

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

Отчет по лабораторной работе №5
по курсу «Алгоритмы и структуры данных»
Тема: Деревья. Пирамида, пирамидальная сортировка.
Очередь с приоритетами.

Вариант 11

Выполнил:
Кузнецов А.Г.
К3140

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

| | |
|--------------------------------------|-----------|
| Содержание отчета | 2 |
| Задачи по варианту | 3 |
| Задача №1. Куча ли? | 3 |
| Задача №2 Высота дерева. | 4 |
| Задача №3 Обработка сетевых пакетов. | 5 |
| Задача №4 Построение пирамиды. | 8 |
| Задача №6 Очередь с приоритетами. | 10 |
| Задача №7 Снова сортировка. | 12 |
| Вывод | 16 |

Задачи по варианту

Задача №1. Куча ли?

Структуру данных «куча», или, более конкретно, «неубывающая пирамида», можно реализовать на основе массива.

Для этого должно выполняться основное свойство неубывающей пирамиды, которое заключается в том, что для каждого $1 \leq i \leq n$ выполняются условия:

1. если $2i \leq n$, то $a_i \leq a_{2i}$,
2. если $2i + 1 \leq n$, то $a_i \leq a_{2i+1}$.

Дан массив целых чисел. Определите, является ли он неубывающей пирамидой.

- Формат входного файла (input.txt). Первая строка входного файла содержит целое число n ($1 \leq n \leq 10^6$). Вторая строка содержит n целых чисел, по модулю не превосходящих $2 \cdot 10^9$.
- Формат выходного файла (output.txt). Выведите «YES», если массив является неубывающей пирамидой, и «NO» в противном случае.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

```
1 def check(n, a):
2     for i in range(1, n + 1):
3         if 2 * i <= n:
4             if not (a[i - 1] <= a[(2 * i) - 1]):
5                 return False
6         if 2 * i + 1 <= n:
7             if not (a[i - 1] <= a[2 * i]):
8                 return False
9     return True
10
11 with open('input.txt', 'r') as f:
12     n = int(f.readline())
13     a = [int(x) for x in f.readline().split(' ')]
14
15 if check(n, a):
16     with open('output.txt', 'w') as f:
17         f.write("YES")
18
19 else:
20     with open('output.txt', 'w') as f:
21         f.write("NO")
22
```

Из файла input.txt получаем количество целых чисел для проверки и сами числа. Далее с помощью функции check проверяем является ли заданная последовательность неубывающей пирамидой и записываем ответ в output.txt

| | |
|------------|-----------|
| input.txt | |
| 1 | 5 |
| 2 | 1 0 1 2 0 |
| output.txt | |
| 1 | NO |

| | |
|------------|-----------|
| input.txt | |
| 1 | 5 |
| 2 | 1 3 2 5 4 |
| output.txt | |
| 1 | YES |

Вывод по задаче: в ходе выполнения первой задачи был создан алгоритм проверки является ли последовательность неубывающей пирамидой

Задача №2 Высота дерева.

В этой задаче ваша цель - привыкнуть к деревьям. Вам нужно будет прочитать описание дерева из входных данных, реализовать структуру данных, сохранить дерево и вычислить его высоту.

- Вам дается корневое дерево. Ваша задача - вычислить и вывести его высоту. Напомним, что высота (корневого) дерева - это максимальная глубина узла или максимальное расстояние от листа до корня. Вам дано произвольное дерево, не обязательно бинарное дерево.
- Формат ввода или входного файла (input.txt). Первая строка содержит число узлов n ($1 \leq n \leq 105$). Вторая строка содержит n целых чисел от -1 до $n-1$ – указание на родительский узел. Если i -ое значение равно -1 , значит, что узел i - корневой, иначе это число является обозначением индекса родительского узла этого i -го узла ($0 \leq i \leq n - 1$). Индексы считать с 0. Гарантируется, что дан только один корневой узел, и что входные данные представляют дерево.
- Формат вывода или выходного файла (output.txt). Выведите целое число – высоту данного дерева.
- Ограничение по времени. 3 сек.
- Ограничение по памяти. 512 мб

```
1 def get_depth(ords, ind):
2     parent = ords[ind]
3     if parent == -1:
4         depth = 1
5     else:
6         depth = get_depth(ords, parent) + 1
```

```

7     return depth
8
9
10 def max_depth(ords):
11     m = get_depth(ords, 0)
12     for ind, parent in enumerate(ords):
13         depth = get_depth(ords, ind)
14         if m < depth:
15             m = depth
16     return m
17
18
19 with open('input.txt', 'r') as f:
20     n = int(f.readline())
21     arr = [int(x) for x in f.readline().split(' ')]
22
23 if (1 <= n <= 10 ** 5) and (any([0 <= x <= (n - 1) for x in arr])):
24     with open('output.txt', 'w') as f:
25         f.write(str(max_depth(arr)))

```

На вход мы получаем число узлов и указания на родительский узел из файла input.txt. Затем мы проверяем удовлетворяют ли значения условиям, если да, то запускаем функцию max_depth, которая получает информацию о дереве и выводит непосредственно высоту дерева. Ответ записываем в файл output.txt



Вывод по задаче: в ходе выполнения второй задачи был создан алгоритм для вывода высоты дерева

Задача №3 Обработка сетевых пакетов.

В этой задаче вы реализуете программу для моделирования обработки сетевых пакетов.

- Вам дается серия входящих сетевых пакетов, и ваша задача - смоделировать их обработку. Пакеты приходят в определенном порядке. Для каждого номера пакета i вы знаете время, когда пакет прибыл A_i и время, необходимое процессору для его обработки P_i (в миллисекундах). Есть только один процессор, и он обрабатывает входящие пакеты в порядке их поступления. Если процессор начал обрабатывать какой-либо пакет, он

не прерывается и не останавливается, пока не завершит обработку этого пакета, а обработка пакета i занимает ровно P_i миллисекунд. Компьютер, обрабатывающий пакеты, имеет сетевой буфер фиксированного размера S . Когда пакеты приходят, они сохраняются в буфере перед обработкой. Однако, если буфер заполнен, когда приходит пакет (есть S пакетов, которые прибыли до этого пакета, и компьютер не завершил обработку ни одного из них), он отбрасывается и не обрабатывается вообще. Если несколько пакетов поступают одновременно, они сначала все сохраняются в буфере (из-за этого некоторые из них могут быть отброшены - те, которые описаны позже во входных данных). Компьютер обрабатывает пакеты в порядке их поступления и начинает обработку следующего доступного пакета из буфера, как только заканчивает обработку предыдущего. Если в какой-то момент компьютер не занят и в буфере нет пакетов, компьютер просто ожидает прибытия следующего пакета. Обратите внимание, что пакет покидает буфер и освобождает пространство в буфере, как только компьютер заканчивает его обработку.

- Формат ввода или входного файла (input.txt). Первая строка содержит размер S буфера ($1 \leq S \leq 105$) и количество n ($1 \leq n \leq 105$) входящих сетевых пакетов. Каждая из следующих n строк содержит два числа, i -ая строка содержит время прибытия пакета A_i ($0 \leq A_i \leq 10^6$) и время его обработки P_i ($0 \leq P_i \leq 10^3$) в миллисекундах. Гарантируется, что последовательность времени прибытия входящих пакетов – неубывающая, однако, она может содержать одинаковые значения времени прибытия нескольких пакетов, в этом случае рассматривается пакет, записанный в входном файле раньше остальных, как прибывший ранее. ($A_i \leq A_{i+1}$ для $1 \leq i \leq n - 1$.)
- Формат вывода или выходного файла (output.txt). Для каждого пакета напечатайте время (в миллисекундах), когда процессор начал его обрабатывать; или -1, если пакет был отброшен. Вывести ответ нужно в том же порядке, как как пакеты были описаны во входном файле.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб

```
1 def simulation(arrival_time, process_time):
2
3     while len(finish_time) > 0 and finish_time[0] <= arrival_time:
4         finish_time.pop(0)
5
6     if len(finish_time) < buffer_size:
7         if len(finish_time) == 0:
8             finish_time.append(arrival_time + process_time)
```

```

9         with open('output.txt', 'a') as f:
10             f.write(str(arrival_time) + '\n')
11     else:
12         arrival_time_new = arrival_time
13         if finish_time[-1] > arrival_time_new:
14             arrival_time_new = finish_time[-1]
15         elif finish_time[-1] == arrival_time_new:
16             arrival_time_new = finish_time[-1] + 1
17         finish_time.append(arrival_time_new + process_time)
18         with open('output.txt', 'a') as f:
19             f.write(str(arrival_time_new)+'\n')
20     else:
21         with open('output.txt', 'a') as f:
22             f.write("-1\n")
23
24
25 with open('input.txt', 'r') as f:
26     buffer_size, count = list(map(int, f.readline().strip().split()))
27     finish_time = []
28     if (1 <= buffer_size <= 10 ** 5) and (1 <= count <= 10 ** 5):
29         for _ in range(count):
30             request = [int(x) for x in f.readline().split(' ')]
31             if (0 <= request[0] <= 10**6) and (0 <= request[1] <=
32 10**3):
33                 simulation(request[0], request[1])
34             else:
35                 with open('output.txt', 'w') as f:
36                     f.write("ERROR")
37     else:
38         with open('output.txt', 'w') as f:
39             f.write("")

```

На вход мы получаем размера буфера, количество входящих пакетов и сами пакеты, у которых мы можем взять их время прибытия и время обработки в миллисекундах, также мы создаем массив `finish_time`, который будем использовать в качестве очереди для значений внутри буфера. Мы проверяем удовлетворяют ли значения условиям, если да, то запускает функцию `simulation`, которая принимает на вход 2 значения: время прибытия пакета и время его обработки. Мы проходим по каждому пакету и записываем в `finish_time` окончательное время обработки пакета. Если там значения, которые меньше времени прибытия пакета, то чистим очередь. Ответ мы записываем в файл `output.txt`

| input | buffer_size | count | output |
|-------|-------------|-------|--------|
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 1 |
| 1 | 1 | 3 | 1 |
| 1 | 1 | 4 | 1 |
| 1 | 1 | 5 | 1 |
| 1 | 1 | 6 | 1 |
| 1 | 1 | 7 | 1 |
| 1 | 1 | 8 | 1 |
| 1 | 1 | 9 | 1 |
| 1 | 1 | 10 | 1 |
| 1 | 1 | 11 | 1 |
| 1 | 1 | 12 | 1 |
| 1 | 1 | 13 | 1 |
| 1 | 1 | 14 | 1 |
| 1 | 1 | 15 | 1 |
| 1 | 1 | 16 | 1 |
| 1 | 1 | 17 | 1 |
| 1 | 1 | 18 | 1 |
| 1 | 1 | 19 | 1 |
| 1 | 1 | 20 | 1 |
| 1 | 1 | 21 | 1 |
| 1 | 1 | 22 | 1 |
| 1 | 1 | 23 | 1 |
| 1 | 1 | 24 | 1 |
| 1 | 1 | 25 | 1 |
| 1 | 1 | 26 | 1 |
| 1 | 1 | 27 | 1 |
| 1 | 1 | 28 | 1 |
| 1 | 1 | 29 | 1 |
| 1 | 1 | 30 | 1 |
| 1 | 1 | 31 | 1 |
| 1 | 1 | 32 | 1 |
| 1 | 1 | 33 | 1 |
| 1 | 1 | 34 | 1 |
| 1 | 1 | 35 | 1 |
| 1 | 1 | 36 | 1 |
| 1 | 1 | 37 | 1 |
| 1 | 1 | 38 | 1 |
| 1 | 1 | 39 | 1 |
| 1 | 1 | 40 | 1 |
| 1 | 1 | 41 | 1 |
| 1 | 1 | 42 | 1 |
| 1 | 1 | 43 | 1 |
| 1 | 1 | 44 | 1 |
| 1 | 1 | 45 | 1 |
| 1 | 1 | 46 | 1 |
| 1 | 1 | 47 | 1 |
| 1 | 1 | 48 | 1 |
| 1 | 1 | 49 | 1 |
| 1 | 1 | 50 | 1 |
| 1 | 1 | 51 | 1 |
| 1 | 1 | 52 | 1 |
| 1 | 1 | 53 | 1 |
| 1 | 1 | 54 | 1 |
| 1 | 1 | 55 | 1 |
| 1 | 1 | 56 | 1 |
| 1 | 1 | 57 | 1 |
| 1 | 1 | 58 | 1 |
| 1 | 1 | 59 | 1 |
| 1 | 1 | 60 | 1 |
| 1 | 1 | 61 | 1 |
| 1 | 1 | 62 | 1 |
| 1 | 1 | 63 | 1 |
| 1 | 1 | 64 | 1 |
| 1 | 1 | 65 | 1 |
| 1 | 1 | 66 | 1 |
| 1 | 1 | 67 | 1 |
| 1 | 1 | 68 | 1 |
| 1 | 1 | 69 | 1 |
| 1 | 1 | 70 | 1 |
| 1 | 1 | 71 | 1 |
| 1 | 1 | 72 | 1 |
| 1 | 1 | 73 | 1 |
| 1 | 1 | 74 | 1 |
| 1 | 1 | 75 | 1 |
| 1 | 1 | 76 | 1 |
| 1 | 1 | 77 | 1 |
| 1 | 1 | 78 | 1 |
| 1 | 1 | 79 | 1 |
| 1 | 1 | 80 | 1 |
| 1 | 1 | 81 | 1 |
| 1 | 1 | 82 | 1 |
| 1 | 1 | 83 | 1 |
| 1 | 1 | 84 | 1 |
| 1 | 1 | 85 | 1 |
| 1 | 1 | 86 | 1 |
| 1 | 1 | 87 | 1 |
| 1 | 1 | 88 | 1 |
| 1 | 1 | 89 | 1 |
| 1 | 1 | 90 | 1 |
| 1 | 1 | 91 | 1 |
| 1 | 1 | 92 | 1 |
| 1 | 1 | 93 | 1 |
| 1 | 1 | 94 | 1 |
| 1 | 1 | 95 | 1 |
| 1 | 1 | 96 | 1 |
| 1 | 1 | 97 | 1 |
| 1 | 1 | 98 | 1 |
| 1 | 1 | 99 | 1 |
| 1 | 1 | 100 | 1 |

| | | | | | |
|-------------|------------|-------------|------------|-------------|------------|
| input | output.txt | input | output.txt | input | output.txt |
| 1 1 2 ✓ 1 0 | | 1 1 2 ✓ 1 0 | | 1 1 2 ✓ 1 0 | |
| 2 0 0 2 0 | | 2 0 0 2 0 | | 2 0 1 2 -1 | |
| 3 0 0 3 | | 3 0 1 3 | | 3 0 0 3 | |

| | | | | | |
|-------------|------------|-------------|------------|-------------|------------|
| input | output.txt | input | output.txt | input | output.txt |
| 1 1 2 ✓ 1 0 | | 1 1 2 ✓ 1 0 | | 1 1 2 ✓ 1 0 | |
| 2 0 1 2 -1 | | 2 0 1 2 1 | | 2 0 1 2 2 | |
| 3 0 1 3 | | 3 1 1 3 | | 3 2 1 3 | |

| | | | |
|-------------|------------|-------------|------------|
| input | output.txt | input | output.txt |
| 1 2 3 ✓ 1 0 | | 1 3 6 ✓ 1 0 | |
| 2 0 1 2 3 | | 2 0 2 2 2 | |
| 3 3 1 3 10 | | 3 1 2 3 4 | |
| 4 10 1 4 | | 4 2 2 4 6 | |
| | | 5 3 2 5 8 | |
| | | 6 4 2 6 -1 | |
| | | 7 5 2 7 | |

Вывод по задаче: в ходе выполнения третьей задачи была реализована программа для моделирования обработки сетевых пакетов с помощью очереди

Задача №4 Построение пирамиды.

В этой задаче вы преобразуете массив целых чисел в пирамиду. Это важнейший шаг алгоритма сортировки под названием HeapSort. Гарантированное время работы в худшем случае составляет $O(n \log n)$, в отличие от среднего времени работы QuickSort, равного $O(n \log n)$. QuickSort обычно используется на практике, потому что обычно он быстрее, но HeapSort используется для внешней сортировки, когда вам нужно отсортировать огромные файлы, которые не помещаются в памяти вашего компьютера.

Первым шагом алгоритма HeapSort является создание пирамиды (heap) из массива, который вы хотите отсортировать.

Ваша задача - реализовать этот первый шаг и преобразовать заданный массив целых чисел в пирамиду. Вы сделаете это, применив к массиву определенное количество перестановок (swaps). Перестановка - это операция, как вы помните, при которой элементы a_i и a_j массива меняются местами для некоторых i и j . Вам нужно будет преобразовать массив в пирамиду, используя только $O(n)$ перестановок. Обратите внимание, что в этой задаче вам нужно будет использовать min-heap вместо max-heap.

- Формат ввода или входного файла (input.txt). Первая строка содержит целое число n ($1 \leq n \leq 105$), вторая содержит n целых чисел a_i входного массива, разделенных пробелом ($0 \leq a_i \leq 109$, все a_i - различны.)

- Формат выходного файла (output.txt). Первая строка ответа должна содержать целое число m - количество сделанных свопов. Число m должно удовлетворять условию $0 \leq m \leq 4n$. Следующие m строк должны содержать по 2 числа: индексы i и j сделанной перестановки двух элементов, индексы считаются с 0. После всех перестановок в нужном порядке массив должен стать пирамидой, то есть для каждого i при $0 \leq i \leq n-1$ должны выполняться условия:

1. если $2i + 1 \leq n - 1$, то $a_i < a_{2i+1}$,

2. если $2i + 2 \leq n - 1$, то $a_i < a_{2i+2}$.

Обратите внимание, что все элементы входного массива различны. Любая последовательность свопов, которая менее $4n$ и после которой входной массив становится корректной пирамидой, считается верной.

- Ограничение по времени. 3 сек.

- Ограничение по памяти. 512 мб.

```
1 def min_heapify(a, n, i):
2     global m
3     smallest = i
4     left = 2 * i + 1
5     right = 2 * i + 2
6
7     if left < n and a[left] < a[smallest]:
8         smallest = left
9
10    if right < n and a[right] < a[smallest]:
11        smallest = right
12
13    if smallest != i:
14        m += 1
15        res.append([i, smallest])
16        a[i], a[smallest] = a[smallest], a[i]
17
18        min_heapify(a, n, smallest)
19
20
21 def min_heap(a, n):
22
23     for i in range(int(n // 2) - 1, -1, -1):
24         min_heapify(a, n, i)
25
26
27 with open('input.txt', 'r') as f:
28     n = int(f.readline())
29     a = [int(x) for x in f.readline().split(' ')]
30     m = 0
31     res = []
32
33 if (1 <= n <= 10 ** 5) and (any([(0 <= i <= 10 ** 9) for i in a])):
34     min_heap(a, n)
```

```

35
36 with open('output.txt', 'a') as f:
37     if 0 <= m <= 4*n:
38         f.write(str(m)+'\n')
39         for _ in range(m):
40             f.write(' '.join(map(str, res[_])) + '\n')
41     else:
42         f.write("ERROR")

```

На вход получаем количество чисел и сами числа из файла input.txt, а также создаем переменную m для подсчета количества сделанных свопов и массив res для хранения пар индексов элементов, которые участвовали в перестановке. Далее проверяем удовлетворяют ли значения условию, если все хорошо, то переходим к выполнению задания. Сначала высчитываем индекс родителя и 2 индекса ребенка. Рассматриваем эти значения и находим значение с наименьшим элементом. Если наименьший индекс не является индексом родителя, то применяем перестановку. По этой схеме проходимся по всему массиву и ответ записываем в output.txt



Вывод по задаче: в ходе выполнения четвертой задачи была реализован алгоритм создания пирамиды из массива

Задача №6 Очередь с приоритетами.

Реализуйте очередь с приоритетами. Ваша очередь должна поддерживать следующие операции: добавить элемент, извлечь минимальный элемент, уменьшить элемент, добавленный во время одной из операций.

- Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 106$) - число операций с очередью. 10 Следующие n строк содержат описание операций с очередью, по одному описанию в строке. Операции могут быть следующими:

- А x – требуется добавить элемент x в очередь.

– X – требуется удалить из очереди минимальный элемент и вывести его в выходной файл. Если очередь пуста, в выходной файл требуется вывести звездочку «*».

– D x y – требуется заменить значение элемента, добавленного в очередь операцией A в строке входного файла номер $x + 1$, на y. Гарантируется, что в строке $x + 1$ действительно находится операция A, что этот элемент не был ранее удален операцией X, и что y меньше, чем предыдущее значение этого элемента.

В очередь помещаются и извлекаются только целые числа, не превышающие по модулю 109.

- Формат выходного файла (output.txt). Выведите последовательно результат выполнения всех операций X, по одному в каждой строке выходного файла. Если перед очередной операцией X очередь пуста, выведите вместо числа звездочку «*».
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

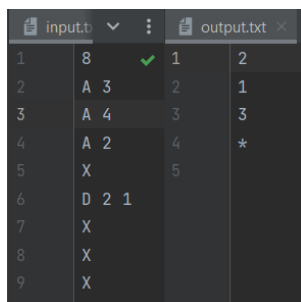
```
1 def insert(num, i):
2     queue.append([num, i])
3
4
5 def delete():
6     try:
7         min_val = 0
8         for i in range(len(queue)):
9             if queue[i][0] < queue[min_val][0]:
10                min_val = i
11            item = queue[min_val][0]
12            del queue[min_val]
13            with open('output.txt', 'a') as f:
14                f.write(str(item)+"\n")
15        except IndexError:
16            with open('output.txt', 'a') as f:
17                f.write("*+"\n")
18
19
20 def decrease_key(x, y):
21     num = 0
22     while x != queue[num][1]:
23         num += 1
24     queue[num] = [y, x]
25
26
27 with open('input.txt', 'r') as f:
28     n = int(f.readline())
29     queue = []
30     if 1 <= n <= 10**6:
31         for i in range(n):
```

```

32         a = [x for x in f.readline().strip().split(' ')]
33         if a[0] == "A":
34             if abs(int(a[1])) <= 10**9:
35                 insert(int(a[1]), i + 1)
36             else:
37                 with open('output.txt', 'w') as f:
38                     f.write("ERROR")
39         elif a[0] == "X":
40             delete()
41         else:
42             decrease_key(int(a[1]), int(a[2]))
43     else:
44         with open('output.txt', 'w') as f:
45             f.write("ERROR")

```

На вход из файла input.txt получаем количество команд и сами команды, а также создаем массив, который будем использовать для реализации очереди. Проверяем удовлетворяют ли данные условиям, если да, то выполняем задание. Функция insert добавляет в очередь элемент и его входной индекс. Функция decrease_key находит элемент с требуемым индексом и заменяет его на меньший элемент. Функция delete() находит минимальный элемент и удаляет его из очереди. В файл output.txt записываем последовательно результаты функции delete()



| input.txt | output.txt |
|-----------|------------|
| 1 8 | 1 2 |
| 2 A 3 | 2 1 |
| 3 A 4 | 3 3 |
| 4 A 2 | 4 * |
| 5 X | 5 |
| 6 D 2 1 | |
| 7 X | |
| 8 X | |
| 9 X | |

Вывод по задаче: в ходе выполнения шестой задачи была реализована очередь с приоритетами, которая поддерживает следующие операции: добавить элемент, извлечь минимальный элемент, уменьшить элемент, добавленный во время одной из операций.

Задача №7 Снова сортировка.

Напишите программу пирамидальной сортировки на Python для последовательности в убывающем порядке. Проверьте ее, создав несколько случайных массивов, подходящих под параметры:

- Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 105$) — 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 по невозрастанию массивом. Между любыми двумя числами должен стоять ровно один пробел.
- Для проверки можно выбрать случай, когда сортируется массив размера 103, 104, 105 чисел порядка 10^9 , отсортированных в обратном порядке; когда массив уже отсортирован в нужном порядке; когда много одинаковых элементов, всего 4-5 уникальных; средний - случайный. Сравните на данных сетях Randomized-QuickSort, MergeSort, HeapSort, InsertionSort.
- Есть ли случай, когда сортировка пирамидой выполнится за $O(n)$?

```

1 def heapify(a, n, i):
2     smallest = i
3     left = 2 * i + 1
4     right = 2 * i + 2
5
6     if left < n and a[left] < a[smallest]:
7         smallest = left
8
9     if right < n and a[right] < a[smallest]:
10        smallest = right
11
12    if smallest != i:
13        a[i], a[smallest] = a[smallest], a[i]
14
15        heapify(a, n, smallest)
16
17
18 def heapSort(a, n):
19     for i in range(int(n // 2) - 1, -1, -1):
20         heapify(a, n, i)
21
22     for i in range(n - 1, -1, -1):
23         a[0], a[i] = a[i], a[0]
24
25         heapify(a, i, 0)
26
27
28 with open('input.txt', 'r') as f:
29     n = int(f.readline())
30     a = [int(x) for x in f.readline().split(' ')]
31
32 with open('output.txt', 'w') as f:
33     if (1 <= n <= 10 ** 5) and (any([abs(i) <= 10 ** 9 for i in a])):
34         heapSort(a, n)
35         f.write(' '.join(map(str, a)))

```

```

36     else:
37         f.write("ERROR")

```

| Массив размера 10^3 | Наихудший случай | Средний случай | Наилучший случай | Когда много одинаковых элементов |
|--------------------------|---------------------|--------------------|---------------------|--|
| Randomized QuickSort | 0.003806 секунд | 0.004385 секунд | 0.003584 секунд | 0.005436 секунд |
| MergeSort | 0.003783 секунд | 0.004282 секунд | 0.003430 секунд | 0.003719 секунд |
| HeapSort | 0.003507 секунд | 0.004494 секунд | 0.006087 секунд | 0.003714 секунд |
| InsertionSort | 0.040452 секунд | 0.034035 секунд | 0.003169 секунд | 0.020674 секунд |

| Массив размера 10^4 | Наихудший случай | Средний случай | Наилучший случай | Когда много одинаковых элементов |
|--------------------------|---------------------|--------------------|---------------------|--|
| Randomized QuickSort | 0.023163 секунд | 0.033338 секунд | 0.021427 секунд | - |
| MergeSort | 0.031833 секунд | 0.034658 секунд | 0.0293505 секунд | 0.032051 секунд |
| HeapSort | 0.033895 секунд | 0.041860 секунд | 0.036219 секунд | 0.035351 секунд |
| InsertionSort | 3.767645 секунд | 1.985520 секунд | 0.006738 секунд | 1.951994 секунд |

| Массив размера 10^5 | Наихудший случай | Средний случай | Наилучший случай | Когда много одинаковых элементов |
|--------------------------|---------------------|--------------------|---------------------|--|
| Randomized QuickSort | 0.236108 секунд | 0.330144 секунд | 0.245145 секунд | - |
| MergeSort | 0.344491 | 0.468581 | 0.322793 | 0.363519 |

| | | | | |
|---------------|----------------------|----------------------|--------------------|----------------------|
| | секунд | секунд | секунд | секунд |
| HeapSort | 0.440605 секунд | 0.045575 секунд | 0.476666 секунд | 0.436095 секунд |
| InsertionSort | 566.777355 секунд | 374.466210 секунд | 0.071400 секунд | 227.637149 секунд |

Из файла input.txt получаем длину массива и сам массив чисел. Проверяем удовлетворяют ли условию значения, если да, то строим кучу, иными словами, переставляем значения в массиве, а дальше один за другим извлекаем элементы из массива. Отсортированный массив записываем в output.txt

Вывод по задаче: в ходе седьмой задачи был реализован метод пирамидальной сортировки в убывающем порядке

Вывод

В ходе лабораторной работы была проведена работа с пирамидами, пирамидальной сортировкой, а также с очередью и очередью с приоритетами