Python Cheat Sheet for Competitive Programming

Author : Competitive Programmer's Guide **Date :** 2024-12-15

Input/Output

Fast Input:

```
python import sys input = sys.stdin.read
```

Single Line Input:

```
python x = int(input()) # Single integer arr = list(map(int, input().split())) # List of integers
```

Multiple Lines Input:

```
python import sys input = sys.stdin.read data = input().splitlines()
```

Fast Output:

```
python sys.stdout.write(str(output) + '\n')
```

Mathematics

Modular Arithmetic:

```
python MOD = 10**9 + 7 res = (a * b) % MOD
```

GCD/LCM:

```
python from math import gcd lcm = lambda x, y: x * y // gcd(x, y)
```

Exponentiation:

```
python pow(base, exp, mod) # Efficient modular exponentiation
```

Prime Numbers (Sieve of Eratosthenes):

```
python def sieve(n): is_prime = [True] * (n + 1) is_prime[0] = is_prime[1] = False for i in range(2, int(n**0.5) + 1): if is_prime[i]: for j in range(i * i, n + 1, i): is_prime[j] = False return is_prime
```

Common Data Structures

Lists:

```
python arr = [0] * n # Initialize array with size n
```

Sorting:

```
python arr.sort() # Ascending arr.sort(reverse=True) # Descending arr = sorted(arr, key=lambda x: (x[0], -x[1])) # Custom sort
```

Heaps (Priority Queue):

```
python import heapq heap = [] heapq.heappush(heap, x) # Push min_element = heapq.heappop(heap) # Pop smallest element
```

Hashmaps:

```python from collections import defaultdict, Counter freq = defaultdict(int) freq[x] += 1

count = Counter(arr) # Frequency count ```
```

Deque:

```
python from collections import deque dq = deque() dq.append(x) # Add to end dq.appendleft(x) # Add to start dq.pop() # Remove from end dq.popleft() # Remove from start
```

Sets:

```
python s = set() s.add(x) if x in s: print("Exists")
```

Searching and Binary Search

Linear Search:

```
python if x in arr: # O(n) print(arr.index(x))
```

Binary Search:

```
python from bisect import bisect_left, bisect_right idx = bisect_left(arr, x) # First position where x can be inserted
```

Loops and Iteration

Iterate with Index:

```
python for i, val in enumerate(arr): print(i, val)
```

Reverse Iteration:

```
python for val in reversed(arr): print(val)
```

Combinations and Permutations:

```
python from itertools import permutations, combinations perms = permutations(arr) # All permutations combs = combinations(arr, r) # All combinations of size r
```

Strings

String Reversal:

```
python s[::-1]
```

Anagram Check:

```
python sorted(s1) == sorted(s2)
```

Palindrome Check:

```
python s == s[::-1]
```

Character Count:

```
python from collections import Counter freq = Counter(s)
```

Dynamic Programming Basics

Initialize 2D DP Table:

```
python dp = [[0] * m for _ in range(n)]
```

Recursion with Memoization:

```python from functools import lru_cache

@lru_cache(None) def solve(x): return solve(x - 1) + solve(x - 2) ```

## Graph Algorithms

### Adjacency List:

```
python graph = defaultdict(list) graph[u].append(v) # Add edge u -> v
```

### DFS:

```
python def dfs(node, visited): visited.add(node) for neighbor in graph[node]: if neighbor not in visited: dfs(neighbor, visited)
```
```

BFS:

python from collections import deque def bfs(start): q = deque([start]) visited = set([start]) while q: node = q.popleft() for neighbor in graph[node]: if neighbor not in visited: visited.add(neighbor) q.append(neighbor)

Time Complexity Hacks

Avoid O(n^2) with Sorting:

For problems requiring pair checking, sort the array and use two pointers: python arr.sort() left, right = 0, len(arr) - 1 while left < right: if arr[left] + arr[right] == target: break elif arr[left] + arr[right] < target: left += 1 else: right -= 1

Miscellaneous Tips

Infinite Value:

python INF = float('inf')

Rounding:

python round(num, 2)

Coordinate Compression:

python unique = sorted(set(arr)) compressed = {val: idx for idx, val in enumerate(unique)}