PYTHON — 4-BO‘LIM

STANDART KUTUBXONALAR VA OPTIMALLIK

1. Nazariy qism (to‘liq)

1. Nega standart kutubxonalar muhim?

Olimpiadada yutqazish sabablari:

- algoritm to‘g‘ri
- lekin kod sekin
- yoki ortiqcha yozilgan

Python’ning kuchi:

tayyor, optimallashtirilgan kutubxonalarda

2. `sys` — tezlikning asosi

Tez kiritish

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

- katta `n` (10^5 – 10^6) da majburiy
- `input()` ga qaraganda ancha tez

Tez chiqarish

```
sys.stdout.write(str(x) + "\n")
```

Olimpiadada `print()` yetarli, lekin juda katta chiqishda `stdout.write` foydali.

3. `math` — matematik hisoblar

Ko‘p ishlatiladiganlari:

```
import math

math.gcd(a, b)      # EKUB
math.lcm(a, b)      # EKUK
math.sqrt(x)
math.factorial(n)
```

O‘zing sikl yozma — tayyor funksiya tezroq va ishonchli.

4. `bisect` — binary search tayyor holatda

```
import bisect

bisect.bisect_left(a, x)
```

```
bisect.bisect_right(a, x)
```

- faqat saralangan massivda
- O(log n)

Juda ko‘p masalada qidirish + joylash uchun ishlatiladi.

5. `heapq` — prioritet navbat (heap)

Min-heap (standart)

```
import heapq

h = []
heapq.heappush(h, x)
heapq.heappop(h)
```

Max-heap

```
heapq.heappush(h, -x)
```

Ishlatiladi:

- eng kichik / eng katta element
- top-K masalalar
- greedy algoritmlar

6. `collections` — kuchli modul

```
deque

from collections import deque

dq = deque()
dq.append(x)
dq.appendleft(x)
dq.popleft()
```

BFS, sliding window — oltin standart

```
Counter

from collections import Counter

cnt = Counter(a)
```

- chastota hisoblash
- yo‘q kalit → 0

`dict` + `get` o‘rniga eng qulay variant

```
defaultdict
```

```
from collections import defaultdict

d = defaultdict(int)
```

Murakkab guruhlashlarda qulay,
lekin ehtiyotkorlik bilan ishlatiladi.

7. `itertools` — kombinatorika va generatorlar

Eng foydalilari:

```
from itertools import permutations, combinations

permutations(a, 2)
combinations(a, 3)
```

Kichik `n` uchun,
katta `n` da ehtiyot bo‘lish kerak (`n!` xavfi).

8. Generatorlar — xotira tejash

Oddiy list:

```
a = [i*i for i in range(n)]
```

Generator:

```
a = (i*i for i in range(n))
```

Generator:

- kam xotira
- lekin faqat 1 marta yuriladi

9. Python’da optimallikning 10 ta oltin qoidasi

To‘g‘ri algoritm tanlang
`set`, `dict` ishlating
`list` ichida qidirmang
`sys.stdin.readline`
Keraksiz obyekt yaratmang
Lokal o‘zgaruvchilarni ishlating
Ichma-ich sikldan qoching
Tayyor kutubxonani tanlang
Keraksiz rekursiyadan qoching
O‘qiladigan kod yozing

10. Python qayerda yutadi, qayerda yutqazadi?

Kuchli joylari:

- string
- dict / set
- tez yozish
- murakkab mantiq

Zaif joylari:

- juda katta n^2
- chuqur rekursiya
- juda ko‘p matematik hisob

Shu joylarda algoritm tanlash hal qiluvchi.

4-BO‘LIM NATIJASI

Bu nazariyani puxta bilgan o‘quvchi:

- Python’ni tez ishlaydigan qurolga aylantiradi
- C++ bilan raqobat qila oladi
- olimpiadada TLE dan qochadi

2. METODIK YORDAM (kengaytirilgan)

1. “Algoritm bor — lekin vaqt yetmadi” sindromini yo‘qotish

Ko‘p o‘quvchi:

- to‘g‘ri algoritm tanlaydi
- lekin Python’da sekkin ishlatadi

Metodik yechim: har algoritm bilan birga qaysi modul ishlatiladi deb o‘rgatish

Masalan:

- BFS → `deque`
- Binary search → `bisect`
- Top-K → `heapq`

2. `sys.stdin.readline` — majburiy refleks

Metodik qoida:

Agar $n \geq 10^5$ bo‘lsa — `input()` taqiqlanadi.

Amaliy mashq:

- bitta masalani `input()` bilan
- bitta masalani `readline()` bilan
  va vaqtni solishtirish

3. `math` funksiyalarini “qayta ixtiro qilmaslik”

Ko‘p xato:

```
while b:
    a, b = b, a % b
```

To‘g‘risi:

```
math.gcd(a, b)
```

Tayyor funksiya:

- tezroq
- kam xato
- aniq

4. `bisect` + `sort` = standart kombinatsiya

Metodik formula:

Qidirish bo‘lsa → sort → bisect

Masala o‘qitishda:

- qo‘lda binary search yozdirish
- keyin `bisect` bilan almashtirish

Pythonda fikrlash shu yerda shakllanadi.

5. `heapq` ni “faqat min-heap” deb qabul qilish

Metodik urg‘u: max-heap = manfiy qiymatlar

Mashq:

- top-3 eng katta
- top-5 eng kichik

6. `deque` ni BFS bilan bog‘lab berish

Metodik yondashuv:

- `list.pop(0)` → sekin
- `deque.popleft()` → tez

Bu farqni real vaqt bilan ko‘rsatish kerak.

7. `Counter` va `defaultdict`ni solishtirish

Metodik farqlash:

- `Counter` → tayyor chastota

- `defaultdict` → murakkab guruhlash

Qachon qaysi biri — alohida jadval bilan beriladi.

8. Generatorni “xotira vositasi” sifatida ko‘rsatish

Metodik misol:

- `list(range(10**7))` → xotira xato
- `range(10**7)` → muammo yo‘q

Generator = xotira yutug‘i.

9. Python optimallik check-list (imtihonda)

Har masalada o‘quvchi o‘ziga savol bersin:

- input tezmi?
- qidirish `set` dami?
- `for` ichida `len()` bormi?
- keraksiz `append` yo‘qmi?

10. “Toza kod” ham optimallik

Metodik urg‘u:

- murakkab kod → ko‘p xato
- soddaroq kod → tezroq tuzatish

3. TIPIK XATOLAR

1. Katta kirishda `input()` ishlatish

Eng ko‘p TLE sababi.

2. `list.pop(0)` bilan BFS yozish

```
q.pop(0)   # O(n)
```

3. Binary search’dan oldin saralamaslik

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

4. `heapq` ni saralash deb o‘ylash

Heap:

- faqat minimalni tez beradi
- to‘liq tartib emas

5. `Counter` ni `dict` deb o‘ylash

```
cnt[x] += 1  # Counter’da mumkin, dict’da yo‘q
```

6. Generatorni 2 marta ishlatish

```
g = (i for i in a)
sum(g)
sum(g)   # 0
```

7. Keraksiz obyekt yaratish

```
for i in range(n):
    x = int(input())
```

O‘rniga batch o‘qish mumkin.

8. `defaultdict` ni nazoratsiz ishlatish: keraksiz kalitlar paydo bo‘ladi

9. Rekursiya + katta n

Stack overflow xavfi.

10. Python sekin degan noto‘g‘ri xulosa

Asl sabab:

noto‘g‘ri algoritm + noto‘g‘ri modul

METODIK XULOSA (4-BO‘LIM)

Agar o‘quvchi:

- to‘g‘ri algoritm tanlasa
- to‘g‘ri modul ishlatsa
- tez kiritish/chiqarishni unutmasa

Python olimpiadada kuchli qurolga aylanadi.