Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Мегафакультет трансляционных информационных технологий

Факультет инфокоммуникационных технологий

Дисциплина: Алгоритмы и структуры данных

**Отчет по Лабораторной работе №3**

Выполнила: Микулина Алиса Романовна

Группа: K3143, 1 курс

Преподаватель: Харьковская Татьяна Александровна

Санкт-Петербург

10.09.2021

**Описание задания**

**Задание 1.**

Написать код для сортировки массива слиянием, используя псевдокод из Лекции 2.

1. Написать код с сигнальными значениями;
2. Изменить код так, чтобы не использовались сигнальные значения.

**Задание 2.**

Модифицировать код сортировки массива слиянием, чтобы в выходном файле в первых строках отображались индексы граничных элементов областей слияния и их значения. В последней строке выводится отсортированный массив.

**Задание 4.**

Написать алгоритм бинарного поиска, который позволяет максимально эффективно искать элементы в массиве, при учете, что он отсортирован.

**Задание 5.**

Написать алгоритм, ищущий элемент, который появляется больше, чем n/2 раз, используя метод “разделяй и властвуй”.

**Задание 7.**

Написать код для поиска максимального подмассива за линейное время.

**Задание 10.**

1) Отредактировать код сортировки слиянием так, чтобы он не тратил время на поэлементное объединение двух массивов, если они уже расположены в порядке возрастания.

2) Определить, в какой момент сортировка слиянием начинает работать быстрее, чем сортировка вставками или выбором.

**Описание решения и исходный код**

**Задача 1.1.**

В данном коде я осуществила стандартную сортировку слиянием на основе псевдокода. В процессе работы алгоритма массив делится пополам, затем каждая из частей делится пополам, и т.д. до тех пор, пока разделенная часть не будет иметь хотя бы один элемент. Затем, с помощью функции merge, массивы объединяются, при этом отсортировываясь. Сортировка происходит до тех пор, пока не будут достигнуты “недостижимые” значения, которые мы используем в качестве сигнальных (10\*\*10). Сигнальные значения уже не сортируются и не добавляются в список.

def merge\_sorting(array, start, end):  
 if start < end:  
 middle = (start + end) // 2  
 merge\_sorting(array, start, middle)  
 merge\_sorting(array, middle+1, end)  
 return merge(array, start, middle, end)  
 else:  
 return True  
  
def merge(arr, start, middle, end):  
 left\_array, right\_array = arr[start:middle + 1:], arr[middle + 1:end + 1:]  
 count\_left, count\_right = 0, 0  
 maximum = 0  
 current = 0  
 for i in range(start, end + 1):  
 if find\_elem(left\_array, count\_left) < find\_elem(right\_array, count\_right):  
 arr[i] = left\_array[count\_left]  
 count\_left += 1  
 else:  
 arr[i] = right\_array[count\_right]  
 count\_right += 1  
 if i == start or arr[i] == arr[i - 1]:  
 current += 1  
 else:  
 maximum = current if current > maximum else maximum  
 current = 1  
 return maximum >= len(arr) / 2  
  
def find\_elem(ar, index):  
 if index < len(ar):  
 return ar[index]  
 else:  
 return 10\*\*10  
  
def sort(array):  
 return merge\_sorting(array, 0, len(array) - 1)  
  
f = open("input.txt", "r")  
length = int(f.readline())  
array = list(map(int, f.readline().split(" ")))  
f.close()  
  
d = open("output.txt", "w")  
d.write(str(int(sort(array))))  
d.close()

**Задача 1.3.**

Код практически идентичен предыдущему, но не использует сигнальных значений. Вместо этого мы используем вспомогательную функцию find\_elem(ar, index), с помощью которой мы находим либо следующий элемент списка, который нужно отсортировать, либо возвращаем число, ультимативно большее, чем элементы списка.

def merge\_sorting(array, start, end):  
 if start < end:  
 middle = (start + end) // 2  
 merge\_sorting(array, start, middle)  
 merge\_sorting(array, middle+1, end)  
 merge(array, start, middle, end)  
 return array  
 else:  
 return array  
  
def merge(arr, start, middle, end):  
 left\_array, right\_array = arr[start:middle + 1:], arr[middle + 1:end + 1:]  
 count\_left, count\_right = 0, 0  
 for i in range(start, end + 1):  
 if find\_elem(left\_array, count\_left) < find\_elem(right\_array, count\_right):  
 arr[i] = left\_array[count\_left]  
 count\_left += 1  
 else:  
 arr[i] = right\_array[count\_right]  
 count\_right += 1  
 return arr  
  
def find\_elem(ar, index):  
 if index < len(ar):  
 return ar[index]  
 else:  
 return 10\*\*10  
  
def sort(array):  
 return merge\_sorting(array, 0, len(array) - 1)  
  
f = open("input.txt", "r")  
length = int(f.readline())  
array = list(map(int, f.readline().split(" ")))  
f.close()  
  
d = open("output.txt", "w")  
d.write(' '.join(list(map(str,sort(array)))))  
d.close()

**Задача 2.**

Функциональная часть кода идентична коду из первого задания. Единственное отличие — поэтапный вывод индексов и их значений после осуществления каждого слияния. К индексам приходится добавлять единицу, т.к. в python итерация начинается с 0, а не с 1.

def merge\_sorting(arr, start, end, file):  
 if start < end:  
 middle = (start + end) // 2  
 merge\_sorting(arr, start, middle, file)  
 merge\_sorting(arr, middle + 1, end, file)  
 merge(arr, start, end, file)  
 return arr  
 else:  
 return arr  
  
  
def merge(arr, start, end, file):  
 middle = (start + end) // 2  
 left\_array, right\_array = arr[start:middle + 1:], arr[middle + 1:end + 1:]  
 count\_left, count\_right = 0, 0  
 for i in range(start, end + 1):  
 if find\_elem(left\_array, count\_left) < find\_elem(right\_array, count\_right):  
 arr[i] = left\_array[count\_left]  
 count\_left += 1  
 else:  
 arr[i] = right\_array[count\_right]  
 count\_right += 1  
 file.write(f'{start + 1} {end + 1} {arr[start]} {arr[end]} \n')  
 return arr  
  
  
def find\_elem(arr, index):  
 if index < len(arr):  
 return arr[index]  
 else:  
 return 10 \*\* 10  
  
  
def sort(arr, file):  
 return merge\_sorting(arr, 0, len(array) - 1, file)  
  
  
with open("input.txt", "r") as f:  
 length = int(f.readline())  
 array = list(map(int, f.readline().split(" ")))  
  
with open("output.txt", "w") as file:  
 sorted = sort(array, file)  
 file.write(' '.join(list(map(str, sorted))))

**Задача 4.**

Решение представляет собой алгоритм простейшего бинарного нерекурсивного поиска. При условии, что список отсортирован, можно брать средний элемент списка, сравнивать его с искомым числом и выбирать половинку списка, в которой будут продолжаться поиски. Обновляются значения end и start, цикл повторяется. Это происходит до тех пор, пока длина списка не станет равна единице. Если один оставшийся элемент равен искомому — ура! Мы выводим его индекс. Если же он не равен искомому числу, выводим -1.

def binary\_search(array, to\_find):  
 start = 0  
 end = len(array)  
 while end - start != 1:  
 middle = start + (end - start) // 2  
 if to\_find < array[middle]:  
 end = middle  
 else:  
 start = middle  
 if to\_find == array[start]:  
 return start  
 else:  
 return -1  
  
with open("input.txt", "r") as f:  
 length = int(f.readline())  
 array = list(map(int, f.readline().split(" ")))  
 to\_find\_num = int(f.readline())  
 to\_find = list(map(int, f.readline().split(" ")))  
  
with open("output.txt", "w") as file:  
 for i in range(to\_find\_num):  
 found = binary\_search(array, to\_find[i])  
 file.write(str(found) + ' ')

**Задача 5.**

В данной задаче воспользуемся тем, что в отсортированном списке одинаковые значения идут подряд. Таким образом, мы просто просчитываем количество вхождений одинакового числа при сортировке двух сливающихся массивов и, если число превышает максимальное число вхождений некого предыдущего числа, то максимальное значение обновляется. В конце сравнивается длина массива, деленная на два, и максимальное значение вхождений какого-либо числа. В результате сравнения получаем значение bool, которое выводим в виде числа.

def merge\_sorting(array, start, end):  
 if start < end:  
 middle = (start + end) // 2  
 merge\_sorting(array, start, middle)  
 merge\_sorting(array, middle+1, end)  
 return merge(array, start, middle, end)  
 else:  
 return True  
  
def merge(arr, start, middle, end):  
 left\_array, right\_array = arr[start:middle + 1:], arr[middle + 1:end + 1:]  
 count\_left, count\_right = 0, 0  
 maximum = 0  
 current = 0  
 for i in range(start, end + 1):  
 if find\_elem(left\_array, count\_left) < find\_elem(right\_array, count\_right):  
 arr[i] = left\_array[count\_left]  
 count\_left += 1  
 else:  
 arr[i] = right\_array[count\_right]  
 count\_right += 1  
 if i == start or arr[i] == arr[i - 1]:  
 current += 1  
 else:  
 maximum = current if current > maximum else maximum  
 current = 1  
 return maximum > len(arr) / 2  
  
def find\_elem(ar, index):  
 if index < len(ar):  
 return ar[index]  
 else:  
 return 10\*\*10  
  
def sort(array):  
 return merge\_sorting(array, 0, len(array) - 1)  
  
f = open("input.txt", "r")  
length = int(f.readline())  
array = list(map(int, f.readline().split(" ")))  
f.close()  
  
d = open("output.txt", "w")  
d.write(str(int(sort(array))))  
d.close()

**Задача 7.**

В данной задаче мы просто поочередно просчитываем линейные подмассивы, обновляя максимальное значение каждый раз, как current\_sum становится больше max\_sum. Если же current\_sum становится неньше нуля, значит нужно обнулить current\_sum, ибо дальше подсчет бесполезен.

def maximum\_array(array):  
 max\_sum = array[0]  
 current\_sum = 0  
 for i in array:  
 current\_sum += i  
 if current\_sum < 0:  
 current\_sum = 0  
 max\_sum = current\_sum if max\_sum < current\_sum else max\_sum  
 return max\_sum  
  
with open("input.txt", "r") as f:  
 length = int(f.readline())  
 array = list(map(int, f.readline().split(" ")))  
  
with open("output.txt", "w") as file:  
 maximum = maximum\_array(array)  
 file.write(str(maximum))

**Задача 10.1.**

В этой задаче мы, по сути, просто добавляем к обычной сортировке слиянием условие, при котором слияние будет проводиться, то бишь нам необходимо, чтобы последний элемент правого кусочка был больше первого массива второго кусочка, иначе мы просто не производим сортировку, так как сортировать уже нечего. Например, это значительно сокращает количество итераций и время выполнения, если на вход поступает уже отсортированный массив.

def merge\_sorting(array, start, end):  
 if start < end:  
 middle = (start + end) // 2  
 merge\_sorting(array, start, middle)  
 merge\_sorting(array, middle+1, end)  
 merge(array, start, middle, end)  
 return array  
 else:  
 return array  
  
def merge(arr, start, middle, end):  
 left\_array, right\_array = arr[start:middle + 1:], arr[middle + 1:end + 1:]  
 count\_left, count\_right = 0, 0  
 if left\_array[-1] > right\_array[0]:  
 for i in range(start, end + 1):  
 if find\_elem(left\_array, count\_left) < find\_elem(right\_array, count\_right):  
 arr[i] = left\_array[count\_left]  
 count\_left += 1  
 else:  
 arr[i] = right\_array[count\_right]  
 count\_right += 1  
 return arr  
  
def find\_elem(ar, index):  
 if index < len(ar):  
 return ar[index]  
 else:  
 return 10\*\*10  
  
def sort(array):  
 return merge\_sorting(array, 0, len(array) - 1)  
  
f = open("input.txt", "r")  
length = int(f.readline())  
array = list(map(int, f.readline().split(" ")))  
f.close()  
  
d = open("output.txt", "w")  
d.write(' '.join(list(map(str,sort(array)))))  
d.close()

**Задача 10.2.**

Проведя тесты, описанные ниже, я выяснила, что сортировка слиянием становится более эффективна, чем сортировки вставками и выбором когда n равно 43

|  |  |  |  |
| --- | --- | --- | --- |
| **Ввод** | **Вставкой** | **Выбором** | **Слиянием** |
| 41 to 1 | 0.001712100000000022 секунд | 0.00143589999999999 секунд | 0.0021537000000000084 секунд |
| 42 to 1 | 0.0013930000000000053 секунд | 0.0011941000000000035 секунд | 0.002076699999999987 секунд |
| ***43 to 1*** | ***0.0017910000000000148 секунд*** | ***0.001455999999999999 секунд*** | ***0.0013856000000000007 секунд*** |
| 44 to 1 | 0.0015272999999999953 секунд | 0.0016018000000000143 секунд | 0.0016287999999999997 секунд |
| 45 to 1 | 0.0015734999999999916 секунд | 0.0014157999999999948 секунд | 0.0017863999999999935 секунд |
| 46 to 1 | 0.0013968000000000036 секунд | 0.0014683000000000057 секунд | 0.0008068999999999993 секунд |

**Описание проведенных тестов.**

**1.1**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 1 to 1000 | 1 to 1000 | 0.004192899999999999 секунд |
| 1000 to 1 | 1 to 1000 | 0.004522800000000007 секунд |
| 1 8 2 1 4 7 3 2 3 6 | 1 1 2 2 3 3 4 6 7 8 | 0.0009220999999999951 секунд |

**1.3**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 1 to 1000 | 1 to 1000 | 0.006011300000000011 секунд |
| 1000 to 1 | 1 to 1000 | 0.008146399999999998 секунд |
| 1 8 2 1 4 7 3 2 3 6 | 1 1 2 2 3 3 4 6 7 8 | 0.0008766999999999942 секунд |

**2**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 1000 to 1020 | 1 6 1000 1005  7 8 1006 1007  7 9 1006 1008  10 11 1009 1010  7 11 1006 1010  1 11 1000 1010  12 13 1011 1012  12 14 1011 1013  15 16 1014 1015  12 16 1011 1015  17 18 1016 1017  17 19 1016 1018  20 21 1019 1020  17 21 1016 1020  12 21 1011 1020  1 21 1000 1020  1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 | 0.0008798 секунд |
| 1020 to 1000 | 1 2 1019 1020  1 3 1018 1020  4 5 1016 1017  1 5 1016 1020  6 7 1014 1015  6 8 1013 1015  9 10 1011 1012  6 10 1011 1015  1 10 1011 1020  11 12 1009 1010  11 13 1008 1010  14 15 1006 1007  11 15 1006 1010  16 17 1004 1005  16 18 1003 1005  19 20 1001 1002  16 20 1001 1005  11 20 1001 1010  1 20 1001 1020  1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 | 0.000973600000000005 секунд |
| 1 8 2 1 4 7 3 2 3 6 | 1 2 1 8  1 3 1 8  4 5 1 4  1 5 1 8  6 7 3 7  6 8 2 7  9 10 3 6  6 10 2 7  1 10 1 8  1 1 2 2 3 3 4 6 7 8 | 0.0007691000000000087 секунд |

**4**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 1 34 67 100 133 166 199 232 265 298 331 364 397 430 463 496 529 562 595 628 661 694 727 760 793 826 859 892 925 958 991 5 1 55 18 298 | 0 -1 -1 9 | 0.0019841999999999915 секунд |
| 1 3 43 604 1234 3444 6123 5 967 604 1 13 43 | -1 3 0 -1 2 | 0.0006885999999999975 секунд |
| 1 1 1 1 | 0 | 0.0007358999999999977 секунд |

**5**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 5 2 3 9 2 2 | 1 | 0.0014718999999999982 секунд |
| 4 1 2 3 4 | 0 | 0.0014676000000000133 секунд |
| 333 45 22 1 222 222 222 222 333 222 222 222 222 45 222 222 56 5 3 222 111 222 333 222 222 222 222 222 55 33 222 5 8 222 222 222 33 222 22 44444 22 222 | 1 | 0.0018458999999999837 секунд |

**7**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 333 45 22 1 222 222 222 222 333 222 222 222 222 45 222 222 56 5 3 222 111 222 333 222 222 222 222 222 55 33 222 5 8 222 222 222 33 222 22 44444 22 222 | 51015 | 0.0014051000000000063 секунд |
| 7 -7 | 7 | 0.0015653999999999946 секунд |
| 8 -7 225 | 226 | 0.0014456999999999942 секунд |

**10.1**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 1 to 1000 | 1 to 1000 | 0.002893500000000007 секунд |
| 1000 to 1 | 1 to 1000 | 0.006392900000000007 секунд |
| 1 8 2 1 4 7 3 2 3 6 | 1 1 2 2 3 3 4 6 7 8 | 0.0015556999999999932 секунд |

**Выводы по проделанной работе.**

При выполнении данной лабораторной работы я научилась работать с алгоритмами бинарного поиска и сортировки слиянием.

В определенный момент я написала решение для 4 задач, и они все в момент перестали работать. Пол часа ушло на поиск проблемы. Проблема: первое число (которое у нас типа n) не соответствовало длине вводимого массива, и при этом (удивительно! ибо обычно я его игнорирую) использовалось в коде.

Мною было решено избавиться от этого нерадивого элемента и забить на первое вводимое число практически во всех задачах. Длину массива и len(a) можно посчитать, а мозгам проще жить стало.

Осуждаю считывание длины как отдельное число.