means old files will be overwritten
14. file.write(\_resStr)
15. **print**("\*------------------------\*\nRandom number has been generated successfully, please check your desktop :)\n------------------------")
16. file.close
18. **def** rNGenerator(lowerLimit, upperLimit, numOfValues, noDuplicates): # The function to generate a series unique or non-unique random numbers with a certain range
19. limit = re.compile(r'^[-+]?[0-9]+$') # For the validation
20. **if** limit.match(str(lowerLimit)) == None **or** limit.match(str(upperLimit)) == None **or** str(numOfValues).isdigit() == False **or** **not**(noDuplicates == "T" **or** noDuplicates == "t" **or** noDuplicates == "F" **or** noDuplicates == "f"): # Check the limit variables must be integers and numOfValues must be positive integer and noDuplicates must be "T", "t", "F or "f"
21. **if** limit.match(str(lowerLimit)) == None:
22. **print**("Invalid input of '"+str(lowerLimit)+"', the parameter lowerLimit must be an integer.\n")
23. **else**:
24. **pass**
25. **if** limit.match(str(upperLimit)) == None:
26. **print**("Invalid input of '"+str(upperLimit)+"', the parameter upperLimit must be an integer.\n")
27. **else**:
28. **pass**
29. **if** str(numOfValues).isdigit() == False:
30. **print**("Invalid input of '"+str(numOfValues)+"', the parameter numOfValues must be an positive integer.\n")
31. **else**:
32. **pass**
33. **if** **not**(noDuplicates == "T" **or** noDuplicates == "t" **or** noDuplicates == "F" **or** noDuplicates == "f"):
34. **print**("Invalid input of '"+str(noDuplicates)+"', the parameter noDuplicates must be 'T', 't', 'F' or 'f'.\n")
35. **else**:
36. **pass**
37. **return**
38. **elif** lowerLimit >= upperLimit: # Check the lower limit must be less or equal to upper limit
39. **print**("Invalid input, the parameter lowerLimit cannot be gearter than parameter upperLimit.\n")
40. **return**
41. **elif** numOfValues > (upperLimit - lowerLimit + 1) **and** (noDuplicates == "T" **or** noDuplicates == "t"): # Check the numOfValues cannot be bigger than the range of limit when the user want unique numbers
42. **print**("Invalid input, the parameter numOfValues cannot be bigger than the range of limit when you want unique random numbers.")
43. **return**
44. **elif** noDuplicates == "F" **or** noDuplicates == "f": # Generate non-unique numbers
45. resStr = ""
46. **for** i **in** range(numOfValues):
47. res = random.randint(lowerLimit, upperLimit)
48. resStr += str(res) + "\n"
49. saveToFile("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates: "+noDuplicates+"\n\n"+resStr,"Desktop/RandomNum\_non-unique.txt")
50. **print**("The algorithm spent {:.2f}s.\n\*------------------------\*".format(time.time()-startTime)) # Print the time spending
51. **else**: # Generate unique numbers
52. resStr = ""
53. resLis = []
54. **while** len(resLis) < numOfValues:
55. res = random.randint(lowerLimit, upperLimit)
56. **if** res **in** resLis:
57. **pass**
58. **else**:
59. resStr += str(res) + "\n" # Use for printing in the file
60. resLis.append(res) # Use for duplicates check
61. saveToFile("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates: "+noDuplicates+"\n\n"+resStr,"Desktop/RandomNum\_unique.txt")
62. **print**("The algorithm spent {:.2f}s.\n\*------------------------\*".format(time.time()-startTime)) # Print the time spending
64. startTime = time.time() # Mark the start time at the beginning
65. rNGenerator(lowerLimit,upperLimit,numOfValues,noDuplicates) # Call the main function

*c).*

**Graph#1:** (Or check the attachment “ZhangZibo\_13684506\_Assn1.xlsx”)

图片包含 屏幕截图

描述已自动生成

**Graph#2:** (Or check the attachment “ZhangZibo\_13684506\_Assn1.xlsx”)

图片包含 屏幕截图

描述已自动生成

*d).*

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Timing(s) | | |
| Worst | Average | Best |
| Unique random number | O(n) | O(n) | O(n) |
| Non-unique random number | O(n^2) | O(n\*log n) | O(n) |

***Task2:***

*a). (All the text files generated can be found in the attachment. TestFile/TaskC0—TaskC6, TaskD0—TaskD6)*

**TestC0:** Generate n = 5000 unique random values from -35500 to 36600 and sort them by insertion sort algorithm. (The comparison testing of TestC1)

**图片包含 瓶子

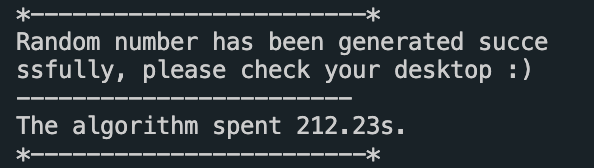
描述已自动生成**

****

**TestC1:** Generate n = 50000 unique random values from -35500 to 36600 and sort them by insertion sort algorithm.

**图片包含 瓶子

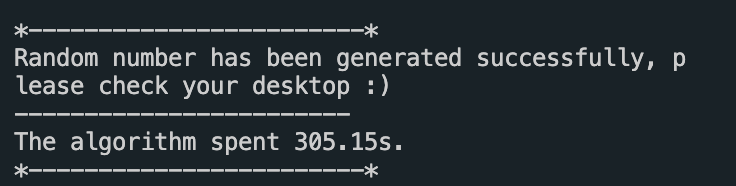
描述已自动生成**

****

**TestC2:** Generate n = 55000 unique random values from -25500 to 46600 and sort them by insertion sort algorithm.

**图片包含 瓶子

描述已自动生成**

****

**TestC3:** Generate n = 60000 unique random values from -15500 to 56600 and sort them by insertion sort algorithm.

**图片包含 瓶子

描述已自动生成**

**图片包含 文字

描述已自动生成**

**TestC4:** Generate n = 65000 unique random values from -13800 to 96800 and sort them by insertion sort algorithm.

**图片包含 瓶子

描述已自动生成**

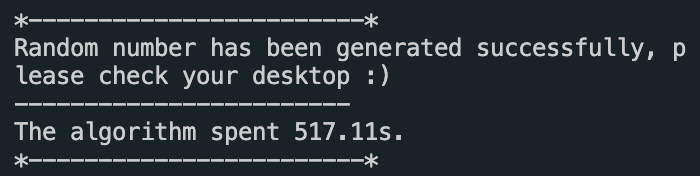
**图片包含 文字

描述已自动生成**

**TestC5:** Generate n = 70000 unique random values from -13800 to 96800 and sort them by insertion sort algorithm.

**图片包含 瓶子

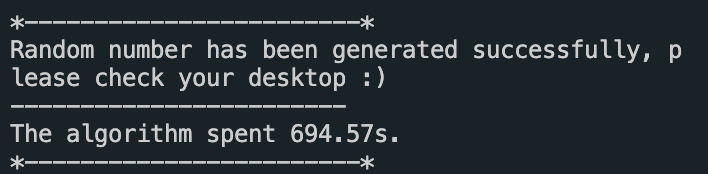
描述已自动生成**

****

**TestC6:** Generate n = 80000 unique random values from -13800 to 96800 and sort them by insertion sort algorithm.

图片包含 瓶子

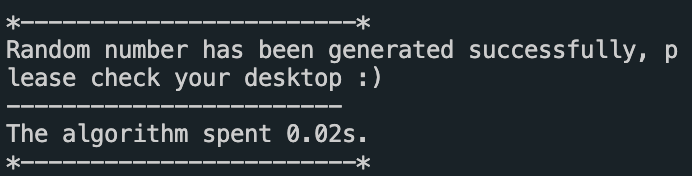
描述已自动生成



**TestD0:** Generate n = 5000 unique random values from -35500 to 36600 and sort them by quick sort algorithm. (The comparison testing of TestD1)

**图片包含 瓶子

描述已自动生成**

****

**TestD1:** Generate n = 50000 unique random values from -35500 to 36600 and sort them by quick sort algorithm.

**图片包含 瓶子

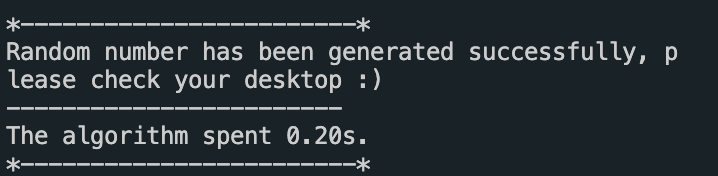
描述已自动生成**

****

**TestD2:** Generate n = 55000 unique random values from -25500 to 46600 and sort them by quick sort algorithm.

**图片包含 瓶子

描述已自动生成**

****

**TestD3:** Generate n = 60000 unique random values from -15500 to 56600 and sort them by quick sort algorithm.

**图片包含 瓶子

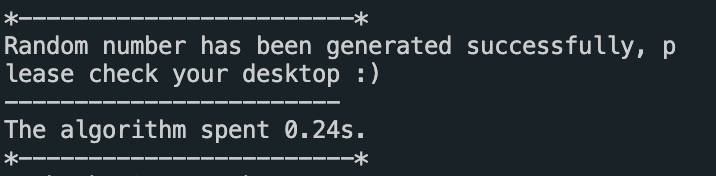
描述已自动生成**

****

**TestD4:** Generate n = 65000 unique random values from -13800 to 96800 and sort them by quick sort algorithm.

**图片包含 瓶子

描述已自动生成**

****

**TestD5:** Generate n = 70000 unique random values from -13800 to 96800 and sort them by quick sort algorithm.

**图片包含 瓶子

描述已自动生成**

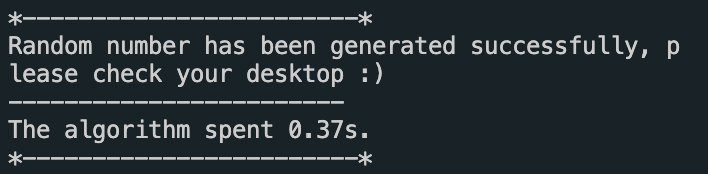
**图片包含 屏幕截图

描述已自动生成**

**TestD6:** Generate n = 80000 unique random values from -13800 to 96800 and sort them by quick sort algorithm.

图片包含 瓶子

描述已自动生成



*b).*

**The source code of insertion sorting:** (Or check the attachment “ZhangZibo\_13684506\_Assn1\_Task2-1.py”)

1. # This is the program for the Assignment1 Task 2, it contains rNGeneration function, saveToFile function and insertionSort functon. This program is used for sorting a series numbers that are generated by rNGenerator by using insertion sorting algorithm and save the result into a file.
3. **import** random # Prepare for the random generator
4. **import** re # Prepare for the validation
5. **import** time # Prepare for the time spend calculation
7. lowerLimit = -35500 # The variable of lower limit, users can chenge the value here
8. upperLimit = 36600 # The variable of upper limit, users can chenge the value here
9. numOfValues = 5000 # The variable of number of random value, users can chenge the value here
10. noDuplicates = "t" # The variable of the unique switch's status, users can change the value here ("T","t","F" or "f")
12. **def** saveToFile(\_resStr, \_fileName): # The function to save file on your desktop and takes 2 parameters whcich are the content to save and the file name
13. file = open(\_fileName, mode = "w", encoding = "utf-8")
14. file.write(\_resStr) # Mode is "w", which means old files will be overwritten
15. **print**("\*------------------------\*\nRandom number has been generated successfully, please check your desktop :)\n------------------------")
16. file.close
18. **def** rNGenerator(lowerLimit, upperLimit, numOfValues, noDuplicates): # The function to generate a series unique or non-unique random numbers with a certain range
19. limit = re.compile(r'^[-+]?[0-9]+$') # For the validation
20. **if** limit.match(str(lowerLimit)) == None **or** limit.match(str(upperLimit)) == None **or** str(numOfValues).isdigit() == False **or** **not**(noDuplicates == "T" **or** noDuplicates == "t" **or** noDuplicates == "F" **or** noDuplicates == "f"): # Check the limit variables must be integers and numOfValues must be positive integer and noDuplicates must be "T", "t", "F or "f"
21. **if** limit.match(str(lowerLimit)) == None:
22. **print**("Invalid input of '"+str(lowerLimit)+"', the parameter lowerLimit must be an integer.\n")
23. **else**:
24. **pass**
25. **if** limit.match(str(upperLimit)) == None:
26. **print**("Invalid input of '"+str(upperLimit)+"', the parameter upperLimit must be an integer.\n")
27. **else**:
28. **pass**
29. **if** str(numOfValues).isdigit() == False:
30. **print**("Invalid input of '"+str(numOfValues)+"', the parameter numOfValues must be an positive integer.\n")
31. **else**:
32. **pass**
33. **if** **not**(noDuplicates == "T" **or** noDuplicates == "t" **or** noDuplicates == "F" **or** noDuplicates == "f"):
34. **print**("Invalid input of '"+str(noDuplicates)+"', the parameter noDuplicates must be 'T', 't', 'F' or 'f'.\n")
35. **else**:
36. **pass**
37. **return**
38. **elif** lowerLimit >= upperLimit: # Check the lower limit must be less or equal to upper limit
39. **print**("Invalid input, the parameter lowerLimit cannot be gearter than parameter upperLimit.\n")
40. **return**
41. **elif** numOfValues > (upperLimit - lowerLimit + 1) **and** (noDuplicates == "T" **or** noDuplicates == "t"): # Check the numOfValues cannot be bigger than the range of limit when the user want unique numbers
42. **print**("Invalid input, the parameter numOfValues cannot be bigger than the range of limit when you want unique random numbers.")
43. **return**
44. **elif** noDuplicates == "F" **or** noDuplicates == "f": # Generate non-unique numbers
45. resLis = []
46. **for** i **in** range(numOfValues):
47. res = random.randint(lowerLimit, upperLimit)
48. resLis.append(res)
49. **return** resLis
50. **else**: # Generate unique numbers
51. resLis = []
52. **while** len(resLis) < numOfValues:
53. res = random.randint(lowerLimit, upperLimit)
54. **if** res **in** resLis:
55. **pass**
56. **else**:
57. resLis.append(res)
58. **return** resLis
60. **def** insertionSort(\_resLis): # The function to sort the list generated by the rNGnerator by insertion sort algorithm
61. **for** i **in** range(1,len(\_resLis)):
62. current = \_resLis[i]
63. j = i
64. **while** j > 0 **and** \_resLis[j-1] > current:
65. \_resLis[j] = \_resLis[j-1]
66. j -= 1
67. \_resLis[j] = current
68. **return** \_resLis
70. resListUnsort = rNGenerator(lowerLimit,upperLimit,numOfValues,noDuplicates) # Generate a list of random number
71. **if** **not** resListUnsort == None: # For the validation
72. startTime = time.time() # Mark the start time at the beginning
73. resLisSorted = insertionSort(resListUnsort)
74. saveToFile("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates: "+noDuplicates+"\n\n"+str(resLisSorted),"Desktop/RandomNum\_Sorted.txt")
75. **print**("The algorithm spent {:.2f}s.\n\*------------------------\*".format(time.time()-startTime))
76. **else**:
77. **pass**

**The source code of quick sorting:** (Or check the attachment “ZhangZibo\_13684506\_Assn1\_Task2-2.py”)

1. # This is the program for the Assignment1 Task 2, it contains rNGeneration function, saveToFile function and quickSort functon. This program is used for sorting a series numbers that are generated by rNGenerator by using insertion sorting algorithm and save the result into a file.
3. **import** random # Prepare for the random generator
4. **import** re # Prepare for the validation
5. **import** time # Prepare for the time spend calculation
7. lowerLimit = -35500 # The variable of lower limit, users can chenge the value here
8. upperLimit = 36600 # The variable of upper limit, users can chenge the value here
9. numOfValues = 5000 # The variable of number of random value, users can chenge the value here
10. noDuplicates = "t" # The variable of the unique switch's status, users can change the value here ("T","t","F" or "f")
12. **def** saveToFile(\_resStr, \_fileName): # The function to save file on your desktop and takes 2 parameters whcich are the content to save and the file name
13. file = open(\_fileName, mode = "w", encoding = "utf-8")
14. file.write(\_resStr) # Mode is "w", which means old files will be overwritten
15. **print**("\*------------------------\*\nRandom number has been generated successfully, please check your desktop :)\n------------------------")
16. file.close
18. **def** rNGenerator(lowerLimit, upperLimit, numOfValues, noDuplicates): # The function to generate a series unique or non-unique random numbers with a certain range
19. limit = re.compile(r'^[-+]?[0-9]+$')
20. **if** limit.match(str(lowerLimit)) == None **or** limit.match(str(upperLimit)) == None **or** str(numOfValues).isdigit() == False **or** **not**(noDuplicates == "T" **or** noDuplicates == "t" **or** noDuplicates == "F" **or** noDuplicates == "f"): # Check the limit variables must be integers and numOfValues must be positive integer and noDuplicates must be "T", "t", "F or "f"
21. **if** limit.match(str(lowerLimit)) == None:
22. **print**("Invalid input of '"+str(lowerLimit)+"', the parameter lowerLimit must be an integer.\n")
23. **else**:
24. **pass**
25. **if** limit.match(str(upperLimit)) == None:
26. **print**("Invalid input of '"+str(upperLimit)+"', the parameter upperLimit must be an integer.\n")
27. **else**:
28. **pass**
29. **if** str(numOfValues).isdigit() == False:
30. **print**("Invalid input of '"+str(numOfValues)+"', the parameter numOfValues must be an positive integer.\n")
31. **else**:
32. **pass**
33. **if** **not**(noDuplicates == "T" **or** noDuplicates == "t" **or** noDuplicates == "F" **or** noDuplicates == "f"):
34. **print**("Invalid input of '"+str(noDuplicates)+"', the parameter noDuplicates must be 'T', 't', 'F' or 'f'.\n")
35. **else**:
36. **pass**
37. **return**
38. **elif** lowerLimit >= upperLimit: # Check the lower limit must be less or equal to upper limit
39. **print**("Invalid input, the parameter lowerLimit cannot be gearter than parameter upperLimit.\n")
40. **return**
41. **elif** numOfValues > (upperLimit - lowerLimit + 1) **and** (noDuplicates == "T" **or** noDuplicates == "t"): # Check the numOfValues cannot be bigger than the range of limit when the user want unique numbers
42. **print**("Invalid input, the parameter numOfValues cannot be bigger than the range of limit when you want unique random numbers.")
43. **return**
44. **elif** noDuplicates == "F" **or** noDuplicates == "f": # Generate non-unique numbers
45. resLis = []
46. **for** i **in** range(numOfValues):
47. res = random.randint(lowerLimit, upperLimit)
48. resLis.append(res)
49. **return** resLis
50. **else**: # Generate unique numbers
51. resLis = []
52. **while** len(resLis) < numOfValues:
53. res = random.randint(lowerLimit, upperLimit)
54. **if** res **in** resLis:
55. **pass**
56. **else**:
57. resLis.append(res)
58. **return** resLis
60. **def** quickSort(\_resLis): # The function to sort the list generated by the rNGnerator by quick sort algorithm
61. **if** len(\_resLis)<2:
62. **return** \_resLis
63. mid = \_resLis.pop(0)
64. left = []
65. right = []
66. **for** i **in** \_resLis:
67. **if** i > mid:
68. right.append(i)
69. **else**:
70. left.append(i)
71. **return** quickSort(left) + [mid] + quickSort(right)
73. resListUnsort = rNGenerator(lowerLimit,upperLimit,numOfValues,noDuplicates) # Generate a list of random number
74. **if** resListUnsort != None: # For the validation
75. startTime = time.time() # Mark the start time at the beginning
76. resLisSorted = quickSort(resListUnsort)
77. saveToFile("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates: "+noDuplicates+"\n\n"+str(resLisSorted),"Desktop/RandomNum\_Sorted.txt")
78. **print**("The algorithm spent {:.2f}s.\n\*------------------------\*".format(time.time()-startTime))
79. **else**:
80. **pass**

*c).*

**Graph#1:** (Or check the attachment “ZhangZibo\_13684506\_Assn1.xlsx”)



**Graph#2:** (Or check the attachment “ZhangZibo\_13684506\_Assn1.xlsx”)



*d).*

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Timing(s) | | |
| Worst | Average | Best |
| Insertion sorting | O(n\*log n) | O(n\*log n) | O(n\*log n) |
| Quick sorting | O(n\*log n) | O(n\*log n) | O(n) |

***Task3:***

*a).*

The CPU of my machine is Intel Core M Duo Core.

The base frequency is 1.1 GHz and the turbo frequency is 2.4GHz.

When the task 1-2 were running, the frequency of the computer is kept at 2.4GHz all the time.

Then, I run my programs (task1-2) on the laboratory’s computer (CPU: Intel Core i7 7700 @ 3.6GHz 4 Core). Every task only takes half of the time.

Thus, the number of CPU in a machine significantly affect the performance of the unknown algorithms. And as far as I know, whether the code is applied to multi-core and multi-threading has a great impact on the results, too.

*b).*

The RAM of my machine is 8 GB 1600MHz DDR3.

When the task 1-2 were running, the memory usage is very low, which is about 30 to 40 MB.

Thus, a greater RAM capacity doesn’t help improve the performance of unknown algorithms under a small script. I think it will definitely affect the performance if the number of values is very huge or the script is very long.

*c).*

Mine OS is Mac OS.

And I have tested all my programs on another windows computer, and the hardware performance of the two computers is very similar. The result is that almost every program will run faster on Mac than Windows. I think the reason for this result is that there are many processes running in the background of Windows at the same time, which causes the CPU of the computer to run many processes all the time, resulting in the low utilization of CPU resources.

Thus, the kind of OS also play an important role of running the code.