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: Сортировка вставками, выбором, пузырьковая.

Выполнил:
Мезенцев Богдан
К3139
Проверил:
Афанасьев Антон Владимирович

Санкт-Петербург
2024 г.

Содержание отчёта

Задачи по варианту

Задание 1. Сортировка вставкой

1) В начале нам необходимо получить данные из входного файла(input.txt)

```
file = open('input.txt', 'r')
n = int(file.readline())
S = list(map(int, file.readline().split()))
```

Получаем две переменные *S* - не отсортированный массив, *n* - количество элементов массива.

2) Потом проверяем введенные данные и пишем алгоритм сортировки вставками для массива *S*. Запускаем цикл *for i* и цикл *for j*, где *i*, *j* - индексы сравниваемых элементов массива. В результате сравнения соседних элементов меняем их местами или оставляем на своих местах.

```
if 1 <= n <= 10**3:
    if any(abs(int(m)) > 10**9 for m in S):
        print("Введите корректные данные")
    else:
        for i in range(1, len(S)):
            for j in range(i, 0, -1):
                if S[j] < S[j - 1]:
                    S[j], S[j - 1] = S[j - 1], S[j]
                else:
                    break
else:
    print('Количество элементов должно быть больше чем 1 и меньше чем 10**3')
```

3) Отсортированный массив записываем в файл(output.txt)

```
outfile = open('output.txt', 'w').write(' '.join(map(str, S)))
```

Входные данные (*input.txt*):

```
15
50 121 1000 69 2 -5 1 3 4 6 7 8 9 5 -47
```

Выходные данные (*output.txt)*:

```
-47 -5 1 2 3 4 5 6 7 8 9 50 69 121 1000
```

Время работы алгоритма и кол-во используемой памяти:

```
Время работы 0.0005791187286376953 секунд
Максимальное использование памяти: 5.05 КВ
```

Задание 3. Сортировка вставкой по убыванию

1) В начале нам необходимо получить данные из входного файла(input.txt)

```
file = open('input.txt', 'r')
n = int(file.readline())
S = list(map(int, file.readline().split()))
```

2) Потом проверяем введенные данные и пишем алгоритм сортировки вставками для массива *S*. Запускаем цикл *for i* и цикл *for j*, где *i, j* - индексы сравниваемых элементов массива. В результате сравнения соседних элементов меняем их местами или оставляем на своих местах. Обратите внимание, что *S[j] > S[j - 1]* для того, чтобы элементы массива сортировались по убыванию.

```
if 1 <= n <= 10**3:
    if any(abs(int(m)) > 10**9 for m in S):
        print("Введите корректные данные")
    else:
        for i in range(1, len(S)):
            for j in range(i, 0, -1):
                if S[j] > S[j - 1]:
                    S[j], S[j - 1] = S[j - 1], S[j]
                else:
                    break
else:
    print('Количество элементов должно быть больше чем 1 и меньше чем 10**3')
```

3) Отсортированный по убыванию массив записываем в файл (output.txt)

```
outfile = open('output.txt', 'w').write(' '.join(map(str, S)))
```

Входные данные (*input.txt*):

```
15
50 121 1000 69 2 -5 1 3 4 6 7 8 9 5 -47
```

Выходные данные (*output.txt)*:

```
1000 121 69 50 9 8 7 6 5 4 3 2 1 -5 -47
```

Время работы алгоритма и кол-во используемой памяти:

```
Время работы 0.00042700767517089844 секунд
Максимальное использование памяти: 5.05 КВ
```

Задание 4. Линейный поиск

1) В начале нам необходимо получить данные из входного файла(input.txt). Также создадим список index для подсчета индексов, если элемент встречается несколько раз.

```
file = open('input.txt', 'r')
A = list(map(int, file.readline().split()))
V = int(file.readline())
index = []  # список для подсчета индексов, если элемент встречается несколько раз
```

2) Далее делаем проверку полученных данных и пишем сам алгоритм поиска

```
if 0 <= len(A) <= 10**3:
    if any(abs(int(m)) > 10**3 for m in A) or abs(V) >= 10**3:
        print("Введите корректные данные")
    else:
        for i in range(0, len(A)):
            if A[i] == V:
                index.append(i + 1)
            else:
                continue
        if len(index) == 0:
            outfile = open('output.txt', 'w').write('-1 --> Число не найдено!')
        else:
            lines = [f'Число V встречается {len(index)} раз/раза\n',
                     f'Индексы числа V в последовательности: {', '.join(map(str, index))}']
            outfile = open('output.txt', 'w')
            outfile.writelines(lines)
else:
    print("Количество элементов привышает ")
```

Если полученные данные корректны, создаём цикл *for i*, где *i* - индекс элемента из массива *A*. Если элемент по индексу *i* равен искомому значению, то сохраняем его порядковый номер в список *index*, если же нет, то идём дальше. Далее проверяем количество индексов и записываем в выходной файл соответствующий текст.

Входные данные (*input.txt*):

```
5 -30 2 43 7 13 10 2 35 2
2
```

Выходные данные (*output.txt)*:

```
Число V встречается 3 раз/раза
Индексы числа V в последовательности: 3, 8, 10
```

Время работы алгоритма и кол-во используемой памяти:

```
Время работы 0.0004119873046875 секунд
Максимальное использование памяти: 5.18 КВ
```

Задание 5. Сортировка выбором

1) В начале нам необходимо получить данные из входного файла(input.txt)

```
file = open('input.txt', 'r')
n = int(file.readline())
S = list(map(int, file.readline().split()))
```

Получаем две переменные *S* - не отсортированный массив, *n* - количество элементов массива.

2) Потом проверяем введенные данные и пишем алгоритм сортировки выбором для массива *S*.

```
if 1 <= n <= 10**3:
    if any(abs(int(m)) > 10**9 for m in S):
        print("Введите корректные данные")
    else:
        for i in range(0, len(S) - 1):
            M = S[i]
            p = i
            for j in range(i+1, len(S)):
                if M > S[j]:
                    M = S[j]
                    p = j

            if p != i:
                k = S[i]
                S[i] = S[p]
                S[p] = k

else:
    print('Количество элементов должно быть больше чем 1 и меньше чем 10**3')
```

Проверив полученные данные, создаём цикл *for i* и *for j*, где *i*, *j* - индексы сравниваемых элементов. *M* - максимальный найденный элемент, который будет перезапоминаться, если будет находится элемент больше него. А переменная *p* хранит в себе индекс элемента *M*. Если значение *p* не равно значению *i*, это значит, что переменная *M* изменилась и меняем элементы массива местами.

3) Отсортированный массив записываем в файл(output.txt)

```
outfile = open('output.txt', 'w').write(' '.join(map(str, S)))
```

Входные данные (*input.txt*):

```
10
-3 5 0 -8 1 10 -50 -7 101 10050
```

Выходные данные (*output.txt)*:

```
-50 -8 -7 -3 0 1 5 10 101 10050
```

Время работы алгоритма и кол-во используемой памяти:

```
Время работы 0.0005118846893310547 секунд
Максимальное использование памяти: 5.05 КВ
```

Задание 7. Знакомство с жителями Сортлэнда

1) В начале нам необходимо получить данные из входного файла(input.txt)

```
file = open('input.txt', 'r')
n = int(file.readline())
M2 = list(map(float, file.readline().split()))# сортируемый массив
M = []# исходный неотсортированный массив
S = []# список с индексами жителей
```

Также создадим список *M* для хранения исходного массива и список *S* для хранения индексов жителей.

2) Проверяем введенные данные. Потом передаем в список *M* не отсортированный массив.

```
if (3 <= n <= 9999) and (n % 2 != 0):
    if any(int(m) > 10**6 for m in M):
        print("Элементы массива M не должны превышать значение 10**6!")
    else:
        M = M2
        for i in range(1, len(M2)):
            for j in range(i, 0, -1):
                if M2[j] < M2[j - 1]:
                    M2[j], M2[j - 1] = M2[j - 1], M2[j]
                else:
                    break
        minimum = min(M)
        medium = M[n // 2]
        maximum = max(M)
        S = [M2.index(minimum) + 1, M2.index(medium) + 1, M2.index(maximum) + 1]
else:
    print("Количество элементов n должно быть 3 <= n <= 9999, а также n - чётное!")
```

Далее для сортировки массива, я использую сортировку вставкой. Потом уже в отсортированном массиве находим минимальный, средний и максимальный элемент. А в список *S* записываю их индексы.

3) Массив с индексами жителей записываем в файл(output.txt)

```
outfile = open('output.txt', 'w').write(' '.join(map(str, S)))
```

Входные данные (*input.txt*):

```
5
10.00 8.70 0.01 5.00 3.00
```

Выходные данные (*output.txt)*:

```
1 3 5
```

Время работы алгоритма и кол-во используемой памяти:

```
Время работы 0.00066900253295898444 секунд
Максимальное использование памяти: 5.05 КВ
```

Задание 9. Сложение двоичных чисел

1) В начале получаем числа из входного файла и записываем их в список *S*. А также создаем переменную для суммы.

```
file = open('input.txt', 'r')
S = list(map(str, file.readline().split()))
summa = 0
```

2) Проверяем полученные данные. Затем переводим числа из двоичной системы счисления в десятичную и складываем их.

```
if any(len(i) > 10**3 for i in S):
    if any(len(i) < 1 for i in S):
        print("Количество бит не должно превышать значения 10**3")
else:
    for i in S:
        m = int(i, 2)
        summa += m
```

3) Записываем в файл сумму чисел, но уже в двоичной системе счисления

```
outfile = open('output.txt', 'w').write(bin(summa)[2:])
```

Входные данные (*input.txt*):

```
00001100011101 000111100101000100101010011101
```

Выходные данные (*output.txt)*:

```
1111001010001001101101110 10
```

Время работы алгоритма и кол-во используемой памяти:

```
Время работы 0.00248765945434570 3 секунд
Максимальное использование памяти: 5.05 КВ
```

Асимптотическая оценка времени работы алгоритма:

Основная сложность данного алгоритма связана с проверкой и преобразованием строк. Время выполнения данного алгоритма можно оценить как *O(n)* при условии того, что количество бит ограничено <= 10**3!

Вывод

В ходе лабораторной работы были изучены разные виды сортировок. Также были проведены тесты алгоритмов по времени и используемой памяти.