ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра ПМиК

РАСЧЁТНО-ГРАФИЧЕСКАЯ РАБОТА  
по дисциплине

«Теория сложности вычислительных процессоров и структур»

Выполнил: студент гр. ИП-814  
Краснов И.В.

Проверил: ст. преп. кафедры ПМиК   
Разинкина Т.Э.

**Оглавление**

[**Постановка задачи** 3](#_Toc59582426)

[**Теоретическая часть** 4](#_Toc59582427)

[**Код программы** 6](#_Toc59582428)

[**Результаты тестирования** 8](#_Toc59582429)

# **Постановка задачи**

*Задача:* реализовать алгоритмы прямого и обратного полу-быстрого преобразования Фурье и протестировать их.

# **Теоретическая часть**

Как известно, в вычислительной математике большое значение имеет следующее преобразование массивов: (ƒ0, ƒ1, ƒ2… ƒn) ↔ (А0, А1, А2… Аn), которое находится с помощью алгоритма дискретного преобразования Фурье. Трудоёмкость такого метода вычисляется по формуле CN2, так как каждое из N слагаемых вычисляется за C = 5 действий. Попробуем ускорить данный алгоритм.

Рассмотрим случай, когда  *N = p1·p2*

Представим k и j в виде:  *k = k1 + p1k2 0 ≤ k ≤ N – 1*

*j = j2 + p2j1 0 ≤ j ≤ N – 1*

Здесь *k1* – остаток от деления *k* на *p1 k1 = k mod p1*

*k2* – частное от деления *k* на *p1 k1 = k div p1*

*0 ≤ k1 ≤ p1 – 1*

*0 ≤ k2 ≤ p2 – 1*

аналогично

­j­1 – частное от деления j на p2 j1 = j div p2

­j­2 – частное от деления j на p2 j2 = j mod p2

0 ≤ j1 ≤ p1 - 1

0 ≤ j2 ≤ p2 - 1

Когда число j меняется от 0 до N, то j1 и j2 пробегают независимым образом все возможные значения в своих диапазонах. Поэтому, вместо однократного суммирования можно применить двукратное:

Заметим, что:

Итак, мы будем вычислять преобразование Фурье не по формуле дискретного преобразования, а по формулам *(1)* и *(2)*:

***(1.)***

***(2)***

Оценим трудоемкость вычисления преобразования Фурье по формулам 2.3:

Массив А(0) переходит в А(1), а оттуда в А(2):

Этот переход осуществляется в 2 этапа.

В формуле *(2.3а)* имеем *р1 • р­2* коэффициентов, каждый их которых есть сумма р1 слагаемых – итого *р1 • р­2 • р1* действий. В *(2.3б)* соответственно *р1 • р­2 • р2* действий. Итого

*T(n) = p1·p2·p1 + p1·p2·p2 = p1·p2·(p1 + p2) = N(p1+p2).*

Если , то трудоемкость будет , что существенно меньше трудоемкости при обычном преобразовании Фурье. Т.е. получаем метод, позволяющий реализовать преобразование Фурье значительно быстрее. Этот метод назван *полубыстрым преобразованием Фурье.*

# **Код программы**

Программа реализована на языке Python.

import cmath

import math

def PBFT(f, n):

a = [0] \* n

a2 = [0] \* n

cs = 0

for k1 in range(p1):

for j2 in range(p2):

for j1 in range(p1):

a[k1 + p1 \* j2] += f[j2 + p2 \* j1] \* cmath.exp(-2 \* cmath.pi \* 1j \* (k1 \* j1 / p1))

cs += 1

a[k1 + p1 \* j2] \*= 1 / p1

print('Прямое преобразование')

for k1 in range(p1):

for k2 in range(p2):

for j2 in range(p2):

a2[k1 + p1 \* k2] += a[k1 + p1 \* j2] \* cmath.exp(-2 \* cmath.pi \* 1j \* (j2 / (p1 \* p2)) \* (k1 + p1 \* k2))

cs += 1

a2[k1 + p1 \* k2] \*= 1 / p2

print('({0} {1:.2f} {2} {3:.2f}i)'.format(' '[a2[k1 + p1 \* k2].real < 0], abs(a2[k1 + p1 \* k2].real), '+-'[a2[k1 + p1 \* k2].imag < 0], abs(a2[k1 + p1 \* k2].imag)))

print('C = ', cs)

return a2

def PBFT\_O(a2, n):

cs = 0

f1 = [0] \* n

a1 = [0] \* n

for k1 in range(p1):

for j2 in range(p2):

for j1 in range(p1):

f1[k1 + p1 \* j2] += a2[j2 + p2 \* j1] \* cmath.exp(2 \* cmath.pi \* 1j \* (k1 \* j1 / p1))

cs += 1

print('Обратное преобразование')

for k1 in range(p1):

for k2 in range(p2):

for j2 in range(p2):

a1[k1 + p1 \* k2] += f1[k1 + p1 \* j2] \* cmath.exp(2 \* cmath.pi \* 1j \* (j2 / (p1 \* p2)) \* (k1 + p1 \* k2))

cs += 1

print ('C = ', cs)

return a1

size = 5

p2 = int(math.sqrt(size))

while True:

if size % p2 == 0 :

break

else :

p2 += 1

p1 = size // p2

print('p1 = ', p1)

print('p2 = ', p2)

f = [0.0] \* size

f = [1, 0, 1, 0, 1]

print('ПБПФ')

A\_PB = [0] \* size

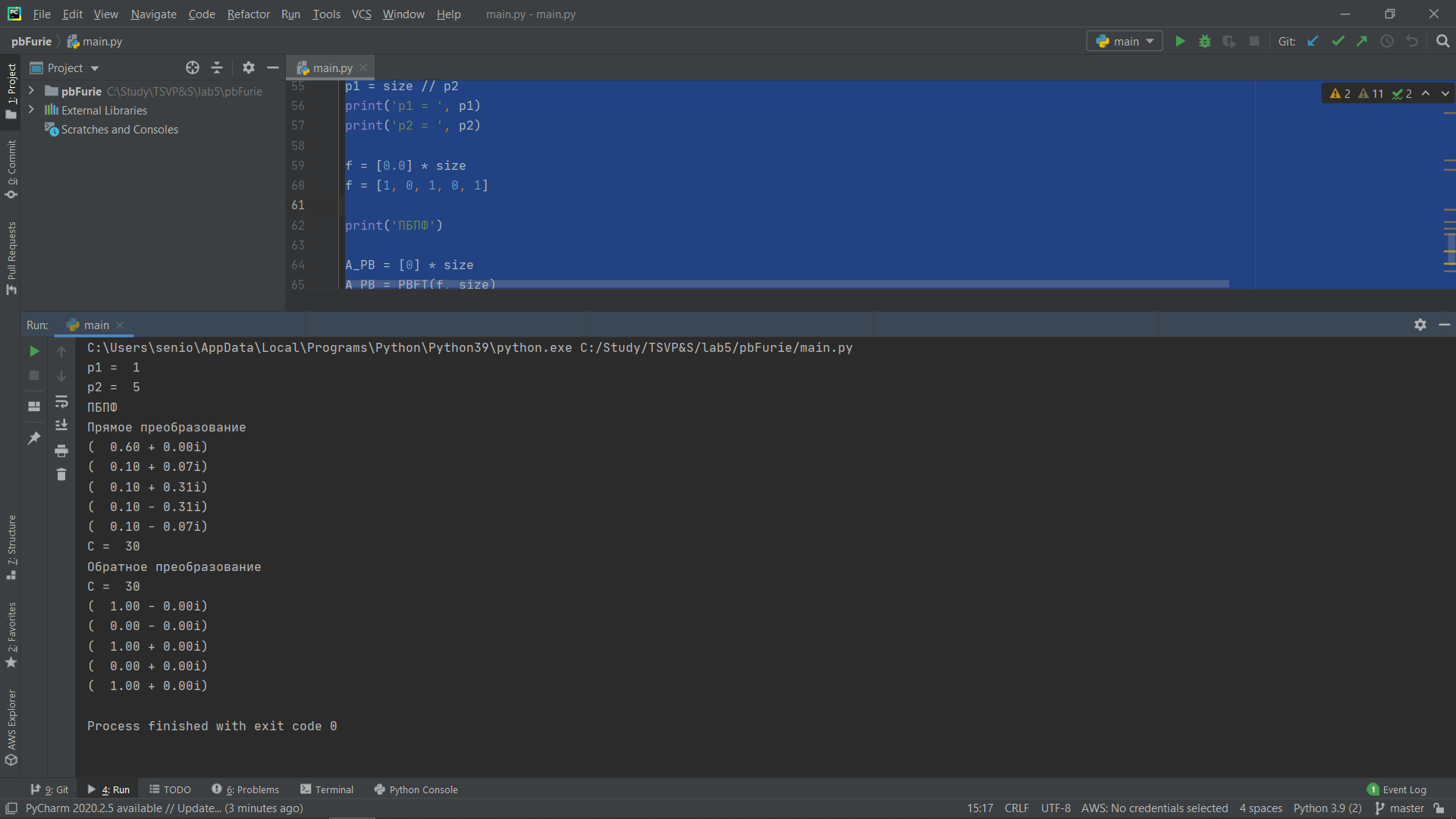
A\_PB = PBFT(f, size)

A\_PB = PBFT\_O(A\_PB, size)

for i in range(size):

print('({0} {1:.2f} {2} {3:.2f}i)'.format(' '[A\_PB[i].real < 0], abs(A\_PB[i].real), '+-'[A\_PB[i].imag < 0], abs(A\_PB[i].imag)))

# **Результаты тестирования**

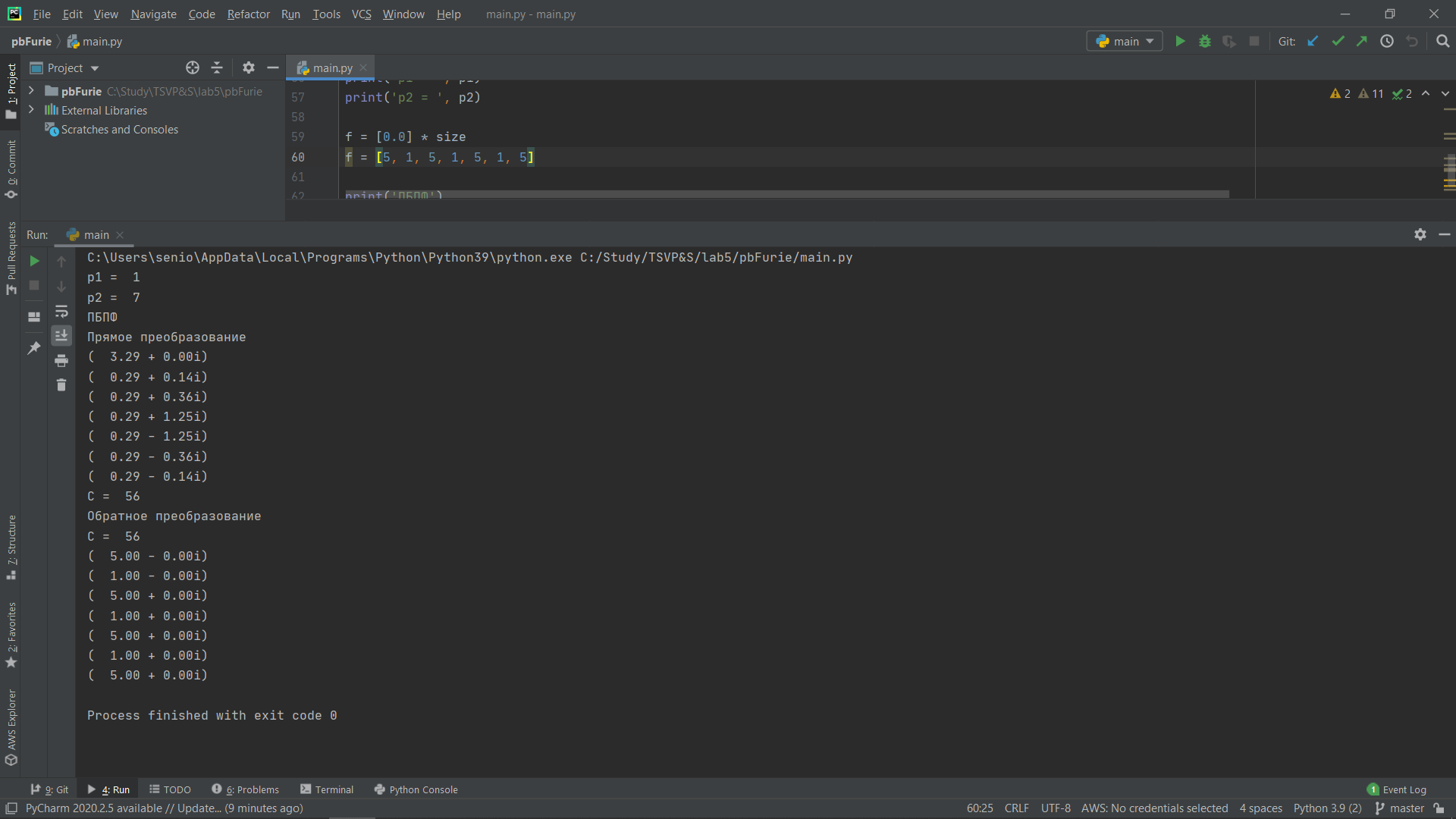


(рис. 1) Результаты первого теста

Для первого теста был выбран массив (1, 0, 1, 0, 1). Алгоритм провёл прямое и обратное преобразование. Для каждого преобразования трудоёмкость составила С = 30. Найдём трудоёмкость по формуле из теоретической части.

T(N) = N(p1 + p2) = 5 \* ( 1 + 6) = 30

Значение совпадает с тем, которое получилось после отработки алгоритма.

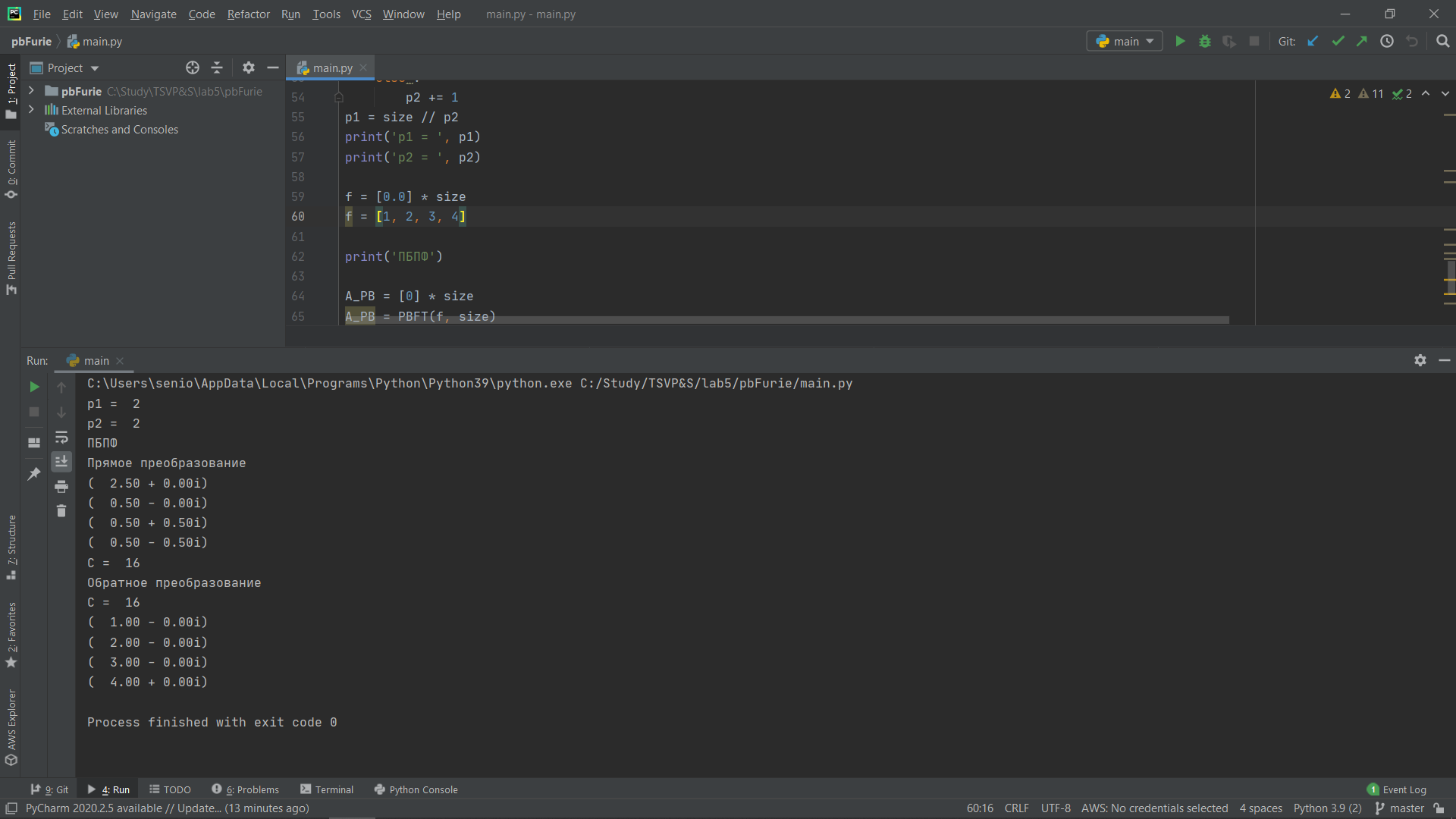


(рис. 2) Результаты второго теста

Для второго теста был выбран массив (5, 1, 5, 1, 5, 1, 5). Трудоёмкость составила С = 56.

T(N) = N(p1 + p2) = 7 \* ( 1 + 7) = 56

Практическое значение совпадает со значением, полученным в результате расчётов.



(рис. 3) Результаты третьего теста

Для последнего тестирования был выбран массив (1, 2, 3, 4). Трудоёмкость составила С = 16.

T(N) = N(p1 + p2) = 4 \* ( 2 + 2) = 16

Практическое значение совпадает со значением, полученным в результате расчётов.