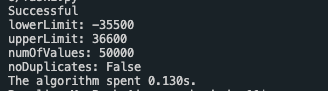
Assignment 1 of CP5602

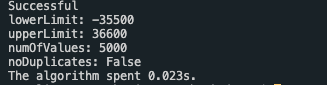
Task1:

a):

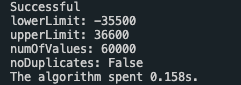
TestA1 Generate n= 50000 random values from -35500 to 36600



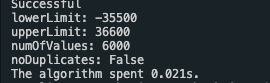
Test a1 (5000) Generate n= 5000 random values from -35500 to 36600( The comparison testing of Test A1):



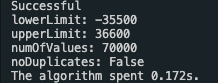
Test A2 Generate n= 60000 random values from -35500 to 36600



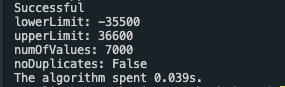
Test a2 Generate n = 6000 random values from -35500 to 36600



Test A3 Generate n = 70000 random values from -35500 to 36600



Test a3 Generate n= 70000 random values from -35500 to 36600



Test An Generate n = 80000 random values from -13800 to 96800

图片包含 文字

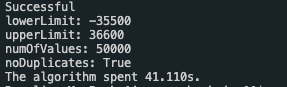
描述已自动生成

Test an Generate n = 8000 random values from -13800 to 96800

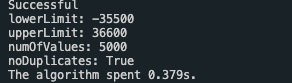
图片包含 文字

描述已自动生成

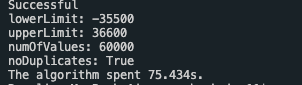
Test B1 Generate n =50000 unique random values from -35500 to 36600.



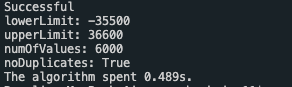
Test b1 Generate n= 5000 unique random values from -35500 to 36600.



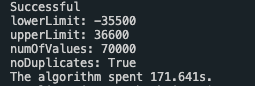
Test B2 Generate n = 60000 unique random values from – 35500 to 36600.



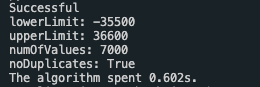
Test b2 Generate n = 6000 unique random values from – 35500 to 36600.



Test B3 Generate n = 70000 unique random values from – 35500 to 36600.



Test b3 Generate n = 7000 unique random values from – 35500 to 36600.



Test Bn Generate n = 80000 random values from -13800 to 96800

图片包含 文字

描述已自动生成

Test bn Generate n = 8000 random values from -13800 to 96800

图片包含 文字

描述已自动生成

b).

Write a "task 1" program to display (for each test), the time taken to:

* generate the required number of random values AND
* store all of them to an external text file (using save-to-file function)
* **import** random
* **import** time
* lowerLimit = -13800
* upperLimit = 96800
* numOfValues = 8000
* noDuplicates = True
* **def** saveToFile(resString, fileName):
* file = open(fileName, mode = "w", encoding = "utf-8")
* file.write(resString)
* **print**("Successful")
* file.close
* **def** rNGenerator(lowerLimit, upperLimit, numOfValues, noDuplicates):
* **if** noDuplicates == False:
* resLis = []
* **for** i **in** range(numOfValues):
* res = random.randint(lowerLimit, upperLimit)
* resLis.append(res)
* saveToFile(str(resLis),"Desktop/RandomNum\_non-unique.txt")
* **print**("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates:",noDuplicates,"\nThe algorithm spent {:.3f}s.".format(time.time()-startTime))
* **else**:
* resLis = []
* **while** len(resLis) < numOfValues:
* res = random.randint(lowerLimit, upperLimit)
* **if** res **in** resLis:
* **pass**
* **else**:
* resLis.append(res)
* saveToFile(str(resLis),"Desktop/RandomNum\_unique.txt")
* **print**("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates:",noDuplicates,"\nThe algorithm spent {:.3f}s.".format(time.time()-startTime))
* startTime = time.time()
* rNGenerator(lowerLimit, upperLimit, numOfValues, noDuplicates)

C).

Graph #1 – the timing results for Test A1 – An and the comparison testing of Test a1-an (non-unique random values)

图片包含 文字

描述已自动生成

图片包含 屏幕截图

描述已自动生成

Graph #2 – the timing results for Test B1 – Bn and the comparison testing of Test b1-bn (unique random values)

图片包含 文字

描述已自动生成

图片包含 屏幕截图, 监视器

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:**

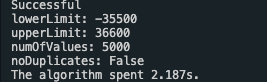
a).

Test C1: Generate n = 50000 random values from -35500 to 36600 and sort them by insertion sort algorithm.

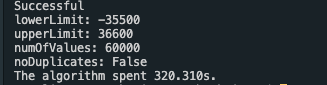
图片包含 文字

描述已自动生成

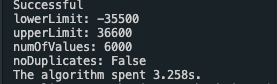
Testc1 Generate n =5000 random values from -35500 to 36600 and sort them by insertion sort algorithm. (The comparison testing of C1).



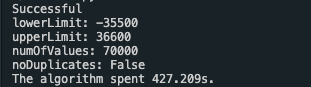
Test C2: Generate n = 60000 random values from -35500 to 36600 and sort them by insertion sort algorithm.



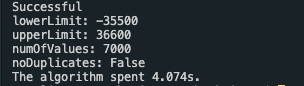
Testc2 Generate n =6000 random values from -35500 to 36600 and sort them by insertion sort algorithm. (The comparison testing of C2).



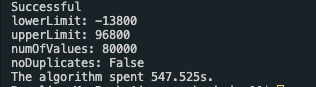
Test C3: Generate n = 70000 random values from -35500 to 36600 and sort them by insertion sort algorithm.



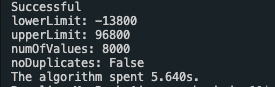
Testc3 Generate n =7000 random values from -35500 to 36600 and sort them by insertion sort algorithm. (The comparison testing of C3).



Test Cn: Generate n = 80000 random values from -13800 to 96800 and sort them by insertion sort algorithm.



Testcn Generate n =8000 random values from -13800 to 96800 and sort them by insertion sort algorithm. (The comparison testing of Cn).

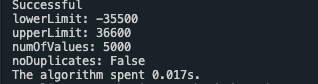


TestD1: Generate n = 50000 random values from -35500 to 36600 and sort them by quick sort algorithm.

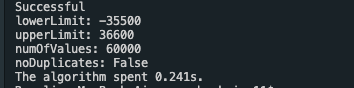
图片包含 文字

描述已自动生成

Testd1: Generate n = 5000 random values from -35500 to 36600 and sort them by quick sort algorithm. (The comparison testing of D1).



TestD2: Generate n = 60000 random values from -35500 to 36600 and sort them by quick sort algorithm.

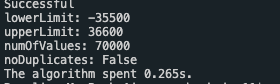


Testd2: Generate n = 6000 random values from -35500 to 36600 and sort them by quick sort algorithm. (The comparison testing of D2).

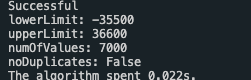
图片包含 文字

描述已自动生成

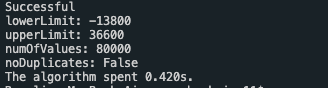
TestD3: Generate n = 70000 random values from -35500 to 36600 and sort them by quick sort algorithm.



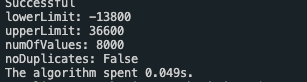
Testd3: Generate n = 7000 random values from -35500 to 36600 and sort them by quick sort algorithm. (The comparison testing of D3).



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



Testdn: Generate n = 8000 random values from -13800 to 96800 and sort them by quick sort algorithm. (The comparison testing of Dn).



b):

The source code of insertion sorting:

1. **import** random
2. **import** time
4. lowerLimit = -13800
5. upperLimit = 96800
6. numOfValues = 8000
7. noDuplicates = False
9. **def** saveToFile(resString, fileName):
10. file = open(fileName, mode = "w", encoding = "utf-8")
11. file.write(resString)
12. **print**("Successful")
13. file.close
15. **def** rNGenerator(lowerLimit, upperLimit, numOfValues, noDuplicates):
16. **if** noDuplicates == False:
17. resLis = []
18. **for** i **in** range(numOfValues):
19. res = random.randint(lowerLimit, upperLimit)
20. resLis.append(res)
21. **return** resLis
22. **else**:
23. resLis = []
24. **while** len(resLis) < numOfValues:
25. res = random.randint(lowerLimit, upperLimit)
26. **if** res **in** resLis:
27. **pass**
28. **else**:
29. resLis.append(res)
30. **return** resLis
32. **def** insertionSort(resList):
33. **for** i **in** range(1,len(resList)):
34. current = resList[i]
35. j = i
36. **while** j > 0 **and** resList[j-1] > current:
37. resList[j] = resList[j-1]
38. j -= 1
39. resList[j] = current
40. **return** resList
42. resListUnsort = rNGenerator(lowerLimit,upperLimit,numOfValues,noDuplicates)
43. startTime = time.time()
44. resLisSorted = insertionSort(resListUnsort)
45. saveToFile(str(resLisSorted),"Desktop/RandomNum\_Sorted.txt")
46. **print**("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates:",noDuplicates,"\nThe algorithm spent {:.3f}s.".format(time.time()-startTime))

The source code of quick sorting:

1. **import** random
2. **import** time
4. lowerLimit = -13800
5. upperLimit = 36600
6. numOfValues = 8000
7. noDuplicates = False
9. **def** saveToFile(resString, fileName):
10. file = open(fileName, mode = "w", encoding = "utf-8")
11. file.write(resString)
12. **print**("Successful")
13. file.close
15. **def** rNGenerator(lowerLimit, upperLimit, numOfValues, noDuplicates):
16. **if** noDuplicates == False:
17. resLis = []
18. **for** i **in** range(numOfValues):
19. res = random.randint(lowerLimit, upperLimit)
20. resLis.append(res)
21. **return** resLis
22. **else**:
23. resLis = []
24. **while** len(resLis) < numOfValues:
25. res = random.randint(lowerLimit, upperLimit)
26. **if** res **in** resLis:
27. **pass**
28. **else**:
29. resLis.append(res)
30. **return** resLis
32. **def** quickSort(resList):
33. **if** len(resList)<2:
34. **return** resList
35. mid = resList.pop(0)
36. left = []
37. right = []
38. **for** i **in** resList:
39. **if** i > mid:
40. right.append(i)
41. **else**:
42. left.append(i)
43. **return** quickSort(left) + [mid] + quickSort(right)
45. resListUnsort = rNGenerator(lowerLimit,upperLimit,numOfValues,noDuplicates)
46. startTime = time.time()
47. resLisSorted = quickSort(resListUnsort)
48. saveToFile(str(resLisSorted),"Desktop/RandomNum\_Sorted.txt")
49. **print**("lowerLimit: "+str(lowerLimit)+"\nupperLimit: "+str(upperLimit)+"\nnumOfValues: "+str(numOfValues)+"\nnoDuplicates:",noDuplicates,"\nThe algorithm spent {:.3f}s.".format(time.time()-startTime))

c).

Graph #1 – the timing results for Test C1 – Cn and the comparison testing of Test c1-cn (non-unique random values)

|  |  |  |
| --- | --- | --- |
|  | NumofValue | Time(s) |
| TestC1 | 50000 | 216.617 |
| Testc1(5000) | 5000 | 2.187 |
| TestC2 | 60000 | 320.31 |
| Testc2(6000) | 6000 | 3.258 |
| TestC3 | 70000 | 427.209 |
| Testc3(7000) | 7000 | 4.074 |
| TestCn | 80000 | 547.525 |
| Testcn(8000) | 8000 | 5.64 |

Graph #2 – the timing results for Test D1 – Dn and the comparison testing of Test d1-dn (non-unique random values)

|  |  |  |
| --- | --- | --- |
|  | NumofValue | Time(s) |
| TestD1 | 50000 | 0.188 |
| Testd1(5000) | 5000 | 0.017 |
| TestD2 | 60000 | 0.241 |
| Testd2(6000) | 6000 | 0.019 |
| TestD3 | 70000 | 0.265 |
| Testd3(7000) | 7000 | 0.022 |
| TestD4 | 80000 | 0.42 |
| Testd4(8000) | 8000 | 0.049 |

d).

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

**Task 3:**

a): The CPU of my computer is Intel Core i5 Duo Core, the base frequency is 1.4GHz and the turbo frequency is 2.7GHz. The frequency of the computer is kept at 2.7GHz all the time while the task 1-2 were running. When I use my classmate’s computer (CPU: Intel Core i7 7700 @ 3.6GHz 4 Core) to run my Algorithm, the running time of it cut down almost half, which is faster than my computer. Thus, the number of CPU in a machine significantly affect the performance of unknown algorithms, sorting algorithms and data structure.

b): The RAM of my computer is 4GB 1600MHz DDR3, the memory usage is very low while the task 1-2 were running. Thus, a greater RAM capacity doesn’t help improve the performance of unknown algorithms under a small size, but it will definitely affect the performance for input very huge size.

c): The Operating System running my programs play an important role, in maximizing my performance in Parts 1-3, because when I use MAC operating system and window operation system, the running time of my algorithm on MAC is faster than windows.