PYTHON — 3-BO‘LIM: “ALGORITMLAR (PYTHONDA)”
ALGORITMLAR (PYTHONDA)

1. Nazariy qism (to‘liq)

1. Algoritmik fikrlash Python’da

Python’da muvaffaqiyat:

- kod tezligi emas
- algoritm to‘g‘riligi va murakkabligi bilan belgilanadi.

Qoidasi:

Sekin algoritmni Python tez qila olmaydi.

2. Vaqt murakkabligi (Big-O) — muhim poydevor

Algoritm	Murakkablik
Oddiy yurish	O(n)
Saralash	O(n log n)
Ichma-ich sikl	$O(n^2)$
Binary search	O(log n)

Python’da $O(n^2)$ → faqat n kichik bo‘lsa.

3. Saralash algoritmlari (Sorting)

Python’da:

```
a.sort()
b = sorted(a)
```

- Timsort ishlatiladi
- Barqaror (`stable`)
- O(n log n)

Saralash: ko‘p masalalarda kalit qadam

4. Binary Search (ikkilik qidiruv)

Shart: massiv saralangan bo‘lishi kerak

Python’da:

```
import bisect
pos = bisect.bisect_left(a, x)
```

Qachon?

- qidirish
- minimal/ maksimal mos qiymat

5. Two Pointers texnikasi

Ikki ko‘rsatkich:

- chap (`l`)
- o‘ng (`r`)

Misol:

- juftlik yig‘indisi
- noyob bo‘lak

Murakkablikni O(n) ga tushiradi.

6. Sliding Window (ko‘chma oynacha)

Two pointers’ning maxsus ko‘rinishi.

Ishlatiladi:

- uzluksiz bo‘lak
- maksimal/minimal yig‘indi
- noyob elementlar

Har element 1 marta kiradi/chiqadi → O(n).

7. Prefix Sum (oldindan yig‘indi)

```
pref[i] = pref[i-1] + a[i]
```

Oraliq yig‘indi:

```
pref[r] - pref[l-1]
```

Ko‘p so‘rov → katta tezlik.

8. Greedy algoritmlar

G‘oya:

Har bosqichda eng yaxshi tanlov.

Misollar:

- eng ko‘p vazifa
- minimal vaqt

Eslatma:

- har doim to‘g‘ri emas
- qarshi misol tekshiriladi

9. Rekursiya

Funksiya o‘zini chaqiradi.

```
def f(n):
    if n == 0:
        return 1
    return n * f(n - 1)
```

Python’da:

- rekursiya chuqurligi cheklangan
- ehtiyot bo‘lish kerak

10. Dynamic Programming (DP)

Asosiy g‘oya: takroriy hisobni saqlash

Bosqichlar:
Holat (`dp[i]`)
O‘tish formulasi
Boshlang‘ich holat

Python’da:

- 1D yoki 2D list
- ba’zan `dict`

11. Graf algoritmlariga kirish

BFS (kenglik bo‘yicha qidiruv)

- eng qisqa yo‘l
- `deque` bilan

DFS (chuqurlik bo‘yicha qidiruv)

- komponentlar
- rekursiya yoki stack

Graf — keyingi chuqur mavzu.

12. Bit manipulyatsiya (asoslar)

```
x & 1      # toq/juft
x << 1     # 2 ga ko‘paytirish
```

Tez va xotira tejamkor.

13. Python’da optimallik

- `sys.stdin.readline`
- `sys.stdout.write`
- lokal o‘zgaruvchilar
- keraksiz obyekt yaratmaslik

Python’da optimallik — mayda, lekin muhim.

3-BO‘LIM NATIJASI

Bu nazariyani puxta o‘zlashtirgan o‘quvchi:

- Python’da algoritm tanlay oladi
- C++ algoritmlarini Python’da erkin ko‘chiradi
- olimpiada masalalariga tayyor holatga keladi

2. METODIK YORDAM

1. “Algoritm → kod” qat’iy zanjiri

Python’da eng katta xato — darrov kod yozish.

To‘g‘ri metodika:
Masalani oddiy tilda aytib berish
Algoritmni tanlash (nomi bilan)
Murakkablikni baholash
Keyin Python kodi

O‘quvchi har doim:

“Bu qaysi algoritm?”
degan savolni berishi kerak.

2. Python sekin emas — noto‘g‘ri algoritm sekin

Metodik urg‘u:

$O(n^2)$ → Python’da deyarli doim TLE

$O(n \log n)$, $O(n)$ → Python’da bemalol ishlaydi

Shu sabab: `set`, `dict`, `deque` → majburiy qurollar

3. Har algoritmni “qachon ishlaydi?” bilan berish

Misol:

- Binary search
  - faqat saralangan massivda

- Two pointers
  - monoton yurish bo‘lsa
- Sliding window
  - uzluksiz bo‘lak bo‘lsa

Algoritmni shartlari bilan yodlatish kerak.

4. Saralashni “kalit qadam” sifatida ko‘rsatish

Metodik xato:

“Masala qiyin — sort qilaylik”

To‘g‘ri yondashuv:

- sort nimani soddalashtiradi?
- sortdan keyin qaysi algoritm ishlaydi?

5. Two pointers va sliding window’ni chizma bilan o‘rgatish

Tavsiya:

- `l` va `r` ni doskada chizish
- har qadamda qaysi biri siljidi va nega

Bu algoritmlar Python’da oltin standart.

6. Prefix sum — “vaqtni sotib olish” vositasi

Metodik ketma-ketlik:

- avval har so‘rovni sikl bilan
- keyin prefix sum bilan

“1 so‘rov = 1 amal” g‘oyasi shakllanadi.

7. Greedy algoritmlarni ehtiyotkorlik bilan berish

Muhim qoida:

Greedy faqat isbot bo‘lsa ishlatiladi.

Metodik tavsiya:

- qarshi misolni oldindan ko‘rsatish
- nega ishlamaydi — tahlil qilish

8. Rekursiyani DFS bilan bog‘lash

Rekursiya:

- matematik emas

- chaqiruvlar daraxti

Metodik usul:

- kichik n bilan qo‘lda chaqirish
- stack tushunchasini tushuntirish

9. DP ni formuladan boshlash

Har doim:

- avval rekurrens
- keyin `dp` massiv
- oxirida kod

Koddan boshlash — xato.

10. Python optimallikni alohida o‘rgatish

Majburiy odatlar:

```
import sys
input = sys.stdin.readline
```

- keraksiz obyekt yaratmaslik
- `append` o‘rniga `+=` ishlatmaslik
- lokal o‘zgaruvchi ishlatish

3. TIPIK XATOLAR

1. Saralanmagan massivda binary search

```
bisect_left(a, x)   # sort yo‘q
```

Natija noto‘g‘ri bo‘ladi.

2. Two pointers’da noto‘g‘ri ko‘rsatkich siljitish

- ikkala pointerni ham yuritish
- yoki umuman siljitmaslik

Cheksiz sikl yoki noto‘g‘ri javob.

3. Sliding window’da shartni kech tekshirish

```
sum += a[r]
while sum > k:
```

`while` qachon ishlashi aniq bo‘lishi kerak.

4. Prefix sum indekslarini adashtirish

```
pref[r] - pref[l]   # XATO
```

To‘g‘risi:

```
pref[r] - pref[l-1]
```

5. Greedy’ni tekshirmasdan ishlatish

“Eng kichigini tanlaymiz” — har doim to‘g‘ri emas.

6. Rekursiyada asosiy holatni unutish

```
def f(n):
    return f(n-1)
```

Stack overflow.

7. DP jadvalini noto‘g‘ri tartibda to‘ldirish

- kerakli holat hali hisoblanmagan

DP — tartib demakdir.

8. BFS o‘rniga DFS ishlatish (eng qisqa yo‘l)

DFS eng qisqa yo‘l bermaydi.

9. Katta ma’lumotda `list` ichida qidirish

```
if x in a:   # a katta list
```

`set` kerak edi.

10. Python tezligini bahona qilish

Asl muammo:

noto‘g‘ri algoritm

METODIK XULOSA (3-BO‘LIM)

Agar o‘quvchi:

- algoritmni nomi bilan taniysa
- qachon ishlashini bilsa
- Python’ni optimallik bilan ishlatsa

u holda Python C++ dan kam emas.