QO‘YILGAN MASALANI YECHISH UCHUN TURLI XIL MURAKKABLIKDAGI ALGORITMLARDAN ENG SAMARALI ALGORITMNI ANIQLASH

Xushboqov Ismoil Urolmaxamatovich
Denov tadbirkorlik va pedagogika instituti, "Axborot texnologiyalari" kafedrasi O'qituvchisi
Pardayeva Dilfuza Najmiddin qizi
Denov tadbirkorlik va pedagogika instituti, "Axborot texnologiyalari" kafedrasi O'qituvchisi
Saidaxmedov Eldor Islomovich
Denov tadbirkorlik va pedagogika instituti, "Axborot texnologiyalari" kafedrasi O'qituvchisi
Shaydullayev Jahongir Qudrat o‘g‘li
Denov tadbirkorlik va pedagogika instituti Axborot texnologiyalari kafedrasi o'qituvchisi
Lobar Eshmirzayeva Toyir qizi
Denov tadbirkorlik va pedagogika instituti talabasi

Annotatsiya: Ushbu maqolada aniq integralni taqribiy hisoblashda foydalaniladigan trapetsiya, simpson, to‘g‘ri to‘rtburchaklar usullarining turli xil algoritm murakkabligini baholash va eng samaralisini aniqlash bayon qilingan.

Kalit so’zlar: Algoritm, tahlil, aniq integral, taqribiy hisoblash, trapetsiya usuli, simpson usuli, to‘g‘ri to‘rtburchaklar usuli.

Аннотация: В статье описана оценка сложности алгоритмов различных трапециевидных, симпсоновских и прямоугольных методов, используемых при приближенном вычислении определенного интеграла, и определение наиболее эффективного из них.

Ключевые слова: Алгоритм, анализ, точный интеграл, приближенный расчет, метод трапеций, метод Симпсона, метод прямоугольностей.

Abstract: This article describes the evaluation of the algorithm complexity of different trapezoidal, simpson, and rectangular methods used in the approximate calculation of the definite integral and the determination of the most effective one.

Key words: Algorithm, analysis, exact integral, approximate calculation, trapezoidal method, Simpson's method, rectangular method.

Turli xil algoritmlarning murakkabligini tahlil qilish va qo‘yilgan masalani yechish uchun eng samarali algoritmni topish uchun katta O yozuvi (Big-O) algoritmning murakkabligini tavsiflash uchun ishlatiladigan statistik o‘lchovlardan foydalaniladi. Buni uchun aniq integralni taqribiy hisoblash usullari - trapetsiya,

simpson, to‘rtburchak usullarini algoritm murakkabligini baholashda tadbiq etamiz.

Katta O (Big-O) yozuvi algoritmga kiritilgan ma’lumotlar va algoritmni bajarish uchun zarur bo‘lgan qadamlar o‘rtasidagi munosabatni bildiradi. U katta “O” harfi bilan belgilanadi, undan keyin ochilish va yopish qavslari keladi. Qavs ichida “n” yordamida algoritm tomonidan kiritilgan va bajarilgan qadamlar o‘rtasidagi bog‘liqlik ko‘rsatilgan.

Misol uchun, agar kirish va uning bajarilishini yakunlash uchun algoritm tomonidan qilingan qadam o‘rtasida chiziqli bog‘liqlik mavjud bo‘lsa, ishlatiladigan katta-O belgisi O(n) bo‘ladi, ya’ni n=1 bo‘lganda 1 qadam qo‘yiladi. n=10 bo‘lganda 10 ta qadam qo‘yiladi. Xuddi shunday, kvadratik funksiyalar uchun katta-O yozuvi $O(n^2)$, ya’ni n=1 bo‘lganda 1 qadam qo‘yiladi. n=10 da 100 ta qadam qo‘yiladi. n = 1 da, bu ikkalasi bir xil ishlaydi! Bu kirish va ushbu kiritishni qayta ishlash uchun qadamlar soni o‘rtasidagi bog‘liqlikni kuzatish, ba’zi bir aniq kirishlar bilan funktsiyalarni baholashdan ko‘ra yaxshiroq bo‘lishining yana bir sababidir.

Quyida eng keng tarqalgan Katta-O(Big-O)funksiyalari keltirilgan:

Doimiy(*o‘zgarmas*) – O(c) *(O(1))*
Chiziqli – O(n)
Kvadrat – $O(n^2)$
Kub – $O(n^3)$
Eksponensial – (2^n)
Logarifmik – O(log(n))
Logarifmik chiziqli – O(nlog(n))

Katta-O(Big-O) ning qanday hisoblanishi haqida tasavvurga ega bo‘lish uchun keling, doimiy, chiziqli va kvadratik murakkablikning ba’zi misollarini ko‘rib chiqaylik.

Doimiy murakkablik - O(C): Agar algoritmning bajarilishini yakunlash uchun zarur bo‘lgan qadamlar kiritilgan ma’lumotlar sonidan qat’iy nazar, doimiy bo‘lib qolsa, algoritmning murakkabligi doimiy deyiladi. Doimiy murakkablik O(c) bilan belgilanadi, bunda c har qanday doimiy son bo‘lishi mumkin.

Pythonda ro‘yxatdagi birinchi elementning kvadratini topib, keyin uni ekranda chop etadigan oddiy algoritm yozamiz:

```
def doimiy(element):
    natija = element[0] * element[0]
    print(natija)
doimiy([4, 5, 6, 8])
```

Yuqoridagi skriptda, kirish o‘lchamidan yoki kirish ro‘yxatidagi elementlar sonidan qat’iy nazar, algoritm faqat 2 qadamni bajaradi:

1. Birinchi elementning kvadratini topish

2. Natijani ekranda chop etish.

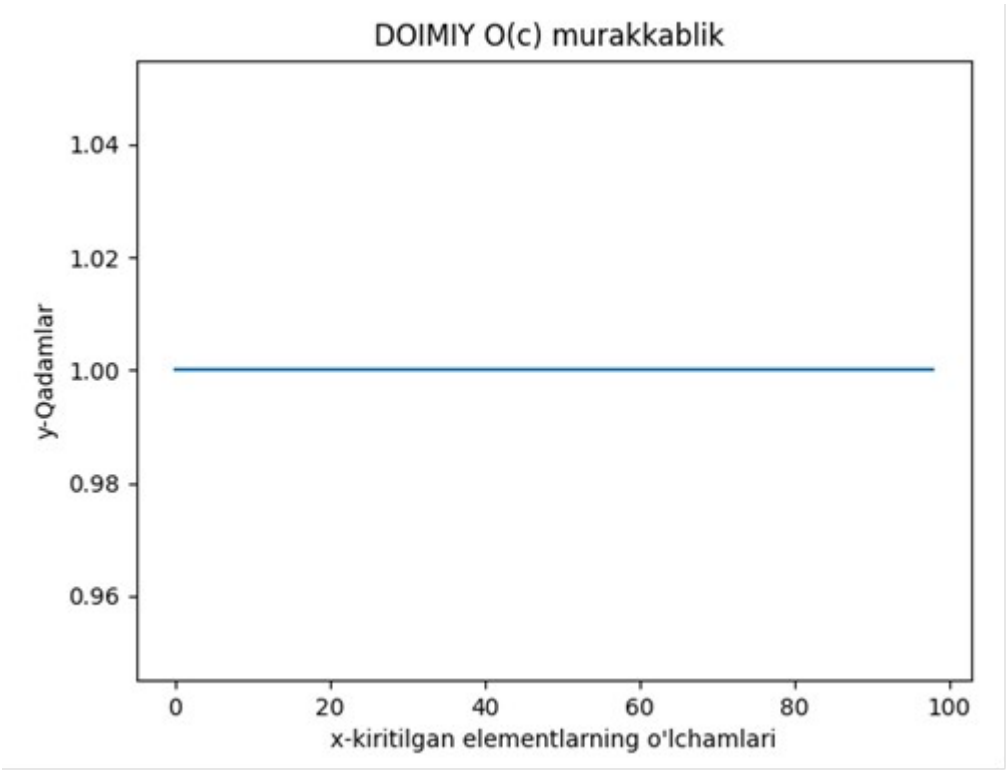
Shunday qilib, murakkablik doimiy bo‘lib qoladi.

Agar X o‘qida kiritilgan elementlarning o‘lchamlari va Y o‘qidagi qadamlar soni o‘zgaruvchan chiziqli chizma chizilsa natija to‘g‘ri chiziqdan iborat bo‘ladi.

Buni quyidagi dastur yordamida tasavvur qilishga harakat qilamiz.

Kirishlar sonidan qat’iy nazar, bajarilgan qadamlar soni bir xil bo‘lib qoladi:

```
import matplotlib.pyplot as plt
import numpy as np
qadamlar = []
def doimiy(n):
    return 1
for i in range(1, 100):
    qadamlar.append(doimiy(i))
plt.plot(qadamlar)
plt.xlabel('x-kiritilgan elementlarning o\'lchamlari')
plt.ylabel('y-Qadamlar')
plt.title('DOIMIY O(c) murakkablik ')
plt.show()
```



1-rasm. Doimiy(o‘zgarmas) murakkablikda algoritmning kiritilgan elementlarning o‘lchamlari bilan qadamlarning bog‘lanishi

Chiziqli murakkablik - O(n)

Algoritmning bajarilishini yakunlash uchun zarur bo‘lgan bosqichlar kirishlar soniga qarab chiziqli ravishda ortib yoki kamaysa, algoritmning murakkabligi chiziqli deyiladi. Chiziqli murakkablik O(n) bilan belgilanadi. Ushbu misolda ro‘yxatdagi barcha elementlarni konsolga ko‘rsatadigan oddiy dastur yozamiz:

```
def chiziqli(elementlar):
    for element in elementlar:
        print(element)
chiziqli([4, 5, 6, 8])
```

chiziqli() funksiyasining murakkabligi yuqoridagi misolda chiziqli, chunki *for*ning takrorlanish soni kirish elementlari ro‘yxatining o‘lchamiga teng bo‘ladi. Misol uchun, agar ro‘yxatda 4 ta element bo‘lsa, *for* 4-marta bajariladi.

Quyida x o‘qidagi kirishlar soni va y o‘qidagi qadamlar soni bilan chiziqli murakkablik algoritmi uchun ko‘rinishni tuzamiz:

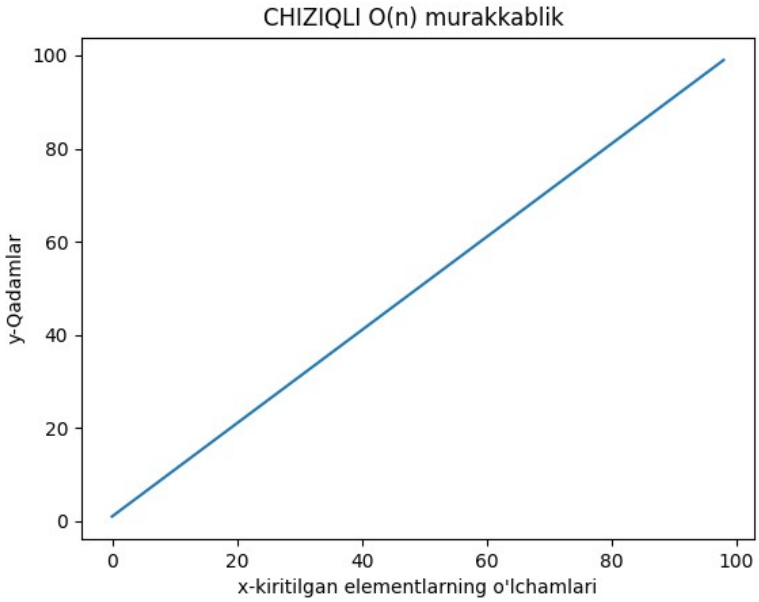
```
import matplotlib.pyplot as plt
import numpy as np
qadamlar = []
def chiziqli(elementlar):
    for element in elementlar:
        qadamlar.append(element)
chiziqli([0,1,2,3,4,5,6,7,8,9,10])
plt.plot(qadamlar)
plt.xlabel('x-kiritilgan elementlarning o\'lchamlari')
plt.ylabel('y-Qadamlar')
plt.title('CHIZIQLI O(n) murakkablik ')
plt.show()
```

Shuni ta’kidlash kerakki, katta kirishlar bilan doimiylar qiymatini yo‘qotadi. Shuning uchun biz odatda Big-O belgisidan doimiylarni olib tashlaymiz va O(2n) kabi ifoda odatda O(n) ga qisqartiriladi. O(2n) ham, O(n) ham chiziqli – aniq qiymat emas, balki chiziqli munosabat muhim.

2-rasm. Chiziqli murakkablikda algoritmning kiritilgan elementlarning o‘lchamlari bilan qadamlarning bog‘lanishi

Kvadrat murakkablik - O(n²)

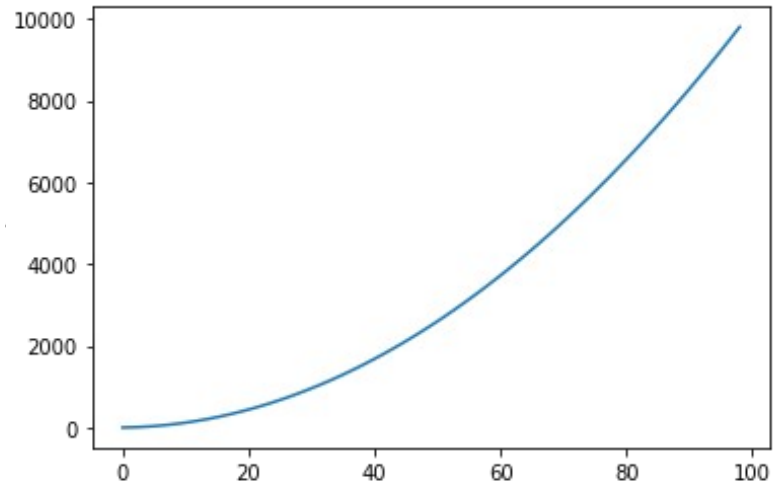
Algoritmni bajarish uchun zarur bo‘lgan bosqichlar kiritishdagi elementlar sonining kvadratik funktsiyasi bo‘lsa, algoritmning murakkabligi kvadrat deyiladi. Kvadrat murakkablik O(n²) bilan belgilanadi:




```
ruyxat=['ali','vali']
for i in ruyxat:
    for j in ruyxat:
        print(j)
```

Amalga oshirilgan qadamlarning umumiy soni n * n, bu yerda n - kirish massividagi elementlar soni.

Quyidagi grafik kvadrat



3-rasm. Kvadrat murakkablikda algoritmning kiritilgan elementlarning o‘lchamlari bilan qadamlarning bog‘lanishi

Logarifmik murakkablik - O(logn)

Ba‘zi algoritmlar logarifmik murakkablikka erishadi, masalan, Binary Search. Ikkilik qidiruv massivning o‘rtasini tekshirish va element bo‘lmagan yarmini kesish orqali massivdagi elementni qidiradi. Qolgan yarmida buni yana bajaradi va element topilgunga qadar xuddi shu amallarni davom ettiradi. Har bir bosqichda u massivdagi elementlar sonini ikki barobarga qisqartiradi. Bu massivni saralashni talab qiladi va biz ma‘lumotlar (masalan, tartiblangan) haqida taxmin qilishimiz kerak. Agar kiruvchi ma‘lumotlar haqida taxminlar qila olsangiz, algoritmning murakkabligini kamaytiradigan qadamlarni qo‘yishingiz mumkin.

Hajm murakkablik

Algoritmning bajarilishini yakunlash uchun zarur bo‘lgan qadamlar sonini hisoblaydigan vaqt murakkabligiga qo‘shimcha ravishda, siz dasturni bajarish paytida xotirada ajratishingiz kerak bo‘lgan joy miqdorini ko‘rsatadigan bo‘shliq murakkabligini ham topishingiz mumkin.

Quyidagi misolni ko‘rib chiqamiz:

```
def murakkab(n):
    ruyxat = []
    for raqam in n:
        ruyxat.append(raqam * raqam)
    return ruyxat
raqam = [2, 4, 6, 8, 10]
print(murakkab(raqam))
```

```
[4, 16, 36, 64, 100]
```

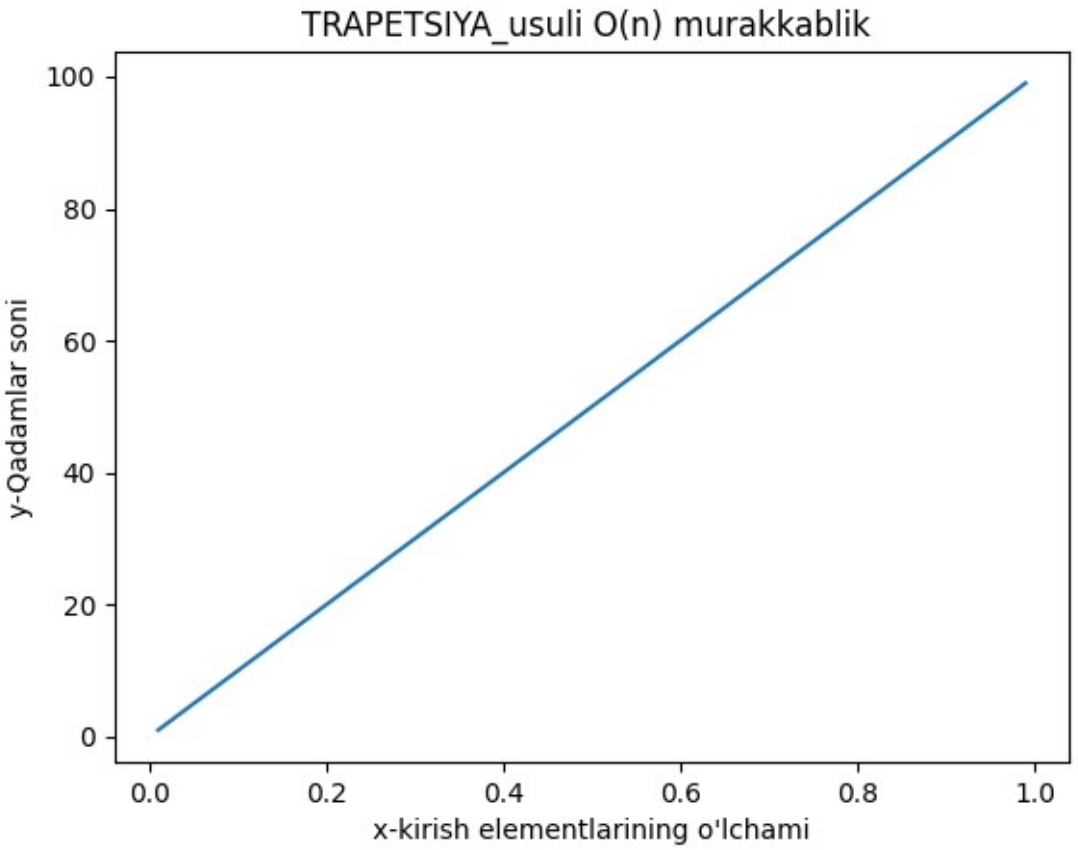
murakkablik() funksiyasi butun sonlar ro‘yxatini qabul qiladi va mos keladigan kvadratlar ro‘yxatini qaytaradi. Algoritm kirish ro‘yxatidagi kabi bir xil miqdordagi elementlar uchun xotirani ajratishi kerak. Shuning uchun algoritmning **hajm murakkabligi** O(n) ga aylanadi.

Masalaning qo‘yilishi: Yuqorida keltirilgan algoritmlarni vaqt va hajm bo‘yicha baholash usullardan foydalangan holda $\int_0^1 \frac{d(x)}{1+x^2}$ integralni taqribiy hisoblash usullari simpson, trapetsiya, to‘g‘ri to‘rtburchaklar usullarining algoritmik murakkabligi baholansin.

1-usul: Berilgan aniq integralni trapetsiya usulida hisoblash dasturini Python dasturlash tilida tuzamiz.

```
import matplotlib.pyplot as plt
import numpy as np
#1-USUL TRAPETSIYA
z=[]
s=[]
def trapetsiya_formulasi(a, b, n):
    h = (b - a) / n
    integral = 0
    for i in range(1, n):
        x = a + i * h
        z.append(x)
        s.append(i)
        integral += 1 / (1 + x**2)
    integral += (1 / (1 + a**2) + 1 / (1 + b**2)) / 2
    integral *= h
    return integral
a = 0  # Integrallash chegarasi
b = 1  # Integrallash chegarasi
n = 100  # bo‘linishlar soni
trapetsiya_formulasi(a, b, n)
plt.plot(z,s)
plt.xlabel('x-kirish elementlarining o\'lchami')
plt.ylabel('y-Qadamlar soni')
plt.title(TRAPETSIYA_usuli O(n) murakkablik ')
plt.show()
```

Natija:



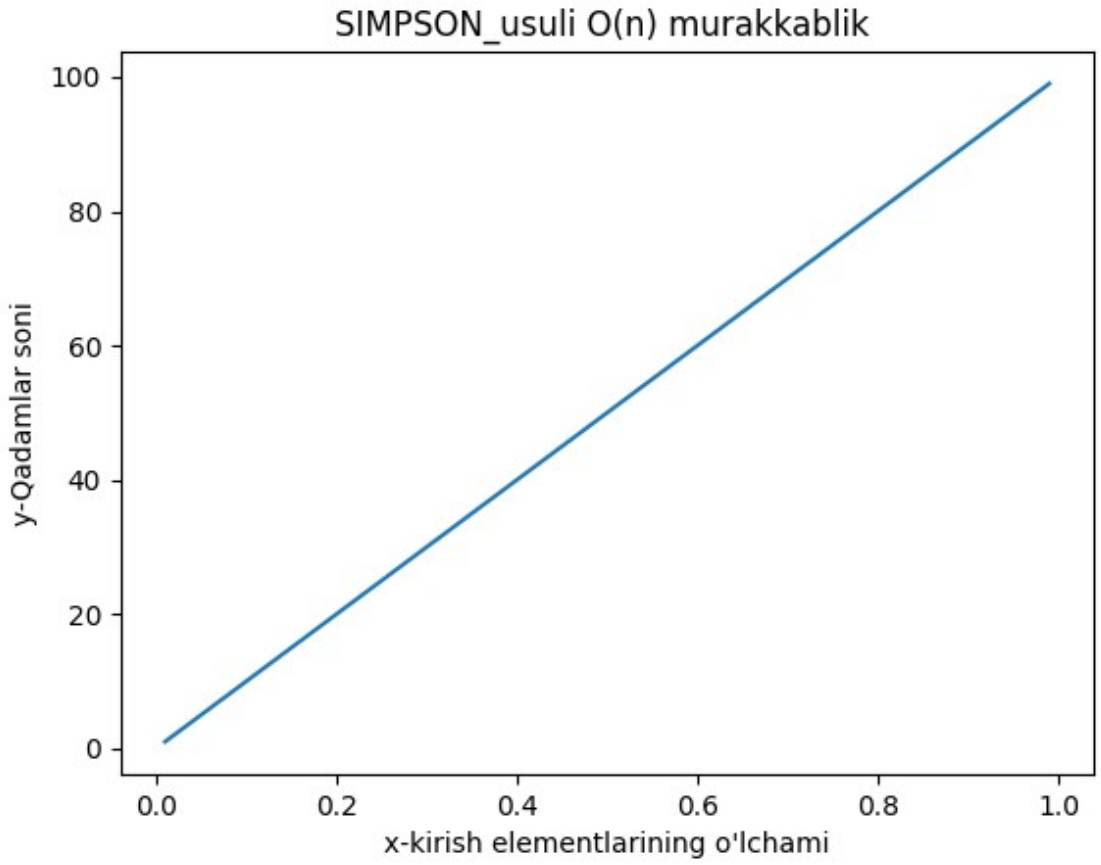
4-rasm. Trapetsiya usulida berilgan algoritmda kiritilgan elementlarning o‘lchamlari bilan qadamlarning bog‘liqligi

4-rasmdan ko‘rinib turibdiki, trapetsiya usuli **O(n)** chiziqli murakkablikga ega.

2-usul: Berilgan aniq integralni simpson usulida hisoblash dasturini Python dasturlash tilida tuzamiz.

```
a = 0  # Integrallash chegarasi
b = 1  # Integrallash chegarasi
n = 100  # bo‘linishlar soni
# 2-USUL SIMPSON
s=[]
r=[]
def simpson_integral(f, a, b, n):
    h = (b - a) / n
    x = a
    sum1 = 0
```

```
    sum2 = 0
    for i in range(1, n):
        x += h
        r.append(x)
        s.append(i)
        if i % 2 == 0:
            sum1 += f(x)
        else:
            sum2 += f(x)
    integral = (h / 3) * (f(a) + f(b) + 4 * sum1 + 2 * sum2)
    return integral
def d(x):
    return 1 / (1 + x ** 2)
simpson_integral(d, a, b, n)
plt.plot(r,s)
plt.xlabel('x-kirish elementlarining o\'lchami')
plt.ylabel('y-Qadamlar soni')
plt.title('SIMPSON_usuli O(n) murakkablik ')
plt.show()
```



Natija:

5-rasm. Simpson usulida berilgan algoritmda kiritilgan elementlarning o‘lchamlari bilan qadamlarning bog‘liqligi

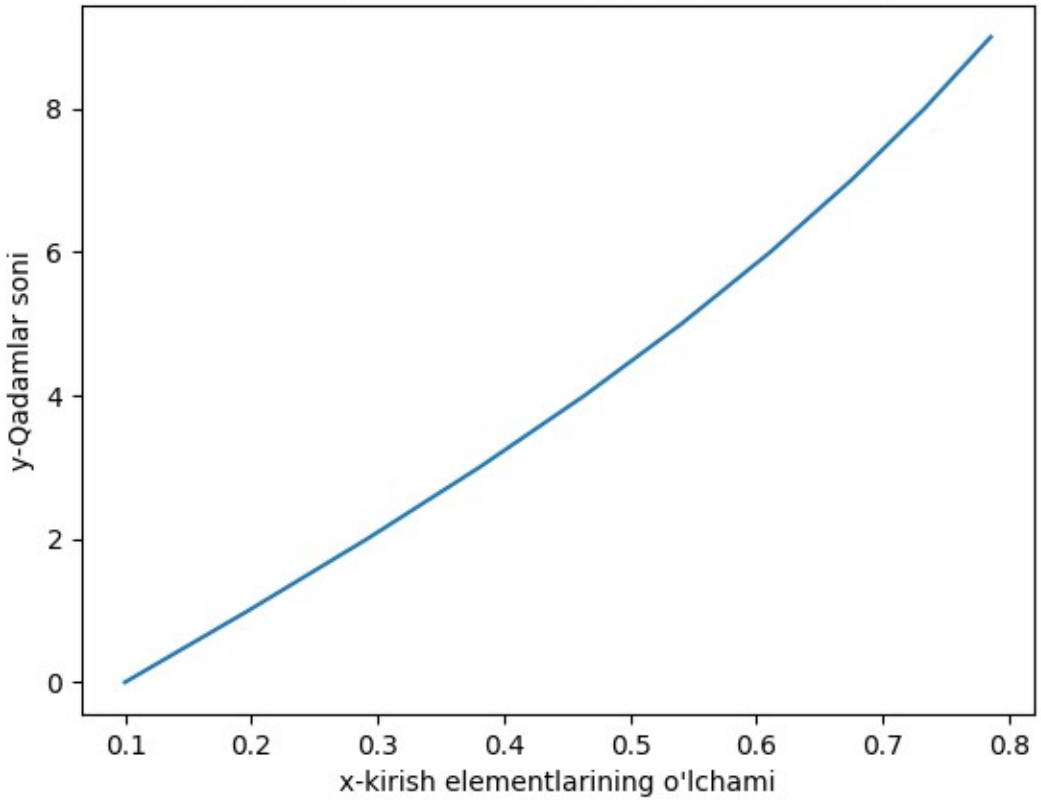
5-rasmdan ko‘rinib turibdiki, Simpson usuli **O(n)** chiziqli murakkablikga ega.

3-usul: Berilgan aniq integralni to‘g‘ri to‘rtburchaklar usulida hisoblash dasturini Python dasturlash tilida tuzamiz.

```
# 2-USUL  to‘g‘ri to‘rtburchaklar
s=[]
r=[]
def turtburchaklar_integral(f, a, b, n):
    h = (b - a) / n
    integral = 0

    for i in range(n):
        x1 = a + i * h
        x2 = a + (i + 1) * h
        integral += f((x1 + x2) / 2) * h
        s.append(i)
        r.append(integral)
    return integral
def d(x):
    return 1 / (1 + x ** 2)
a = 0  # Boshlang'ich nuqta
b = 1  # Tugatish nuqta
n = 10  # Bo‘linishlar soni
natija = turtburchaklar_integral(d, a, b, n)
print(s)
print(r)
print(natija)
plt.plot(r,s)
plt.xlabel('x-kirish elementlarining o\'lchami')
plt.ylabel('y-Qadamlar soni')
plt.title('TO\'RTBURCHAK_usuli O(n) murakkablik ')
plt.show()
```

Natija:



6-rasm. To‘g‘ri to‘rtburchaklar usulida berilgan algoritmda kiritilgan elementlarning o‘lchamlari bilan qadamlarning bog‘liqligi

6-rasmdan ko‘rinib turibdiki, to‘g‘ri to‘rtburchaklar usuli **O(n)** chiziqli murakkablikga ega.

Xulosa

Aniq integralni taqribiy hisoblashda foydalaniladigan trapetsiya, simpson, to‘g‘ri to‘rtburchaklar usullarining algoritm murakkabligini baholashda erishilgan natijalar quyidagicha: To‘g‘ri to‘rtburchaklar, tarpetsiya va simpson usullari bir xil ya’ni chiziqli O(n) murakkablikga ega. Chunki aniq integralni taqribiy hisoblashda ishlatiladigan to‘g‘ri to‘rtburchaklar, tarpetsiya va simpson usullari integralni hisoblashda yuzalarga bo‘lib hisoblaydi, demak har bir yuzani hisoblash uchun takrorlanuvchi jarayon sodir bo‘ladi. Bu o‘z navbatida har qanday dasturlash tili *for* siklga murojat qilishga to‘g‘ri keladi. For sikli n ga bog‘liq ravishda chiziqli

o‘zgarib boradi. Bu esa O(n) murakkablikni keltirib chiqaradi. Tahlil natijalariga ko’ra algoritmlarning samaradorligi bir xil ekanligi aniqlandi.

Foydalanilgan adabiyotlar ro‘yxati

1. Skiena, S. S. (1998). The algorithm design manual (Vol. 2). New York: springer.
2. Wilf, H. S. (2002). Algorithms and complexity. AK Peters/CRC Press.
3. Karimov, F. (2022). ANIQ INTEGRALNI TAQRIBIY HISOBLASH. ЦЕНТР НАУЧНЫХ ПУБЛИКАЦИЙ (buxdu. uz), 14(14).
4. G‘oyibnazarovna, X. M. (2023). ANIQ INTEGRALNING TATBIQLARI. TAQRIBIY HISOBLASH USULLARI. Journal of new century innovations, 12(5), 104-113.
5. Muhamediyeva, D., Mirzaraxmedova, A., & Xushboqov, I. (2020). PROBLEMS OF PARAMETRIC PROGRAMMING WITH S INDEPENDENT PARAMETERS. InterConf.
6. Якубов, С. Х., & Хушвоков, И. У. (2022). Вычислительный эксперимент по расчету оптимизации арок по весу. Universum: технические науки, (6-1 (99)), 33-37.