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: TimSort

Студент гр. 3384 _____ Пьянков М.Ф.

Преподаватель _____ Шестопалов Р.П.

Санкт-Петербург
2024

Цель работы

Изучение и реализация сортировки Timsort, а также соответствующих методов и алгоритмов, необходимых для реализации алгоритма. Сравнение времени работы алгоритма на разных входных данных.

Задание

Тема лабораторной работы: Реализация и исследование алгоритма сортировки TimSort.

Реализация:

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:

Вычисление min_run по размеру массива n (для упрощения отладки n уменьшается, пока не станет меньше 16, а не 64)

Разбиение массива на частично-упорядоченные (в т.ч. и по убыванию) блоки длины не меньше min_run

Сортировка вставками каждого блока

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: "

Пример #1 (min_run = 10)

Ввод

20

1 -2 3 -4 5 6 -7 -8 9 -10 11 -10 -9 8 7 -7 -6 6 5 4

Вывод

Part 0: 11 -10 9 -8 -7 6 5 -4 3 -2 1

Part 1: -10 -9 8 7 -7 -6 6 5 4

Gallops 0: 0

Merge 0: 11 -10 -10 9 -9 -8 8 -7 7 -7 6 -6 6 5 5 -4 4 3 -2 1

Answer: 11 -10 -10 9 -9 -8 8 -7 7 -7 6 -6 6 5 5 -4 4 3 -2 1

Пример #2 (min_run = 8)

Ввод

16

-1 2 3 4 5 -6 7 8 -8 -8 7 -7 7 6 -5 4

Вывод

Part 0: 8 7 -6 5 4 3 2 -1

Part 1: -8 -8 7 -7 7 6 -5 4

Gallops 0: 1

Merge 0: 8 -8 -8 7 7 -7 7 6 -6 5 -5 4 4 3 2 -1

Answer: 8 -8 -8 7 7 -7 7 6 -6 5 -5 4 4 3 2 -1

Выполнение работы

Реализованный алгоритм сортировки Timsort можно представить в виде нескольких частей:

1)Разделение исходного массива на run-ы, максимальная длина которых определяется функцией min_run(), а также их сортировка вставками во время формирования(основное тело функции timsort()) и запись в стек.

2)Слияние массивов из стека, осуществляемое с помощью функции stack_merging().

3)Внутри функции stack_merging() вызывается функция merge() для соответствующих элементов стека.

4)Внутри функции merge() происходит модифицированный алгоритм слияния: элементы поочерёдно добавляются в итоговый массив res, но как только количество добавленных элементов из конкретного массива достигает количества gallop_size, алгоритм переходит в фазу «галопа».

5)Фаза «галопа»: запускается модифицированный алгоритм бинарного поиска для поиска индекса, до которого можно добавлять все элементы из конкретного массива.

6)Внутри модифицированного алгоритма бинарного поиска возведением в квадрат ищется промежуток, в котором может находится элемент с необходимым индексом. Далее запускается стандартный алгоритм бинарного поиска и возвращается нужный индекс.

7)В конце работы функции stack_merging() получаем один элемент в стеке, который и является итоговым ответом.

Разработанный программный код см. в приложении А.

Исследование

Для проведения исследования были подготовлены 4 типа данных: 2 — полностью отсортированных массива, 1 массив с случайными числами и 1 массив с упорядоченными списками внутри.

	10	1000	100000
Массив, упорядоченный по убыванию	2.5779e-6	0,00044	0,0046
Массив, упорядоченный по возрастанию	2.6854e-6	0,00061	0,0035
Массив случайных чисел с упорядоченными подмассивами	2.7123e-6	0,00220	0,0191
Массив случайных чисел	2.8257e-6	0,00320	0,0430

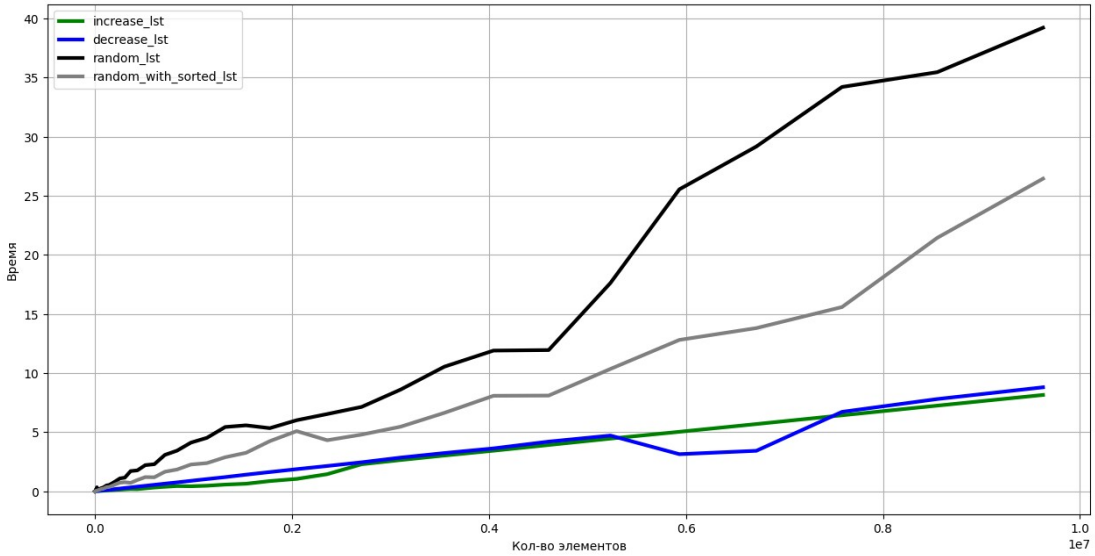

Рисунок 1 - Графики скорости работы Timsort на разных наборах данных (до 1e7 элементов)

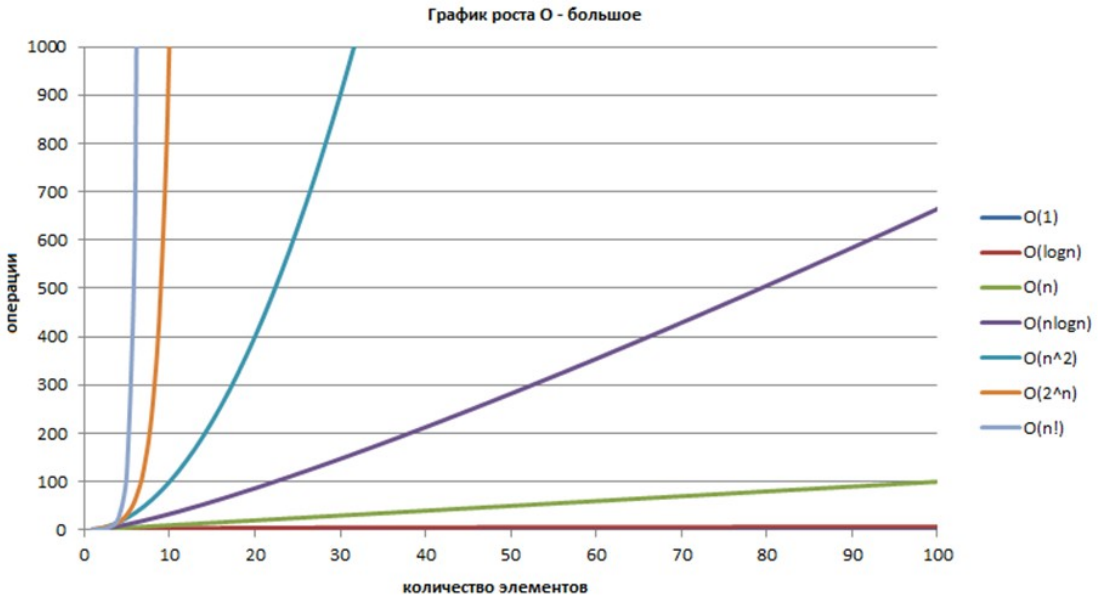

Рисунок 2 — Графики сложности работы алгоритма (O(f(x)))

Исходя из графиков видно, что алгоритм сортировки хуже всего справляется с массивами, состоящими из полностью случайных чисел, без упорядоченных подмассивов. Также обратим внимание на то, что на графике проглядывается сложность алгоритма в O(nlogn) (рис 2.) на массиве с случайными числами и массиве с упорядоченными подмассивами (больше на массиве с абсолютно случайными числами, на рис 1.) – среднее и худшее время работы алгоритма. На убывающем и возрастающем массивах видна сложность O(n) – лучшее время работы алгоритма.

Выводы

Был успешно изучен и реализован новый вид сортировки — Timsort. Поставленная задача решена, также освоены другие типы сортировок, такие как: сортировка вставками, сортировка слиянием, сортировка подсчётом, быстрая сортировка, а также алгоритмы бинарного поиска и слияния и их модификации.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main_lb2.py

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

def tim_sort(arr):
    global gallops_arr
    gallops_arr = []
    if len(arr) == 0:
        return None
    if len(arr) == 1:
        print(f'Part 0: {arr[0]}')
        return str(arr[0])

    def min_run(size):
        res = 0
        while(size >= 16):
            res |= (size & 1)
            size >>= 1
        return res + size

    def bin_search(ptr, data, elem):
        if abs(data[0]) < abs(elem):
            return 0
        ind = ptr
        while(ind < len(data) and abs(data[ind]) >= abs(elem)):
            ind *= 2
        right = min(ind, len(data) - 1)
        left = ptr
        while (True):
            if left > right:
                return len(data)
            mid = (left + right) // 2
            if abs(data[mid]) > abs(elem):
                left = mid + 1
            elif abs(data[mid]) < abs(elem):
                if mid - 1 >= 0 and abs(data[mid - 1]) >= abs(elem):
                    return mid
                right = mid - 1
            else:
                left = mid + 1

    def merge(arr1, arr2, gallop_size=3):
        global gallops_arr
        gallops = 0
        ptr1 = 0
        cnt1 = 0
        ptr2 = 0
        cnt2 = 0
        res = []
        while (ptr1 < len(arr1) and ptr2 < len(arr2)):
            if abs(arr1[ptr1]) >= abs(arr2[ptr2]):
```

```
                res.append(arr1[ptr1])
                ptr1 += 1
                cnt1 += 1
                cnt2 = 0
                if cnt1 == gallop_size:
                    gallops += 1
                    index = bin_search(ptr1, arr1, arr2[ptr2])
                    res += arr1[ptr1:index]
                    cnt1 = 0
                    cnt2 = 0
                    ptr1 = index
            elif abs(arr1[ptr1]) < abs(arr2[ptr2]):
                res.append(arr2[ptr2])
                ptr2 += 1
                cnt2 += 1
                cnt1 = 0
                if cnt2 == gallop_size:
                    gallops += 1
                    index = bin_search(ptr2, arr2, arr1[ptr1])
                    res += arr2[ptr2:index]
                    cnt1 = 0
                    cnt2 = 0
                    ptr2 = index
        res += arr1[ptr1:]
        res += arr2[ptr2:]
        gallops_arr.append(gallops)
        return res

    def stack_merging(stk):
        data = stk
        stk = []
        cnt = 0
        for i in data:
            stk.append(i)
            while (len(stk) > 1):
                   if len(stk) > 2 and not (stk[-3][0] > stk[-1][0] +
stk[-2][0]):
                    if stk[-3][0] < stk[-1][0]:
                        stk[-2][1] = merge(stk[-3][1],stk[-2][1])
                        print(f'Gallops {cnt}: {gallops_arr[cnt]}')
                            print(f'Merge {cnt}: ' + ' '.join(map(str,
stk[-2][1])))
                        cnt += 1
                        stk[-2][0] = len(stk[-2][1])
                        stk.pop(-3)
                    else:
                        stk[-2][1] = merge(stk[-2][1], stk[-1][1])
                        print(f'Gallops {cnt}: {gallops_arr[cnt]}')
                            print(f'Merge {cnt}: ' + ' '.join(map(str,
stk[-2][1])))
                        cnt += 1
                        stk[-2][0] = len(stk[-2][1])
                        stk.pop(-1)

                if not (stk[-2][0] > stk[-1][0]):
                    stk[-2][1] = merge(stk[-2][1], stk[-1][1])
```

```
                    print(f'Gallops {cnt}: {gallops_arr[cnt]}')
                    print(f'Merge {cnt}: ' + ' '.join(map(str, stk[-2]
[1])))
                    cnt += 1
                    stk[-2][0] = len(stk[-2][1])
                    stk.pop(-1)
                else:
                    break


        while (len(stk) > 1):
            if len(stk) > 2:
                if stk[-3] < stk[-1]:
                    stk[-2][1] = merge(stk[-3][1], stk[-2][1])
                    print(f'Gallops {cnt}: {gallops_arr[cnt]}')
                    print(f'Merge {cnt}: ' + ' '.join(map(str, stk[-2]
[1])))
                    cnt += 1
                    stk[-2][0] = len(stk[-2][1])
                    stk.pop(-3)
                else:
                    stk[-2][1] = merge(stk[-2][1], stk[-1][1])
                    print(f'Gallops {cnt}: {gallops_arr[cnt]}')
                    print(f'Merge {cnt}: ' + ' '.join(map(str, stk[-2]
[1])))
                    cnt += 1
                    stk[-2][0] = len(stk[-2][1])
                    stk.pop(-1)
            if len(stk) > 1:
                stk[-2][1] = merge(stk[-2][1], stk[-1][1])
                print(f'Gallops {cnt}: {gallops_arr[cnt]}')
                    print(f'Merge {cnt}: ' + ' '.join(map(str, stk[-2]
[1])))
                cnt += 1
                stk[-2][0] = len(stk[-2][1])
                stk.pop(-1)
        return stk

    stk = []

    max_size = min_run(len(arr))
    run = []

    increase = 1
    start = 0
    while(start < len(arr)):

        if len(run) >= 2 and increase:
            if abs(arr[start]) > abs(run[-1]):
                run.append(arr[start])
                start += 1
                 elif abs(arr[start]) <= abs(run[-1]) and len(run) <
max_size:
                for i in range(len(run)):
                    if abs(run[i]) >= abs(arr[start]):
                        run.insert(i, arr[start])
```

```
                        start += 1
                        break
            else:
                stk.append([len(run), run[::-1]])
                run = []
        elif len(run) >= 2 and not(increase):
            if abs(arr[start]) <= abs(run[-1]):
                run.append(arr[start])
                start += 1
                  elif abs(arr[start]) > abs(run[-1]) and len(run) <
max_size:
                for i in range(len(run)):
                    if abs(run[i]) < abs(arr[start]):
                        run.insert(i, arr[start])
                        start += 1
                        break
            else:
                stk.append([len(run), run])
                run = []

        if (len(run) < 2):
            run.append(arr[start])
            start += 1

        if len(run) == 2:
            if abs(run[1]) > abs(run[0]):
                increase = 1
            else:
                increase = 0

        if start == len(arr):
            if increase:
                stk.append([len(run), run[::-1]])
            else:
                stk.append([len(run), run])
            break

    for i in range(len(stk)):
        s = ' '.join(map(str, stk[i][1]))
        print(f'Part {i}: ' + s)

    stk = stack_merging(stk)

    return(' '.join(map(str, stk[0][1])))

print('Answer: ' + tim_sort(arr))
```