САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3

по курсу «Алгоритмы и структуры данных»

Тема: Быстрая сортировка, сортировки за линейное время

Вариант 20

Выполнил:

Смирнов Георгий Валерьевич

К3139

Проверила:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# 

# 

# 

# 

# 

# 

[**Задачи по варианту 3**](#_30j0zll)

[**Задача. №1 Улучшение Quick sort. 3**](#_1fob9te)

[**Задача №2 Анти-quick sort. 7**](#_3znysh7)

[**Задача №9 Ближайшие точки. 10**](#_2et92p0)

[**Вывод 13**](#_26in1rg)

# 

# 

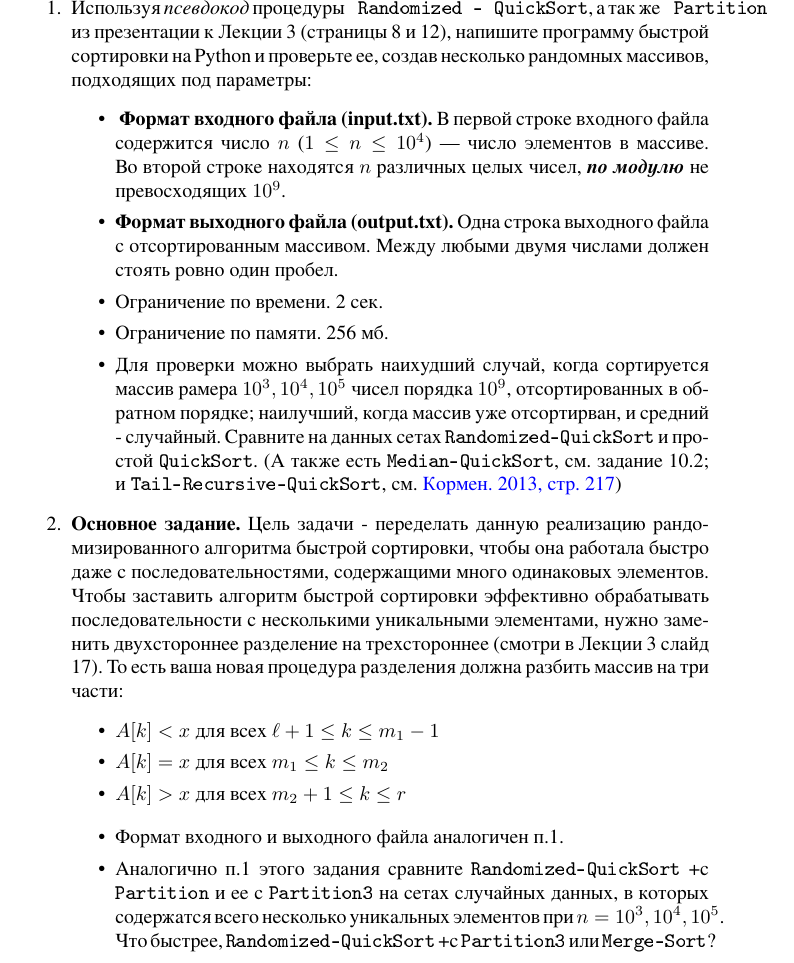
# 

# 

# 

# Задачи по варианту

## Задача. №1 Улучшение Quick sort.



**Код программы**

import random

import time

import tracemalloc

def partition(arr, low, high):

pivot = arr[high]

i = low - 1

for j in range(low, high):

if arr[j] <= pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i + 1], arr[high] = arr[high], arr[i + 1]

return i + 1

def randomized\_partition(arr, low, high):

rand\_pivot = random.randint(low, high)

arr[high], arr[rand\_pivot] = arr[rand\_pivot], arr[high]

return partition(arr, low, high)

def randomized\_quick\_sort(arr, low, high):

if low < high:

pi = randomized\_partition(arr, low, high)

randomized\_quick\_sort(arr, low, pi - 1)

randomized\_quick\_sort(arr, pi + 1, high)

def main():

# Начало отслеживания времени и памяти

start\_time = time.perf\_counter()

tracemalloc.start()

# 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()))

# Сортировка массива

randomized\_quick\_sort(array, 0, n - 1)

# Запись отсортированного массива в файл output.txt

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

file.write(' '.join(map(str, array)))

# 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")

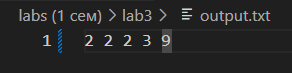
if \_\_name\_\_ == "\_\_main\_\_":

main()

Результат работы кода на примерах из текста задачи:

1)





## Задача №2 Анти-quick sort.

## 

**Код программы**

import time

import tracemalloc

def generate\_worst\_case(n):

return list(range(n, 0, -1))

def main():

# Начало отслеживания времени и памяти

start\_time = time.perf\_counter()

tracemalloc.start()

# Чтение данных из файла input.txt

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

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

# Генерация худшего случая для QuickSort

worst\_case\_array = generate\_worst\_case(n)

# Запись результата в файл output.txt

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

file.write(' '.join(map(str, worst\_case\_array)))

# 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()

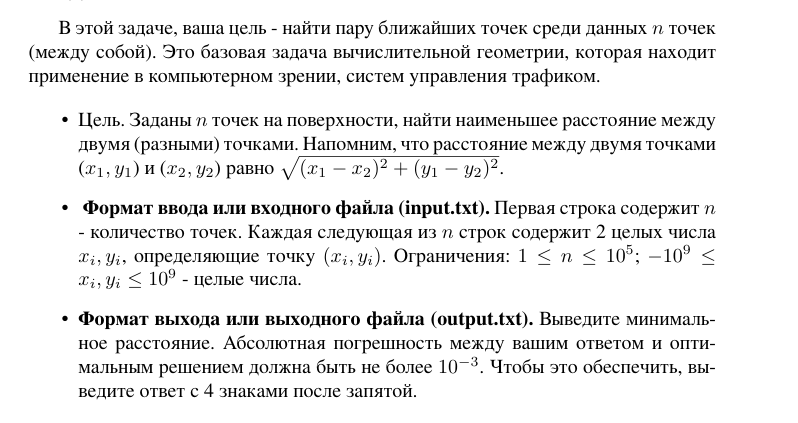
Результат работы кода на примерах из текста задачи:

1)





## Задача №9 Ближайшие точки.



**Код программы**

import math

def distance(p1, p2):

return math.sqrt((p1[0] - p2[0]) \*\* 2 + (p1[1] - p2[1]) \*\* 2)

def closest\_pair(points):

def closest\_pair\_rec(px, py):

if len(px) <= 3:

return min((distance(px[i], px[j]), (px[i], px[j]))

for i in range(len(px)) for j in range(i + 1, len(px)))

mid = len(px) // 2

Qx = px[:mid]

Rx = px[mid:]

midpoint = px[mid][0]

Qy = list(filter(lambda x: x[0] <= midpoint, py))

Ry = list(filter(lambda x: x[0] > midpoint, py))

(d1, pair1) = closest\_pair\_rec(Qx, Qy)

(d2, pair2) = closest\_pair\_rec(Rx, Ry)

d = min(d1, d2)

if d == d1:

best\_pair = pair1

else:

best\_pair = pair2

strip = [p for p in py if abs(p[0] - midpoint) < d]

for i in range(len(strip)):

for j in range(i + 1, min(i + 7, len(strip))):

p, q = strip[i], strip[j]

dst = distance(p, q)

if dst < d:

d = dst

best\_pair = (p, q)

return d, best\_pair

px = sorted(points, key=lambda x: x[0])

py = sorted(points, key=lambda x: x[1])

return closest\_pair\_rec(px, py)

def main():

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

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

points = [tuple(map(int, file.readline().strip().split())) for \_ in range(n)]

min\_distance, \_ = closest\_pair(points)

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

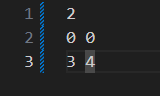
file.write(f"{min\_distance:.4f}\n")

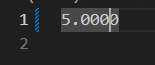
if \_\_name\_\_ == "\_\_main\_\_":

main()

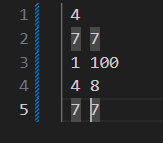
Результат работы кода на примерах из текста задачи:

1)





2)





3)

