**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. 3344 |  | Тукалкин. В.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Создание алгоритма гибридной сортировки TimSort.

## Задание

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

Необходимо отсортировать его алгоритмом сортировки TimSort по убыванию модуля. 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.

Кратко алгоритм сортировки можно описать так:

1. Вычисление min\_run по размеру массива n (для упрощения отладки n уменьшается, пока не станет меньше 16, а не 64)

2. Разбиение массива на частично-упорядоченные (в т.ч. и по убыванию) блоки длины не меньше min\_run

3. Сортировка вставками каждого блока

4. Слияние каждого блока с сохранением инварианта и использованием галопа (галоп начинать после 3-х вставок подряд)

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.

Для исследования используйте стандартный алгоритм вычисления min\_run и начинайте галоп после 7-ми вставок подряд.

Примечание:

Нельзя пользоваться готовыми библиотечными функциями для сортировки, нужно сделать реализацию сортировки вручную.

Сортировка должна быть устойчивой.

Обратите внимание на пример.

Формат ввода

Первая строка содержит натуральное число n - размерность массива, следующая строка содержит элементы массива через пробел.

Формат вывода

Выводятся разделённые блоки для сортировки в формате "Part i: \*отсортированный разделённый массив\*"

Затем для каждого слияния выводится количество вхождений в режим галопа и получившийся массив в формате

"Gallops i: \*число вхождений в галоп\*

Merge i: \*итоговый массив после слияния\*"

Последняя строчка содержит финальный результат сортировки массива с надписью "Answer: "

## Выполнение работы

1. BinarySearch(element, array, left, right) – O(logN) – выполняет поиск element в array с начальными границами left и right.
2. InsertionSort(array):

В лучшем случае – O(N)

В среднем – O(N^2)

В наихудшем случае – O(N^2)

Сортирует входящий массив для функции SplitArray.

1. mergesort(leftArray, rightArray) – O(N) – производит слияние двух массивов.
2. minRunLength(n) – O(1) – функция расчёта число minrun, от которого будет считать последующих массивов.
3. PrintSubarrays(subarrays) – O(N) – выводит сообщение с подмассивами.
4. SplitArray(array, minrun) – O(N) – разделяет массив на подмассивы.
5. TimSort(lengthArray, array):

В лучшем случае – O(N)

В среднем – O(NlogN)

В наихудшем случае – O(NlogN)

Сортирует массив в обратном порядке и выводит промежуточные результаты.

## Тестирование программы

Тесты для проверки корректности работы реализованной сортировки TimSort находятся в файле tests.py. Каждый тест покрывает основные операции. Результаты исследования можно увидеть в таблице 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер данных | Данные | minrun | Время выполнения (сек) | Сложность |
| 10 | Отсортированные | 10 | 0,000001 | O(N) |
| 10 | Случайные | 10 | 0.000001 | O(NlogN) |
| 10 | Обратно отсортированный | 10 | 0.000001 | O(NlogN) |
| 1000 | Отсортированные | 63 | 0.002000 | O(N) |
| 1000 | Случайные | 63 | 0.013053 | O(NlogN) |
| 1000 | Обратно отсортированный | 63 | 0.040088 | O(NlogN) |
| 100000 | Отсортированные | 49 | 0.066467 | O(N) |
| 100000 | Случайные | 49 | 1.299887 | O(NlogN) |
| 100000 | Обратно отсортированный | 49 | 2.304103 | O(NlogN) |

Табл. 1 Результаты исследования

## Выводы

Был создан алгоритм сортировки TimSort.

# Приложение А Исходный код программы

Название файла: main.py

from modules.TimSort import TimSort

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

length = int(input())

inputArray = [int(x) for x in input().split()]

TimSort(length, inputArray)

Название файла: SplitArray.py

from modules.InsertionSort import InsertionSort

def SplitArray(array, minrun):

arrays = []

run = []

number\_of\_arrays = 0

fupp = 0

fdown = 0

size\_of\_array = 0

added\_elements = 0

while added\_elements < len(array):

x = array[added\_elements]

if len(run) > 0:

if (abs(x) > abs(run[-1]) and fupp == 0) or (abs(x) <= abs(run[-1]) and fdown == 0):

if abs(x) > abs(run[-1]):

fdown = 1

if abs(x) <= abs(run[-1]):

fupp = 1

run.append(x)

size\_of\_array += 1

else:

if size\_of\_array < minrun:

run.append(x)

size\_of\_array += 1

fupp = 1

fdown = 1

else:

InsertionSort(run)

arrays.append(run)

run = []

run.append(x)

number\_of\_arrays += 1

size\_of\_array = 1

fdown = 0

fupp = 0

else:

run.append(x)

size\_of\_array += 1

added\_elements += 1

InsertionSort(run)

arrays.append(run)

return arrays

Название файла: InsertionSort.py

def InsertionSort(array):

for i in range(1, len(array)):

tmp = array[i]

j = i - 1

while j >= 0 and abs(array[j]) < abs(tmp):

array[j + 1] = array[j]

j -= 1

array[j + 1] = tmp

Название файла: mergesort.py

from modules.BinarySearch import BinarySearch

def mergesort(leftArray, rightArray):

counterGallop = 0

mergedArray = []

leftArrayIndex = 0

rightArrayIndex = 0

leftArrayCount = 0

rightArrayCount = 0

while leftArrayIndex < len(leftArray) and rightArrayIndex < len(rightArray):

if abs(leftArray[leftArrayIndex]) >= abs(rightArray[rightArrayIndex]):

mergedArray.append(leftArray[leftArrayIndex])

leftArrayIndex += 1

leftArrayCount += 1

rightArrayCount = 0

else:

mergedArray.append(rightArray[rightArrayIndex])

rightArrayIndex += 1

rightArrayCount += 1

leftArrayCount = 0

if leftArrayCount == 3:

counterGallop += 1

index = BinarySearch(rightArray[rightArrayIndex], leftArray, leftArrayIndex - 1, len(leftArray))

for i in range(leftArrayIndex, index):

mergedArray.append(leftArray[i])

leftArrayIndex += 1

leftArrayCount = 0

rightArrayCount = 0

elif rightArrayCount == 3:

counterGallop += 1

index = BinarySearch(leftArray[leftArrayIndex], rightArray, rightArrayIndex - 1, len(rightArray))

for i in range(rightArrayIndex, index):

mergedArray.append(rightArray[i])

rightArrayIndex += 1

rightArrayCount = 0

leftArrayCount = 0

mergedArray.extend(leftArray[leftArrayIndex:])

mergedArray.extend(rightArray[rightArrayIndex:])

return mergedArray, counterGallop

Название файла: minRunLength.py

def minRunLength(n):

flag = 0

while n >= 16:

flag |= n & 1

n >>= 1

return n + flag

Название файла: BinarySearch.py

def BinarySearch(element, array, left, right):

while left < right:

mid = (left + right) // 2

if abs(array[mid]) >= abs(element):

left = mid + 1

else:

right = mid

return left

Название файла: PrintSubarrays.py

def PrintSubarrays(subarrays):

for i in range(len(subarrays)):

subarray = list(map(str, subarrays[i]))

subarray\_str = " ".join(subarray)

print(f"Part {i}: {subarray\_str}")

Название файла: TimSort.py

from modules.mergesort import mergesort

from modules.minRunLength import minRunLength

from modules.SplitArray import SplitArray

from modules.PrintSudarrays import PrintSubarrays

def TimSort(lengthArray, array):

splitArrays = SplitArray(array, minRunLength(lengthArray))

PrintSubarrays(splitArrays)

stack = []

gallopsI = []

mergeI = []

for element in splitArrays:

stack.append(element)

while len(stack) >= 3:

x = len(stack[-1])

y = len(stack[-2])

z = len(stack[-3])

if z <= x + y and y <= x:

if x >= z:

stack[-2], m = mergesort(stack[-2], stack[-3])

mergeI.append(stack[-2])

stack.pop(-3)

gallopsI.append(m)

else:

stack[-2], m = mergesort(stack[-2], stack[-1])

mergeI.append(stack[-2])

stack.pop()

gallopsI.append(m)

else:

break

while len(stack) == 2:

x = len(stack[-1])

y = len(stack[-2])

if y <= x:

stack[-2], m = mergesort(stack[-2], stack[-1])

gallopsI.append(m)

mergeI.append(stack[-2])

stack.pop()

else:

break

while len(stack) > 1:

if len(stack) == 2:

stack[-2], m = mergesort(stack[-2], stack[-1])

gallopsI.append(m)

mergeI.append(stack[-2])

stack.pop()

elif len(stack) >= 3:

x = len(stack[-1])

z = len(stack[-3])

if x >= z:

stack[-2], m = mergesort(stack[-2], stack[-3])

mergeI.append(stack[-2])

stack.pop(-3)

gallopsI.append(m)

else:

stack[-2], m = mergesort(stack[-2], stack[-1])

mergeI.append(stack[-2])

stack.pop()

gallopsI.append(m)

if len(gallopsI) == 0:

print()

for i in range(len(gallopsI)):

print(f"Gallops {i}: {gallopsI[i]}")

print(f"Merge {i}: {' '.join([str(x) for x in mergeI[i]])}")

print(f"Answer: {' '.join([str(x) for x in stack[0]])}")

return stack[0]

Название файла: tests.py

from modules.minRunLength import minRunLength

from modules.BinarySearch import BinarySearch

from modules.SplitArray import SplitArray

from modules.mergesort import mergesort

from modules.InsertionSort import InsertionSort

from modules.TimSort import TimSort

def test\_minRunLength():

assert minRunLength(64) == 8

def test\_BinarySearch():

assert BinarySearch(7, [2, 7, 4, -1, 46, 0, -8, 4, 8, 45, 324, 9], 0, 12) == 12

def test\_SplitArray():

assert SplitArray([-1, 2, 3, 4, 5, -6, 7, 8, -8, -8, 7, -7, 7, 6, -5, 4], 8) == [[8, 7, -6, 5, 4, 3, 2, -1], [-8, -8, 7, -7, 7, 6, -5, 4]]

def test\_mergesort():

assert mergesort([8, 7, -6, 5, 4, 3, 2, -1], [-8, -8, 7, -7, 7, 6, -5, 4]) == ([8, -8, -8, 7, 7, -7, 7, 6, -6, 5, -5, 4, 4, 3, 2, -1], 1)

def test\_InsertionSort():

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

InsertionSort(arr)

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

def test\_TimSort():

assert TimSort(16, [-1, 2, 3, 4, 5, -6, 7, 8, -8, -8, 7, -7, 7, 6, -5, 4]) == [8, -8, -8, 7, 7, -7, 7, 6, -6, 5, -5, 4, 4, 3, 2, -1]

test\_minRunLength()

test\_BinarySearch()

test\_SplitArray()

test\_mergesort()

test\_InsertionSort()

test\_TimSort()