পাইথনে স্পেস কমপ্লেক্সিটি (Space Complexity) কী?

সহজ কথায়, **স্পেস কমপ্লেক্সিটি** হলো একটা প্রোগ্রাম বা অ্যালগরিদম চালাতে তোমার কম্পিউটারের মেমরি (RAM) কতটা জায়গা নেবে তা মাপার একটা উপায়। এটা টাইম কমপ্লেক্সিটির মতোই, কিন্তু সময়ের বদলে স্পেস (মেমরি) নিয়ে কথা বলে। প্রোগ্রামটা যত বড় ইনপুট নেবে (যেমন, n-টা সংখ্যা), ততই স্পেসের চাহিদা কীভাবে বাড়বে বা থাকবে তা দেখা হয়।

পাইথনে এটা বিশেষভাবে গুরুত্বপূর্ণ কারণ পাইথন অটোমেটিক মেমরি ম্যানেজ করে (গার্বেজ কালেকশন দিয়ে), কিন্তু বড় ডেটা সেটে স্পেস অপটিমাইজ করা দরকার। আমরা এটা **Big O নোটেশন** দিয়ে মাপি, যেমন:

- **O(1)**: কনস্ট্যান্ট স্পেস – ইনপুট যত বড় হোক, স্পেস একই থাকে।
- **O(n)**: লিনিয়ার স্পেস – ইনপুটের সাইজের সাথে স্পেস লিনিয়ারভাবে বাড়ে।
- **$O(n^2)$**: কোয়াড্র্যাটিক স্পেস – n-টা এলিমেন্টের জন্য n×n ম্যাট্রিক্সের মতো।
- **O(log n)**: লগারিদমিক স্পেস – বাইনারি সার্চের মতো।

স্পেস কমপ্লেক্সিটি দুই ভাগে ভাগ করা যায়:

- **ফিক্সড স্পেস (Fixed Space)**: প্রোগ্রামের কনস্ট্যান্ট ভেরিয়েবল, যা ইনপুট সাইজের উপর নির্ভর করে না।
- **ভেরিয়েবল স্পেস (Variable Space)**: লিস্ট, অ্যারে, স্ট্যাক ইত্যাদি যা ইনপুটের সাথে বাড়ে।

এছাড়া, **অক্সিলিয়ারি স্পেস (Auxiliary Space)** বলে একটা জিনিস আছে – এটা ইনপুট আর আউটপুট ছাড়া অতিরিক্ত স্পেস, যা প্রোগ্রাম চালাতে ব্যবহার হয় (যেমন, টেম্পোরারি লিস্ট)। সাধারণত স্পেস কমপ্লেক্সিটি অক্সিলিয়ারি স্পেসকেই বোঝায়।

কীভাবে স্পেস কমপ্লেক্সিটি বের করতে হয়? নিয়মগুলো

স্পেস কমপ্লেক্সিটি বের করার নিয়মগুলো সহজ:

1. **ইনপুট সাইজকে n ধরো**: n হলো ইনপুটের সাইজ (যেমন, লিস্টের লেন্থ)।
2. **প্রোগ্রামে কোন কোন ভেরিয়েবল স্পেস নেবে তা দেখো**:
    - প্রিমিটিভ টাইপ (int, float, bool): সাধারণত কনস্ট্যান্ট স্পেস, O(1)। পাইথনে একটা int 28-32 বাইট নেয় (ডিপেন্ড করে সিস্টেমে)।
    - লিস্ট, ডিকশনারি, সেট: O(n), কারণ n-টা এলিমেন্ট ধরে।
    - স্ট্রিং: O(n), যদি n-টা ক্যারেক্টার হয়।
    - রিকার্সিভ ফাংশন: প্রতি কল-এ স্ট্যাক ফ্রেম নেয়, তাই O(n) যদি n-লেভেল রিকার্সন হয়।
3. **সবচেয়ে খারাপ কেস দেখো**: ওয়ার্স্ট-কেস সিনারিওতে কত স্পেস লাগবে।
4. **কনস্ট্যান্ট ফ্যাক্টর ইগনোর করো**: Big O-তে 2n-কে O(n) বলা হয়, কনস্ট্যান্ট ড্রপ করো।
5. **ইনপুট/আউটপুটকে গণনা করো না যদি অক্সিলিয়ারি স্পেস দেখো**: কিন্তু টোটাল স্পেসে সবকিছু যোগ করো।
6. **পাইথনে স্পেস চেক করার টুল**: `sys.getsizeof()` দিয়ে কোনো অবজেক্টের সাইজ বাইটে দেখা যায়। কিন্তু এটা শুধু অবজেক্টের নিজস্ব সাইজ দেয়, চাইল্ড অবজেক্ট (যেমন লিস্টের এলিমেন্ট) আলাদা যোগ করতে হয়।

এখন উদাহরণ দিয়ে ক্লিয়ার করি। আমি কয়েকটা পাইথন কোডের উদাহরণ দেব, এবং তাদের স্পেস কমপ্লেক্সিটি ব্যাখ্যা করব। আমি কোড চালিয়ে আসল মেমরি সাইজও দেখিয়েছি (পাইথন 3.12-এ)।

উদাহরণ 1: কনস্ট্যান্ট স্পেস – O(1)

এখানে একটা ফাংশন যা দুটো সংখ্যা যোগ করে। ইনপুট যত বড় হোক (যেমন, একটা বড় লিস্ট), কিন্তু ফাংশনটা শুধু দুটো ভেরিয়েবল ব্যবহার করে, তাই স্পেস একই থাকে।

python Collapse Wrap Run Copy

```python
import sys  # মেমরি চেকের জন্য

def constant_space(a, b):
    result = a + b  # শুধু কয়েকটা int ভেরিয়েবল
    return result

# মেমরি চেক
print(sys.getsizeof(5))  # একটা int-এর সাইজ: 28 bytes (আমার সিস্টেমে)
print(sys.getsizeof(10))  # একই: 28 bytes
```

ব্যাখ্যা:

- নিয়ম অনুসারে: কোনো লিস্ট বা অ্যারে নেই যা n-এর সাথে বাড়বে। শুধু ফিক্সড ভেরিয়েবল (a, b, result)।
- স্পেস কমপ্লেক্সিটি: O(1) – কনস্ট্যান্ট।
- আসল মেমরি: প্রতি int 28 bytes, টোটাল ~84 bytes (কিন্তু Big O-তে এটা O(1))।
- যদি ইনপুট একটা n-এলিমেন্ট লিস্ট হয়, কিন্তু ফাংশনটা লিস্ট ব্যবহার না করে, তাহলে অক্সিলিয়ারি স্পেস O(1)।

উদাহরণ 2: লিনিয়ার স্পেস – O(n)

এখানে একটা ফাংশন যা n-টা সংখ্যার লিস্ট বানায়। ইনপুট সাইজ n-এর সাথে স্পেস বাড়বে।

python Collapse Wrap Run Copy

```python
import sys

def linear_space(n):
    lst = list(range(n))  # n-টা এলিমেন্টের লিস্ট
    return lst

n = 10
lst = linear_space(n)

# মেমরি চেক
print(sys.getsizeof(lst))  # লিস্টের নিজস্ব সাইজ: 136 bytes (আমার সিস্টেমে)
total_size = sys.getsizeof(lst) + sum(sys.getsizeof(x) for x in lst)
print(total_size)  # টোটাল: 416 bytes (লিস্ট + এলিমেন্টস)
```

ব্যাখ্যা:

- নিয়ম অনুসারে: lst ভেরিয়েবল n-টা এলিমেন্ট ধরে, প্রতি এলিমেন্ট (int) ~28 bytes, প্লাস লিস্টের ওভারহেড (~136 bytes ফাঁকা লিস্টের জন্য, কিন্তু বাড়তে পারে)।
- স্পেস কমপ্লেক্সিটি: O(n) – n-এর সাথে লিনিয়ারভাবে বাড়ে।
- আসল মেমরি: n=10-এ 416 bytes। যদি n=100 হয়, তাহলে ~136 + (100*28) = ~2936 bytes – দেখা যায় বাড়ছে।
- অক্সিলিয়ারি স্পেস: O(n), কারণ লিস্টটা অতিরিক্ত।

উদাহরণ 3: কোয়াড্র্যাটিক স্পেস – $O(n^2)$

একটা n×n ম্যাট্রিক্স বানানো।

python Collapse Wrap Run Copy

```python
def quadratic_space(n):
    matrix = [[0 for _ in range(n)] for _ in range(n)]  # n*n এলিমেন্ট
    return matrix

# মেমরি চেক (অনুমান): প্রতি রো O(n), টোটাল O(n²)
```

ব্যাখ্যা:

- নিয়ম: প্রতি রো-তে n-টা এলিমেন্ট, টোটাল n রো = n^2 এলিমেন্ট।
- স্পেস: $O(n^2)$।
- উদাহরণ: n=3-এ 9 এলিমেন্ট, প্রতি int 28 bytes = ~252 bytes + ওভারহেড। বড় n-এ অনেক স্পেস নেবে।

উদাহরণ 4: রিকার্সিভ ফাংশন – O(n) স্ট্যাক স্পেস

ফ্যাক্টোরিয়াল রিকার্সন দিয়ে।

python Collapse Wrap Run Copy

```python
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)  # প্রতি কল-এ স্ট্যাক ফ্রেম
```

ব্যাখ্যা:

- নিয়ম: প্রতি রিকার্সিভ কল-এ একটা স্ট্যাক ফ্রেম (ভেরিয়েবল + রিটার্ন অ্যাড্রেস) নেয়, টোটাল n লেভেল = O(n)।
- স্পেস: O(n)। পাইথনে ডিফল্ট রিকার্সন লিমিট 1000, তারপর স্ট্যাক ওভারফ্লো।
- অক্সিলিয়ারি: O(n), কোনো লিস্ট নেই কিন্তু স্ট্যাক স্পেস।

টিপস এবং সতর্কতা

- **স্পেস অপটিমাইজ করো**: লিস্টের বদলে জেনারেটর ব্যবহার করো (yield) – এতে O(1) স্পেস হয় কারণ একসাথে সব ধরে না।
- **পাইথন-স্পেসিফিক**: লিস্ট ডায়নামিক, তাই ওভার-অ্যালোকেট করে (অতিরিক্ত স্পেস নেয় ভবিষ্যতের জন্য), কিন্তু Big O-তে ইগনোর।
- **চেক করার টুল**: `sys.getsizeof()` ছাড়া `pympler` লাইব্রেরি (asizeof) দিয়ে ডিপ সাইজ চেক করা যায়, কিন্তু ডিফল্টে নেই।
- **প্র্যাকটিস**: LeetCode বা HackerRank-এ প্রবলেম সলভ করতে গিয়ে স্পেস দেখো।