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

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Тема работы
Вариант 1

Выполнил:
Барецкий М.С.
К3141

Проверила:
Афанасьев А. А.

Санкт-Петербург
2024 г.

Содержание отчета

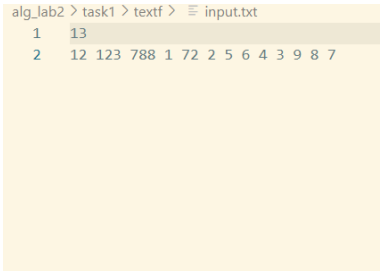
Задачи по варианту

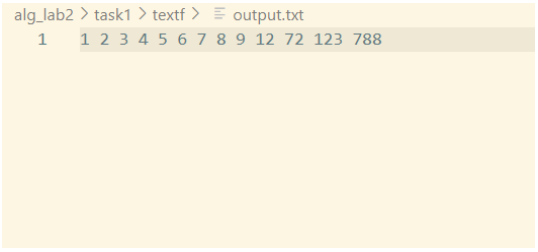
Задача №1. Название задачи [N баллов]

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

Листинг кода. (именно листинг, а не скрины)

```
def merge(arr, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid

    L = [0] * (n1 + 1)
    R = [0] * (n2 + 1)

    for i in range(0, n1):
        L[i] = arr[left + i]
    for j in range(0, n2):
        R[j] = arr[mid + 1 + j]

    L[n1] = float('inf')
    R[n2] = float('inf')

    i = 0
    j = 0
    k = left
    while k <= right:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

def merge_sort(arr, left, right):
    if left < right:
        mid = (left + right) // 2
        merge_sort(arr, left, mid)
        merge_sort(arr, mid + 1, right)
        merge(arr, left, mid, right)
    return arr
```





Этот код реализует алгоритм сортировки слиянием, который рекурсивно разделяет массив на подмассивы и затем сливает их в отсортированном порядке. Время работы алгоритма — O(n log n).

```
def merge_without_inf(arr, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid

    L = arr[left:mid+1]
    R = arr[mid+1:right+1]

    i = j = 0
    k = left

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]

            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

def merge_sort_without(arr, left, right):
    if left < right:
        mid = (left + right) // 2
        merge_sort_without(arr, left, mid)
```

```
        merge_sort_without(arr, mid + 1, right)
        merge_without_inf(arr, left, mid, right)
```

alg_lab2 > task1 > textf > input_without.txt

```
1   10
2   1 9 2 8 3 7 4 7 6 2
```

alg_lab2 > task1 > textf > output_without.txt

```
1   1 2 2 3 4 6 7 7 8 9
```

Этот код реализует сортировку слиянием без использования "бесконечности" для упрощения слияния. Функция merge_without_inf выполняет слияние двух отсортированных подмассивов, копируя их в новые массивы L и R, а затем сливая их обратно в основной массив. В функции merge_sort_without происходит рекурсивное деление массива на подмассивы и их сортировка, после чего сливаются отсортированные части с помощью merge_without_inf.

	Время выполнения	Затраты памяти
Пример из задачи	0.006703900002321461	0.01731586456298828 МБ
Пример из задачи(Без бесконечности)	0.0009927999999490567	0.017304420471191406

Вывод по задаче: Сортировка слиянием без бесконечности быстрее обычной.

Дополнительные задачи

Задача №4. 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.

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0
    while low <= high:
        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid
    return -1
```

alg_lab2 > task4 > textf > input.txt

```
5
1 5 8 12 13
5
8 1 23 1 11
```

alg_lab2 > task4 > textf > output.txt

```
2 0 -1 0 -1
```

Этот код реализует бинарный поиск для нахождения элемента x в отсортированном массиве arr. Функция последовательно сокращает диапазон поиска, сравнивая элемент с серединой массива, и возвращает индекс элемента, если он найден. Если элемент отсутствует, функция возвращает -1.

	Время выполнения	Затраты памяти
Пример из задачи	0.004600500000378815 5	0.017362594604492188

Вывод по задаче: Бинарный поиск — это эффективный алгоритм для поиска элемента в отсортированном массиве. Он работает по принципу "разделяй и властвуй", постепенно сужая область поиска.

Задача №5. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a1, a2, ...an, и нужно проверить, содержит ли она элемент, который появляется больше, чем n/2 раз

```
def count_occurrences(array, target, left, right):
    count = 0
    for i in range(left, right + 1):
        if array[i] == target:
            count += 1
    return count

def find_majority_candidate(array, left, right):
    if left == right:
        return array[left]

    mid = (left + right) // 2

    left_candidate = find_majority_candidate(array, left, mid)
    right_candidate = find_majority_candidate(array, mid + 1, right)

    if left_candidate == right_candidate:
        return left_candidate

    left_count = count_occurrences(array, left_candidate, left, right)
       right_count  =  count_occurrences(array,  right_candidate,  left,
right)

    if left_count > right_count:
        return left_candidate
    return right_candidate

def is_majority_element(array):
    n = len(array)
    candidate = find_majority_candidate(array, 0, n - 1)

    if candidate is not None:
        total_count = count_occurrences(array, candidate, 0, n - 1)
        if total_count > n // 2:
            return 1
```

```
    return 0
```

Этот код находит **преобладающий элемент** в массиве. Функция find_majority_candidate рекурсивно ищет кандидата, а count_occurrences подсчитывает его вхождения. В is_majority_element проверяется, встречается ли кандидат более чем в половине элементов массива. Если да — возвращает 1, иначе — 0.

alg_lab2 > task5 > textf > input.txt

```
1  5
2  2 3 9 2 2
```

alg_lab2 > task5 > textf > output.txt

```
1  1
```

	Время выполнения	Затраты памяти
Пример из задачи	0.005433499998616753	0.017192840576171875

Задача №7.Поиск максимального подмассива за линейное время

Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отслеживая найденный к данному моменту максимальный подмассив. Зная максимальный подмассив массива A[1..j], распространите ответ на поиск максимального подмассива, заканчивающегося индексом j + 1, воспользовавшись следующим наблюдением: максимальный подмассив массива A[1..j + 1] представляет собой либо максимальный подмассив массива A[1..j], либо подмассив A[i..j + 1] для некоторого $1 \le i \le j + 1$. Определите максимальный подмассив вида A[i..j + 1] за константное время, зная максимальный подмассив, заканчивающийся индексом j

```python
def find_max_subarray(arr):
    max_sum = current_sum = 0
    start = end = 0
    start_index = 0
    max_subarray = []

    for i, num in enumerate(arr):
        current_sum += num
        if current_sum > max_sum:
            max_sum = current_sum
            start = start_index
            end = i
            max_subarray = arr[start:end + 1]
        if current_sum < 0:
            current_sum = 0
            start_index = i + 1

    return max_subarray
```

Этот код находит **максимальный подмассив** с наибольшей суммой в массиве, используя алгоритм Кадана. Он отслеживает текущую сумму подмассива, обновляет максимальную сумму и сохраняет индексы подмассива, если текущая сумма больше. Если сумма становится отрицательной, начинается новый подмассив. Возвращается сам максимальный подмассив.

alg_lab2 > task7 > textf > input.txt

```
1    -1 -1 -1 -1 5 5 5 -1 -1 -1 -1 -1 -1 5
```

alg_lab2 > task7 > textf > output.txt

```
1    5 5 5
```

	Время выполнения	Затраты памяти

Пример из задачи	0.0.00044329999946057796	0.017253875732421875

Задача №10.

Реализуйте сортировку слиянием, учитывая, что можно сэкономить на отсортированных массивах, которые не нужно объединять. Проверьте A[q], меньше он или равен A[q + 1], и объедините их, только если A[q] > A[q + 1], где q - середина при делении в Merge_Sort.

```
def merge(arr, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid

    L = [0] * (n1 + 1)
    R = [0] * (n2 + 1)

    for i in range(0, n1):
        L[i] = arr[left + i]
    for j in range(0, n2):
        R[j] = arr[mid + 1 + j]

    L[n1] = float('inf')
    R[n2] = float('inf')

    i = 0
    j = 0
    k = left
    while k <= right:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

def merge_sort(arr, left, right):
    if left < right:
        mid = (left + right) // 2
        merge_sort(arr, left, mid)
        merge_sort(arr, mid + 1, right)
        if arr[mid] > arr[mid + 1]:
```

```
            merge(arr, left, mid, right)
    return arr
```

Этот код реализует сортировку слиянием с оптимизацией: перед слиянием проверяется, нужно ли его выполнять (если элементы уже отсортированы). Функция merge сливает два подмассива, а merge_sort рекурсивно делит массив и выполняет слияние только при необходимости.

alg_lab2 > task10 > textf > input.txt

```
10
1 9 2 8 3 7 4 7 6 1
```

alg_lab2 > task10 > textf > output.txt

```
1 1 2 3 4 6 7 7 8 9
```

	Время выполнения	Затраты памяти
Пример из задачи	0.005639499999233521 5	0.017202377319335938

Вывод

(по всей лабораторной)