**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Реализация и исследование алгоритма сортировки TimSort

|  |  |  |
| --- | --- | --- |
| Студент гр. 2381 |  | Дудкин М. В. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2023**Цель работы.**

**Реализовать алгоритм сортировки TimSort, протестировать на входных данных разного размера, провести исследование сложности алгоритма и сравнить с теоретическими данными**

**Задание**

**Имеется массив данных для сортировки int arr[] размера n**

**Необходимо отсортировать его алгоритмом сортировки Timsort по следующему критерию: по наименьшему значению квадрата элемента (в случае равенства значений элементов в квадрате - сортировка происходит по убыванию).**

**Так как Timsort - это гибридный алгоритм, содержащий в себе сортировку слиянием и сортировку вставками, то вам предстоит использовать оба этих алгоритма. Поэтому нужно выводить разделённые блоки, которые уже отсортированы сортировкой вставками.**

**Исследование**

**После успешного решения задачи в рамках курса проведите исследование данной сортировки на различных размерах данных (10/1000/100000), сравнив полученные результаты с теоретической оценкой (для лучшего, среднего и худшего случаев), и разного размера min\_run. Результаты исследования предоставьте в отчете.**

**Ход работы**

**Для выполнения работы была создана вспомогательная функция insertion\_sort. Она сортирует подмассивы по значению квадрата элемента. Основная функция - TimSort рекурсивно разбивает исходный массив на блоки размером min\_run элементов, далее эти блоки сортируются вставками с помощью функции insertion\_sort и сливаются в отсортированный массив.**

**Были проведены исследования для значений min\_run = 32 (Таблица 1) и min\_run = 100 (Таблица 2) и объемов данных 10, 1000, 100 000 элементов.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Длина массива** | **10** | **1000** | **100 000** |
| **Худший случай, c** | **0.000675** | **0.025872** | **1.355530** |
| **Средний случай, c** | **0.0** | **0.004928** | **0.421072** |
| **Лучший случай, c** | **0.0** | **0.002571** | **0.402871** |

**Таблица 1. (min\_run = 32)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Длина массива** | **10** | **1000** | **100 000** |
| **Худший случай, c** | **0.000903** | **0.029738** | **2.995572** |
| **Средний случай, c** | **0.0** | **0.003992** | **0.423843** |
| **Лучший случай, c** | **0.0** | **0.003990** | **0.346516** |

**Таблица 2. (min\_run = 100)**

**Выводы.**

Алгоритм сортировки TimSort был изучен и реализован на языке программирования python. В ходе работы было проведено исследования сложности алгоритма по времени в зависимости от объема входных данных и размера подмассива.

Приложение

Название файла main.py

import math

min\_run = 32

def insertion\_sort(arr, n):

for i in range(1, n):

temp = arr[i]

j = i - 1

while (j >= 0 and temp \*\* 2 < arr[j] \*\* 2):

arr[j + 1] = arr[j]

j = j - 1

while (j >= 0 and temp > arr[j] and temp == -arr[j]):

arr[j + 1] = arr[j]

j = j - 1

arr[j + 1] = temp

def TimSort(arr, n):

Len = n

if Len <= min\_run:

insertion\_sort(arr, n)

return arr

mid = math.ceil(Len // min\_run / 2) \* min\_run

arr\_1 = arr[:mid:]

TimSort(arr\_1, mid)

arr\_2 = arr[mid::]

TimSort(arr\_2, Len - mid)

p1, p2 = 0, 0

arr.clear()

while p1 != mid and p2 != (Len - mid):

if arr\_1[p1]\*\*2 < arr\_2[p2]\*\*2:

arr.append(arr\_1[p1])

p1 += 1

elif arr\_1[p1]\*\*2 > arr\_2[p2]\*\*2:

arr.append(arr\_2[p2])

p2 += 1

elif arr\_1[p1] < arr\_2[p2]:

arr.append(arr\_2[p2])

p2 += 1

else:

arr.append(arr\_1[p1])

p1 += 1

if p1 == mid:

arr.extend(arr\_2[p2::])

if p2 == (Len - mid):

arr.extend(arr\_1[p1::])

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input())

arr = list(map(int, input().split()))

TimSort(arr, n)

print(f"Answer:", \*arr)

Название файла tests.py

from main import \*

import random

def test\_insertion\_sort\_1():

arr = [8, 9, 2, 1, 3, 4, 5]

insertion\_sort(arr, len(arr))

assert arr == [1, 2, 3, 4, 5, 8, 9]

def test\_insertion\_sort\_2():

arr = [-1, 1, 0, 2, 1, 1, 1, -1, -2]

insertion\_sort(arr, len(arr))

assert arr == [0, 1, 1, 1, 1, -1, -1, 2, -2]

def test\_insertion\_sort\_3():

arr = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

insertion\_sort(arr, len(arr))

assert arr == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

def test\_TimSort\_1():

arr = [-5, -1, 6, -6, 0, -4, 7, -2, -2, 9, -2]

TimSort(arr, len(arr))

assert arr == [0, -1, -2, -2, -2, -4, -5, 6, -6, 7, 9]

def test\_TimSort\_2():

arr = [random.randint(-100, 100) for i in range(50)]

TimSort(arr, len(arr))

assert arr == sorted(arr, key=lambda x: x \*\* 2)

def test\_TimSort\_3():

arr = [5, 4, 3, 2, 1]

TimSort(arr, len(arr))

assert arr == [1, 2, 3, 4, 5]

test\_insertion\_sort\_1()

test\_insertion\_sort\_2()

test\_insertion\_sort\_3()

test\_TimSort\_1()

test\_TimSort\_2()

test\_TimSort\_3()