САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2
по курсу «Алгоритмы и структуры данных»
Тема: Сортировка

Выполнил:
Юшков А.М.
К3139

Проверил:

Санкт-Петербург
2024 г.

1 задача. Сортировка слиянием

1. Используя *псевдокод* процедур `Merge` и `Merge-sort` из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько рандомных массивов, подходящих под параметры:

   - **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 2 \cdot 10^4$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
   - **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
   - Ограничение по времени. 2сек.
   - Ограничение по памяти. 256 мб.

2. Для проверки можно выбрать наихудший случай, когда сортируется массив размера 1000, $10^4, 10^5$ чисел порядка 10^9, отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.

3. Перепишите процедуру `Merge` так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или R скопированы обратно в массив A, после чего в этот массив копируются элементы, оставшиеся в непустом массиве.

или перепишите процедуру `Merge` (и, соответственно, `Merge-sort`) так, чтобы в ней не использовались значения границ и середины - p, r и q.

Код

```
def merge_sort(a):
    if len(a) > 1:
        mid = len(a) // 2
        left_half = a[:mid]
        right_half = a[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                a[k] = left_half[i]
                i += 1
            else:
                a[k] = right_half[j]
                j += 1
            k += 1
        while i < len(left_half):
            a[k] = left_half[i]
            i += 1
            k += 1
```

```
        while j < len(right_half):
            a[k] = right_half[j]
            j += 1
            k += 1
with open("input.txt", "r") as file:
    n = int(file.readline())
    arr = [int(i) for i in file.readline().split()]
merge_sort(arr)
print(arr)
```

Пояснение к коду:

1)Считываем из файла данные

2)Разделяем пассив на левую и правую части

3)рекурсивно сортируем левые и правые части

4)Заполняем массив

Вывод по задаче: В ходе выполнения задачи я узнал как происходит сортировка слиянием, вспомнил циклы, функции, условные операторы и методы ввода-вывода

2 задача. Сортировка слиянием+

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания *с помощью сортировки слиянием*.

Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

Код

```
def merge(left, right):
    merged = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged

def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    merged = merge(left, right)
    print("Слияние:", left, right)
    print("Значение:", merged)
    return merged

with open("input.txt", "r") as file:
```

```
    n = int(file.readline())
    a = [int(i) for i in file.readline().split()]
a = merge_sort(a)
print(a)
```

Пояснение к коду: В отличие от предыдущей задачи функция сортировки левой и правой части вынесена отдельно, выводится промежуточные шаги сортировки

3 задача. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда $i < j$, а $A_i > A_j$. Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной. Например, в сортированном массиве число инверсий равно 0, а в массиве, сортированном наоборот - каждые два элемента будут составлять инверсию (всего $n(n-1)/2$).

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
- **Формат выходного файла (output.txt).** В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

Код:

```
def merge_and_count(arr, temp_arr, left, mid, right):
    i = left
    j = mid + 1
    k = left
    inv_count = 0
    while i <= mid and j <= right:
        if arr[i] <= arr[j]:
            temp_arr[k] = arr[i]
            i += 1
        else:
            temp_arr[k] = arr[j]
            inv_count += (mid - i + 1)
            j += 1
        k += 1
    while i <= mid:
        temp_arr[k] = arr[i]
        i += 1
        k += 1
    while j <= right:
        temp_arr[k] = arr[j]
        j += 1
        k += 1
    for i in range(left, right + 1):
        arr[i] = temp_arr[i]
```

```
    return inv_count
def merge_sort_and_count(arr, temp_arr, left, right):
    inv_count = 0
    if left < right:
        mid = (left + right) // 2
        inv_count += merge_sort_and_count(arr, temp_arr, left, mid)
        inv_count += merge_sort_and_count(arr, temp_arr, mid + 1, right)
        inv_count += merge_and_count(arr, temp_arr, left, mid, right)
    return inv_count
with open("input.txt", "r") as file:
    n = int(file.readline())
    a = [int(i) for i in file.readline().split()]
n = len(a)
temp_array = [0] * n
result = merge_sort_and_count(a, temp_array, 0, n - 1)
print("Количество инверсий:", result)
```

Пояснение к коду:

1)Считывем данные из файла

2)Вызываем функцию, ей на вход подается левая и правая границы и два массива

3)Сортируем массив и считаем количество инверсий

Вывод по задаче: В ходе выполнения задачи я узнал как происходит сортировка слиянием, как посчитать кол-во инверсий, вспомнил циклы, функции, условные операторы и методы ввода-вывода

4 задача. Бинарный поиск

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве, и последовательность $a_0 < a_1 < ... < a_{n-1}$ из n **различных** положительных целых чисел в порядке возрастания, $1 \leq a_i \leq 10^9$ для всех $0 \leq i < n$. Следующая строка содержит число k, $1 \leq k \leq 10^5$ и k положительных целых чисел $b_0, ...b_{k-1}$, $1 \leq b_j \leq 10^9$ для всех $0 \leq j < k$.
- **Формат выходного файла (output.txt).** Для всех i от 0 до $k-1$ вывести индекс $0 \leq j \leq n-1$, такой что $a_i = b_j$ или -1, если такого числа в массиве нет.

Код

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
```

```
    return -1

with open("input.txt", "r") as file:
    n1 = int(file.readline())
    a = [int(i) for i in file.readline().split()]
    n2 = int(file.readline())
    b = [int(i) for i in file.readline().split()]
c = []
for i in range(len(b)):
    c.append(binary_search(a, b[i]))
print(c)
```

Пояснение к коду

1)Считываем данный из файла

2)запускаем цикл поиска для каждого элемента массива

3)С помощью алгоритма бинарного поиска ищем число в массиве

5 задача. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов $a_1, a_2, ...a_n$, и нужно проверить, содержит ли она элемент, который появляется больше, чем $n/2$ раз. Наивный метод это сделать:

```
import time, psutil, random
t_start = time.perf_counter()
def majority_element(nums):
    candidate = None
    count = 0
    for num in nums:
        if count == 0:
            candidate = num
            count = 1
        elif num == candidate:
            count += 1
        else:
            count -= 1
    if candidate is not None and nums.count(candidate) > len(nums) // 2:
        return True
    else:
        return False

with open("input.txt", "r") as file:
    n = int(file.readline())
    a = [int(i) for i in file.readline().split()]
if majority_element(a):
    print('1')
else:
    print('0')
```

Пояснение

1)Считываем данный из файла

2)Ищем кандидата

3)Проверяем является ли предположительный кандидат кандидатом

Вывод

В ходе выполнения лабораторной работы были изучены методы сортировки слиянием, бинарный поиск и поиск представителя большинства