САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №6

по курсу «Алгоритмы и структуры данных»

Тема: Сортировка слиянием. Метод декомпозиции

Выполнил:

Смирнов Георгий Валерьевич

К3139

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[**Содержание отчета 2**](#_gjdgxs)

[**Задачи по варианту 3**](#_30j0zll)

[Задача №1. Множество 3](#_1fob9te)

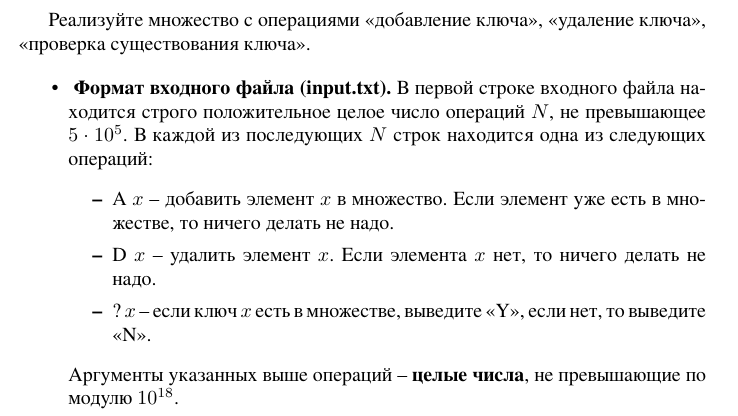
[Задача №4. Прошитый ассоциативный массив 6](#_3znysh7)

[Задача №5.](#_2et92p0) Выборы в США [12](#_3dy6vkm)

[**Вывод 23**](#_1t3h5sf)

# Задачи по варианту

## Задача №1. Множество.



**Код программы**

import time

import tracemalloc

def process\_operations():

with open('input.txt', 'r') as infile, open('output.txt', 'w') as outfile:

n = int(infile.readline().strip())

my\_set = set()

for \_ in range(n):

operation = infile.readline().strip().split()

op\_type = operation[0]

if op\_type == 'A':

x = int(operation[1])

my\_set.add(x)

elif op\_type == 'D':

x = int(operation[1])

my\_set.discard(x)

elif op\_type == '?':

x = int(operation[1])

if x in my\_set:

outfile.write('Y\n')

else:

outfile.write('N\n')

def main():

# Начало отслеживания времени и памяти

start\_time = time.perf\_counter()

tracemalloc.start()

# Обработка операций

process\_operations()

# Подсчет времени и памяти

end\_time = time.perf\_counter()

current, peak = tracemalloc.get\_traced\_memory()

tracemalloc.stop()

print(f"Время выполнения: {end\_time - start\_time:.9f} секунд")

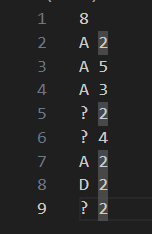
print(f"Использование памяти: {current / 10\*\*6:.6f} MB; Пиковое использование: {peak / 10\*\*6:.6f} MB")

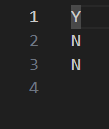
if \_\_name\_\_ == "\_\_main\_\_":

main()

Результат работы кода на примерах из текста задачи:

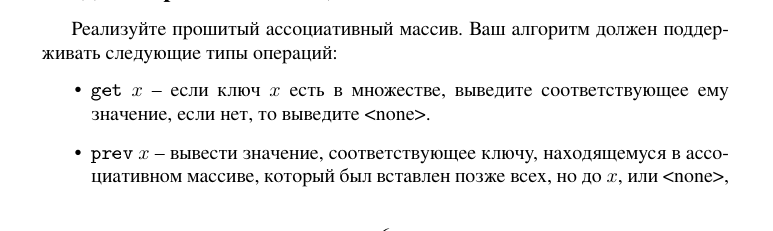
1)

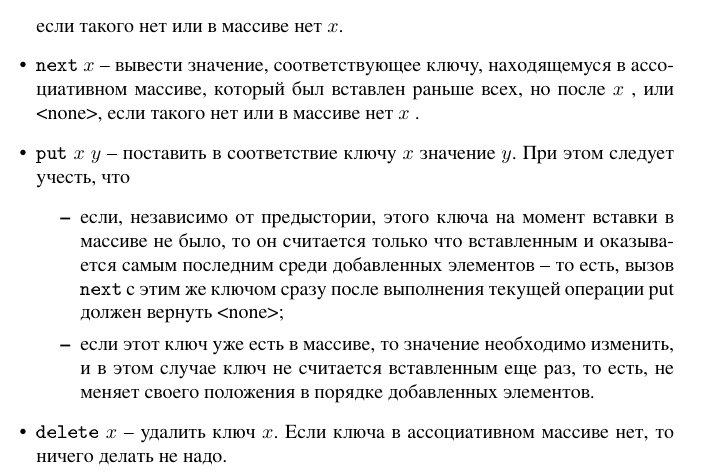




|  | время выполнения, с | использование памяти, МБ |
| --- | --- | --- |
| пример из варианта | 0.001052300 | 0.000187 |

## Задача №4. Прошитый ассоциативный ряд.





**Код программы**

class ThreadedMap:

def \_\_init\_\_(self):

# Инициализация словаря для хранения ключ-значение и списка для порядка вставки

self.map = {}

self.order = []

def put(self, x, y):

# Добавление или обновление значения

if x not in self.map:

self.order.append(x)

self.map[x] = y

def delete(self, x):

# Удаление ключа, если он существует

if x in self.map:

del self.map[x]

self.order.remove(x)

def get(self, x):

# Получение значения по ключу или 'none', если ключа нет

return self.map.get(x, 'none')

def prev(self, x):

# Получение предыдущего значения в порядке вставки

if x not in self.map:

return 'none'

index = self.order.index(x)

if index == 0:

return 'none'

return self.map[self.order[index - 1]]

def next(self, x):

# Получение следующего значения в порядке вставки

if x not in self.map:

return 'none'

index = self.order.index(x)

if index == len(self.order) - 1:

return 'none'

return self.map[self.order[index + 1]]

def process\_operations(input\_file, output\_file):

# Обработка операций из входного файла и запись результатов в выходной файл

threaded\_map = ThreadedMap()

with open(input\_file, 'r') as infile, open(output\_file, 'w') as outfile:

n = int(infile.readline().strip())

for \_ in range(n):

operation = infile.readline().strip().split()

op\_type = operation[0]

x = operation[1]

if op\_type == 'put':

y = operation[2]

threaded\_map.put(x, y)

elif op\_type == 'delete':

threaded\_map.delete(x)

elif op\_type == 'get':

result = threaded\_map.get(x)

outfile.write(f"{result}\n")

elif op\_type == 'prev':

result = threaded\_map.prev(x)

outfile.write(f"{result}\n")

elif op\_type == 'next':

result = threaded\_map.next(x)

outfile.write(f"{result}\n")

def main():

# Основная функция для запуска обработки операций

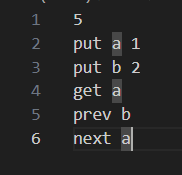
process\_operations('input.txt', 'output.txt')

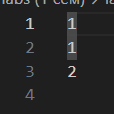
if \_\_name\_\_ == "\_\_main\_\_":

main()

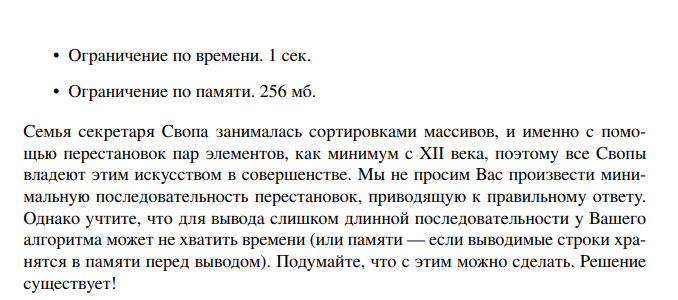
Результат работы кода на примерах из текста задачи:

1)





## Задача №5. Выборы в США.



**Код программы**

import time

import tracemalloc

def insertion\_sort\_with\_swaps(arr):

swaps = []

for i in range(len(arr)):

min\_index = i

for j in range(i + 1, len(arr)):

if arr[j] < arr[min\_index]:

min\_index = j

if min\_index != i:

arr[i], arr[min\_index] = arr[min\_index], arr[i]

swaps.append(f"Swap elements at indices {i + 1} and {min\_index + 1}.")

return swaps

def main():

# Чтение данных из файла input.txt

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

n = int(file.readline().strip())

array = list(map(int, file.readline().strip().split()))

# Начало отслеживания времени и памяти

start\_time = time.perf\_counter()

tracemalloc.start()

# Получение списка операций перестановки

swaps = insertion\_sort\_with\_swaps(array)

# Окончание отслеживания времени и памяти

current, peak = tracemalloc.get\_traced\_memory()

tracemalloc.stop()

end\_time = time.perf\_counter()

# Запись операций в файл output.txt

with open('output.txt', 'w') as file:

for swap in swaps:

file.write(swap + '\n')

file.write("No more swaps needed.\n")

# Вывод времени и памяти

print(f"Время выполнения: {end\_time - start\_time:.6f} секунд")

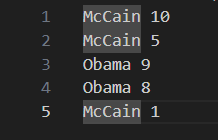
print(f"Использование памяти: {peak / 10\*\*6:.6f} МБ")

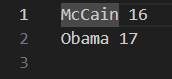
if \_\_name\_\_ == "\_\_main\_\_":

main()

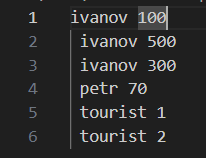
Результат работы кода на примерах из текста задачи:

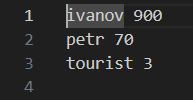
1)





2)





3)



