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, с	0.000675	0.025872	1.355530
Средний случай, с	0.0	0.004928	0.421072
Лучший случай, с	0.0	0.002571	0.402871

Таблица 1. (min_run = 32)

Длина массива	10	1000	100 000
Худший случай, с	0.000903	0.029738	2.995572
Средний случай, с	0.0	0.003992	0.423843
Лучший случай, с	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()
```