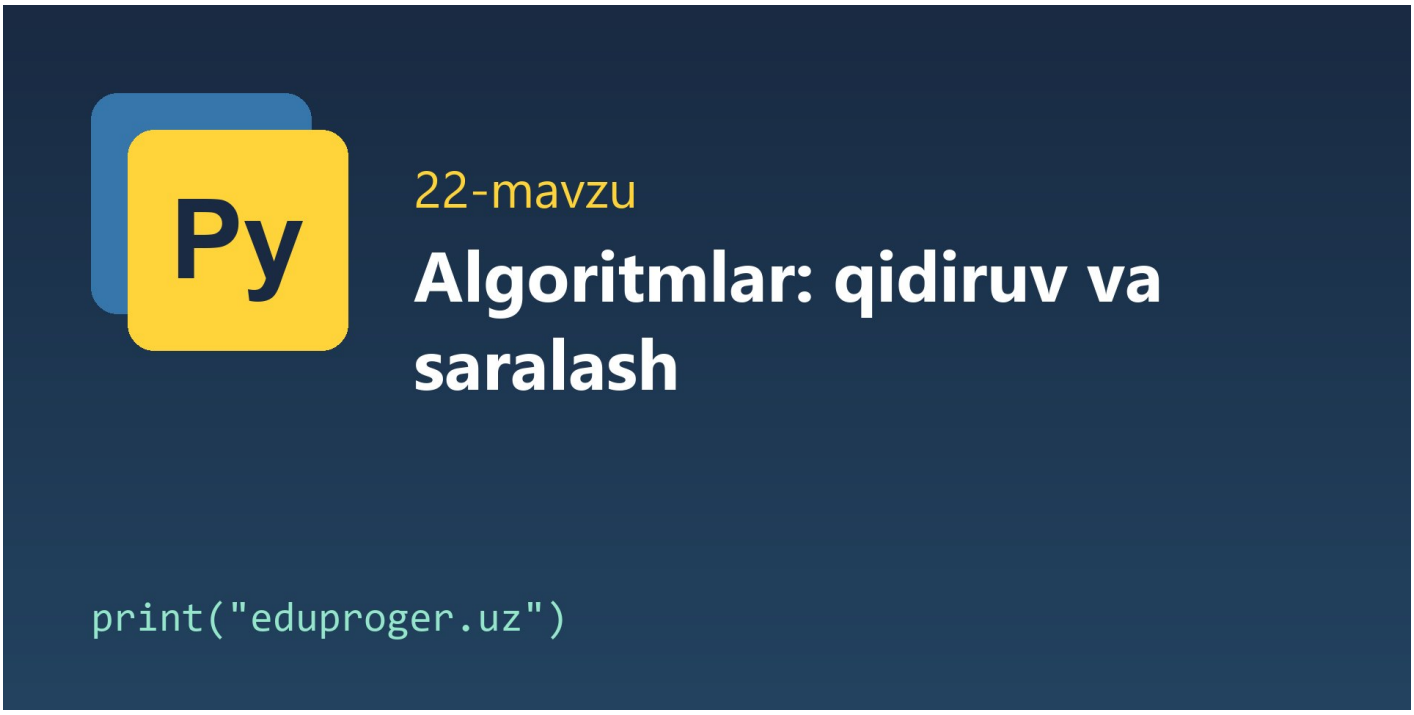
22. Algoritmlar: qidiruv va saralash



Algoritm — masalani yechish uchun aniq qadamlar ketma-ketligi. Bitta masalani turli algoritmlar bilan yechish mumkin, lekin ularning tezligi juda farq qiladi. Bu darsda eng muhim ikki vazifa — **qidiruv** va **saralash** algoritmlarini o'rganamiz. Ular olimpiada va suhbatlarda (interview) eng ko'p so'raladigan mavzu.

1. Samaradorlik: O-belgisi

Algoritm tezligi ma'lumot hajmi n ga qarab qanday o'sishi bilan baholanadi:

Belgi	Nomi	n = 1 000 000 da taxminiy qadamlar	Misol
O(1)	Doimiy	1	Ro'yxat elementini indeks bilan olish
O(log n)	Logarifmik	20	Ikkilik qidiruv
O(n)	Chiziqli	1 000 000	Chiziqli qidiruv, yig'indi
O(n log n)	—	20 000 000	Birlashtirib saralash, sorted()
O(n²)	Kvadratik	1 000 000 000 000	Pufakcha saralash

2. Chiziqli qidiruv — O(n)

Elementlarni boshidan oxirigacha birma-bir tekshiramiz. Ro'yxat saralanmagan bo'lsa ham ishlaydi:

```
def chiziqli(a, x):
    for i, v in enumerate(a):
        if v == x:
            return i
    return -1

sonlar = [42, 7, 19, 3, 88, 15]
print(chiziqli(sonlar, 88), chiziqli(sonlar, 5))
```

Natija:

```
4 -1
```

3. Ikkilik qidiruv — O(log n)

Ro'yxat **saralangan** bo'lsa, har qadamda o'rtadagi element bilan solishtirib, qidiruv sohasini ikki baravar qisqartiramiz — lug'atdan so'z qidirgandek:

```
def ikkilik(a, x):
    chap, ong = 0, len(a) - 1
    qadam = 0
    while chap <= ong:
        qadam += 1
        orta = (chap + ong) // 2
        if a[orta] == x:
            return orta, qadam
        if a[orta] < x:
            chap = orta + 1
        else:
            ong = orta - 1
    return -1, qadam

a = list(range(0, 2000000, 2))   # 1 000 000 ta juft son
print(ikkilik(a, 1234568))
print(ikkilik(a, 7))
```

Natija:

```
(617284, 17)
(-1, 20)
```

Million elementli ro'yxatda ham ~20 qadam! Pythonda tayyor vosita — `bisect` moduli:

```
import bisect
a = [3, 8, 15, 23, 42]
print(bisect.bisect_left(a, 15))
bisect.insort(a, 20)
print(a)
```

Natija:

```
2
[3, 8, 15, 20, 23, 42]
```

4. Pufakcha saralash — O(n²)

Qo'shni elementlarni solishtirib, noto'g'ri tartibda bo'lsa almashtiramiz; har o'tishda eng kattasi oxiriga «suzib chiqadi»:

```
def pufakcha(a):
    a = a[:]
    n = len(a)
    for i in range(n - 1):
        almashdi = False
        for j in range(n - 1 - i):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
                almashdi = True
        if not almashdi:      # almashish bo'lmasa — saralangan
            break
    return a

print(pufakcha([5, 1, 4, 2, 8]))
```

Natija:

```
[1, 2, 4, 5, 8]
```

5. Tanlash saralash — O(n²)

Har qadamda qolgan qismning eng kichigini topib, uni joriy o'ringa qo'yamiz:

```
def tanlash(a):
    a = a[:]
    for i in range(len(a)):
        m = i
        for j in range(i + 1, len(a)):
            if a[j] < a[m]:
                m = j
        a[i], a[m] = a[m], a[i]
    return a

print(tanlash([64, 25, 12, 22, 11]))
```

Natija:

```
[11, 12, 22, 25, 64]
```

6. Qo'yish saralash — O(n²)

Kartalarni qo'lda saralagandek: har yangi elementni chap tomondagi saralangan qismdagi o'z joyiga qo'yamiz. Deyarli saralangan ro'yxatlarda juda tez:

```
def qoyish(a):
    a = a[:]
    for i in range(1, len(a)):
        x = a[i]
        j = i - 1
        while j >= 0 and a[j] > x:
            a[j + 1] = a[j]
            j -= 1
        a[j + 1] = x
    return a

print(qoyish([12, 11, 13, 5, 6]))
```

Natija:

```
[5, 6, 11, 12, 13]
```

7. Birlashtirib saralash — O(n log n)

«Bo'l va hukmronlik qil»: ro'yxatni ikkiga bo'lamiz, har yarmini (rekursiv) saralaymiz va ikki saralangan qismni birlashtiramiz:

```
def birlashtir(a, b):
    natija, i, j = [], 0, 0
    while i < len(a) and j < len(b):
        if a[i] <= b[j]:
            natija.append(a[i]); i += 1
        else:
            natija.append(b[j]); j += 1
    return natija + a[i:] + b[j:]

def merge_sort(a):
    if len(a) <= 1:
        return a
    orta = len(a) // 2
    return birlashtir(merge_sort(a[:orta]), merge_sort(a[orta:]))

print(merge_sort([38, 27, 43, 3, 9, 82, 10]))
```

Natija:

```
[3, 9, 10, 27, 38, 43, 82]
```

8. Amaliyotda: sorted()

Haqiqiy dasturlarda o'rnatilgan `sorted()` / `.sort()` ishlatiladi — u Timsort algoritmi bilan O(n log n) da ishlaydi va barqaror (teng elementlar tartibi saqlanadi). Qo'lda yozilgan algoritmlar — tushunish va olimpiada uchun.

```
talabalar = [("Ali", 85), ("Malika", 92), ("Sardor", 85), ("Zarina", 92)]
# ball bo'yicha kamayish tartibida; teng ballilar kirish tartibida qoladi (bar
for ism, ball in sorted(talabalar, key=lambda t: t[1], reverse=True):
    print(ism, ball)
```

```
katta = list(range(1000000, 0, -1))
print(sorted(katta)[:5])
```

Natija:

```
Malika 92
Zarina 92
Ali 85
Sardor 85
[1, 2, 3, 4, 5]
```

9. Ikki ko'rsatkich usuli

Saralangan ro'yxatda yig'indisi berilgan songa teng juftlikni O(n) da topish — boshidan va oxiridan yurib:

```
def juftlik(a, s):
    i, j = 0, len(a) - 1
    while i < j:
        y = a[i] + a[j]
        if y == s:
            return a[i], a[j]
        if y < s:
            i += 1
        else:
            j -= 1
    return None

print(juftlik([1, 3, 4, 6, 8, 11], 10))
print(juftlik([1, 2, 3], 100))
```

Natija:

```
(4, 6)
None
```

10. Xulosa

- O(log n) < O(n) < O(n log n) < O(n²) — ma'lumot ko'paysa farq juda katta.
- Chiziqli qidiruv — har qanday ro'yxat; ikkilik qidiruv — faqat saralangan, O(log n).
- Pufakcha, tanlash, qo'yish — O(n²); birlashtirib saralash va sorted() — O(n log n).
- Saralangan ma'lumot bilan: bisect, ikki ko'rsatkich.