Contents

1 Misc

1.1 Debug

```
- Pre-Submit:
    - Check output is equivalent to provided test cases
    - Test more cases if not confident
        - Compute small test cases to test against
    - Determine edge-cases
- General:
    - Re-read problem
    - Explain problem & hence solution to teammate (mascot)
    - Have a snack
    - Check type casting in computation and output
    - Division by zero?
- Wrong Answer:
    - Check output format
    - Check boundary test cases
    - Check '==' vs '>=' vs '<='
    - Are values initialized & reset properly?
- Time Limit Exceeded:
    - Calculate time complexity
    - Check loops will always complete
    - Are constant factors reasonable?
    - Is all code necessary?
    - Are there inbuilt functions for the same tasks?
```

1.2 Python

Local variables are accessed faster than global

1.3 Boilerplate

```
def main():
    pass
if __name__ == "__main__":
    main()
```

1.4 Map

```
squared = map(lambda x: x**2, [1, 2, 3, 4])
```

1.5 Filter

```
even = filter(lambda x: x % 2 == 0, [1, 2, 3, 4])
```

1.6 Longest Increasing Subsequence

```
def longest_increasing_subsequence(nums):
    if not nums:
        return 0
    n = len(nums)
    lis = [1] * n
    for i in range(1, n):
        for j in range(0, i):
            if nums[i] > nums[j] and lis[i] < lis[j] + 1:
                lis[i] = lis[j] + 1
    return max(lis)
```

1.7 Longest Common Subsequence

```
def longest_common_subsequence(text1, text2):
    m, n = len(text1), len(text2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if text1[i - 1] == text2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
    return dp[m][n]
```

2 Data Structures

2.1 DefaultDict

```
from collections import defaultdict # no keyerror dict
d = defaultdict(default_value)
```

2.2 Counter

```
from collections import Counter
c = Counter("mississippi") = {'i': 4, 's': 4, 'p': 2, 'm': 1}
c.update("missouri") # adds counts of inp to counts
```

2.3 Double-Ended Queue

```
from collections import deque
q = deque([1, 2, 3, 4, 5, 6])
# q.append(n), q.popLeft()
# q.insert(i, n), q.extend([])
# q.reverse(), q.rotate()
```

2.4 Segment Tree + Example

```
class SegmentTree:
    def __init__(self, nums):
        self.n = len(nums)
        self.tree = [0] * (4 * self.n)
        self.nums = nums
        self.build(1, 0, self.n - 1)
    def build(self, node, start, end):
        if start == end:
            self.tree[node] = self.nums[start]
        else:
            mid = (start + end) // 2
            left_node = node * 2
            right_node = node * 2 + 1
            self.build(left_node, start, mid)
            self.build(right_node, mid + 1, end)
            self.tree[node] = self.tree[left_node] +
            ↪   self.tree[right_node]
    def query(self, L, R):
        return self._query(1, 0, self.n - 1, L, R)
    def _query(self, node, start, end, L, R):
        if R < start or L > end:
            return 0
        if L <= start and R >= end:
            return self.tree[node]
        mid = (start + end) // 2
        left_sum = self._query(node * 2, start, mid, L, R)
        right_sum = self._query(node * 2 + 1, mid + 1, end,
        ↪   L, R)
        return left_sum + right_sum
    def update(self, index, value):
        self._update(1, 0, self.n - 1, index, value)
    def _update(self, node, start, end, index, value):
        if start == end:
            self.tree[node] = value
            self.nums[index] = value
        else:
            mid = (start + end) // 2
            if start <= index <= mid:
                self._update(node * 2, start, mid, index,
                ↪   value)
            else:
                self._update(node * 2 + 1, mid + 1, end,
                ↪   index, value)
            self.tree[node] = self.tree[node * 2] +
            ↪   self.tree[node * 2 + 1]


data = [3, 7, 2, 8, 5]
tree = SegmentTree(data)
tree.update(2, 4)  # Update: Set the value at index 1 to 2
print(tree.query(1, 4)) # Query total in range [1, 4]
↪   inclusive
```

2.5 Heap

```
# Example implementation of heap
# Practically equivalent to minimum-spanning-tree
# Example problem this solves:
# Given N ropes of different lengths, find the minimum cost
↪   to connect these ropes, cost is the sum of their lengths
import heapq
def min(arr):
    n = len(arr)
    heapq.heapify(arr) # transform arr into heap in-place
    res = 0
    while(len(arr) > 1):
        first = heapq.heappop(arr)
        second = heapq.heappop(arr)
        res += first + second
        heapq.heappush(arr, first + second)
    return res
smallest_k = heapq.nsmallest(k, heap)
largest_k = heapq.nlargest(k, heap)
# Push to heap, pop & return smallest
smallest = heapq.heappushpop(heap, item)
```

3 Math

3.1 GCD (Euclidean Algorithm)

```
from math import gcd
```

3.2 Extended GCD

```
def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    else:
        gcd, x, y = extended_gcd(b % a, a)
        return gcd, y - (b // a) * x, x
```

3.3 Modular Inverse

```
n = pow(a, -1, b) # (n*a)modb == 1
```

3.4 Sieve of Eratosthenes

```
def sieve_of_eratosthenes(n):
    primes, p = [True for _ in range(n+1)], 2
    while p * p <= n:
        if primes[p]:
            for i in range(p * p, n + 1, p):
                primes[i] = False
        p += 1
    return [i for i in range(2, n + 1) if primes[i]]
```

3.5 Miller-Rabin

```
# probability-based primality test- >k more accurate, slower
def miller_rabin(n, k=5):
    if n < 4: return n == 2 or n == 3
    if n % 2 == 0: return False
    d, r = n - 1, 0
    while d % 2 == 0:
        d, r = d // 2, r + 1
    def witness(a):
        x = pow(a, d, n)
        if x == 1 or x == n - 1: return True
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1: return True
        return False
    return all(witness(random.randint(2, min(n - 2, 2 +
    ↪   int(2 * (pow(n.bit_length(), 2)) ** 0.5)))) for _ in
    ↪   range(k))
```

3.6 Pollard's Rho

```
# integer factorization
def pollard_rho(n):
    if n == 1: return 1
    f = lambda x: (x * x + 1) % n
    x, y, d = 2, 2, 1
    while d == 1:
        x = f(x)
        y = f(f(y))
        d = gcd(abs(x - y), n)
    if d == n:
        return pollard_rho(n)
    return d
```

4 Numeric

5 Flow

5.1 Sorting

```
a = [1, 5, 4, 3, 2]
a.sort() # sort a, set a to sorted
sorted(a) # return sorted values, leave a
```

5.2 Ternary Search

```
# Find max/min of unimodal func on interval [l, r]
def ternary_search(f, l, r, eps=1e-9):
    while abs(r - l) > eps:
        m1 = l + (r - l) / 3
        m2 = r - (r - l) / 3
        # > = min, < = max
        if f(m1) < f(m2):
            l = m1
        else:
            r = m2
    return (l + r) / 2
```

5.3 Binary Search

```
import bisect
# Returns index of x or -1 if not found
def binary_search(arr, x):
    i = bisect.bisect_left(arr, x)
    if i != len(arr) and arr[i] == x:
        return i
    else:
        return -1
```

5.4 Memoization (+Fibonacci)

```python
# Top-Down DP
# Remember previous computation result, prevent unnecessary
↪  computation
import functools
@functools.lru_cache(maxsize=None)
def fibonacci(n):
    if n < 2: return n
    return fibonacci(n - 1) + fibonacci(n - 2)
```

5.5 Tabulation

```python
# Bottom-up DP
# Fill a table, then compute solution from table
def fibonacci(n):
    m = [0, 1]
    for i in range(2, n+1):
        m.append(m[i-2] + m[i-1])
    return m[n]
```

6 Graph

7 Geometry

8 Strings