COMP 321

Exercise Activity

Group 24: Claire O’Leary, Karl Tchakmakian, Tony Le

Nov. 2025

Table Of Contents

Theme Rationale

Problem

Davy Jones and the Last Hidden Coin

Figure 1: Photo by Walt Disney Studio and DLC Studio on Canva

Davy Jones, the feared captain of the ghostly Flying Dutchman, has spent years plundering countless coins from sunken ships. To keep thieves guessing, he curses every coin in each pile except for a single untouched piece. As a daring adventurer, you have intercepted a ledger describing how many coins lie in each of his piles.

Before the night ends, the restless crew will shout out Q requests for specific coin positions within the combined hoard. For every request, you must determine which pile hides that exact coin before Davy Jones notices your intrusion.

Input

Input consists of three lines. The first line contains two integers N and Q ($1 \leq N, Q \leq 10^9$), the number of piles and the number of crew requests. The second line contains N integers $g_1, g_2, \ldots, g_N$ ($1 \leq g_i \leq 10^9$), where g_i is the number of coins in pile i. The third line contains Q integers $k_1, k_2, \ldots, k_Q$ ($1 \leq k_j \leq 10^9$) describing the positions the crew wants you to inspect in the combined collection.

Output

Output Q lines. For each request k_j, print the 0-based index of the pile that contains the k_j-th coin when the piles are concatenated in order from 0 to $N-1$. If the requested coin does not exist, print `impossible`.

Sample Input 1

```
5 5
3 1 10 2 6
1 3 4 11 23
```

Sample Output 1

```
0
0
1
2
impossible
```

Sample Input 2

```
1 2
5
1 5
```

Sample Output 2

```
0
0
```

Sample Input 3

```
3 3
2 8 4
2 9 14
```

Sample Output 3

```
0
1
2
```

Sample Input 4

```
3 3
4 2 5
6 7 11
```

Sample Output 4

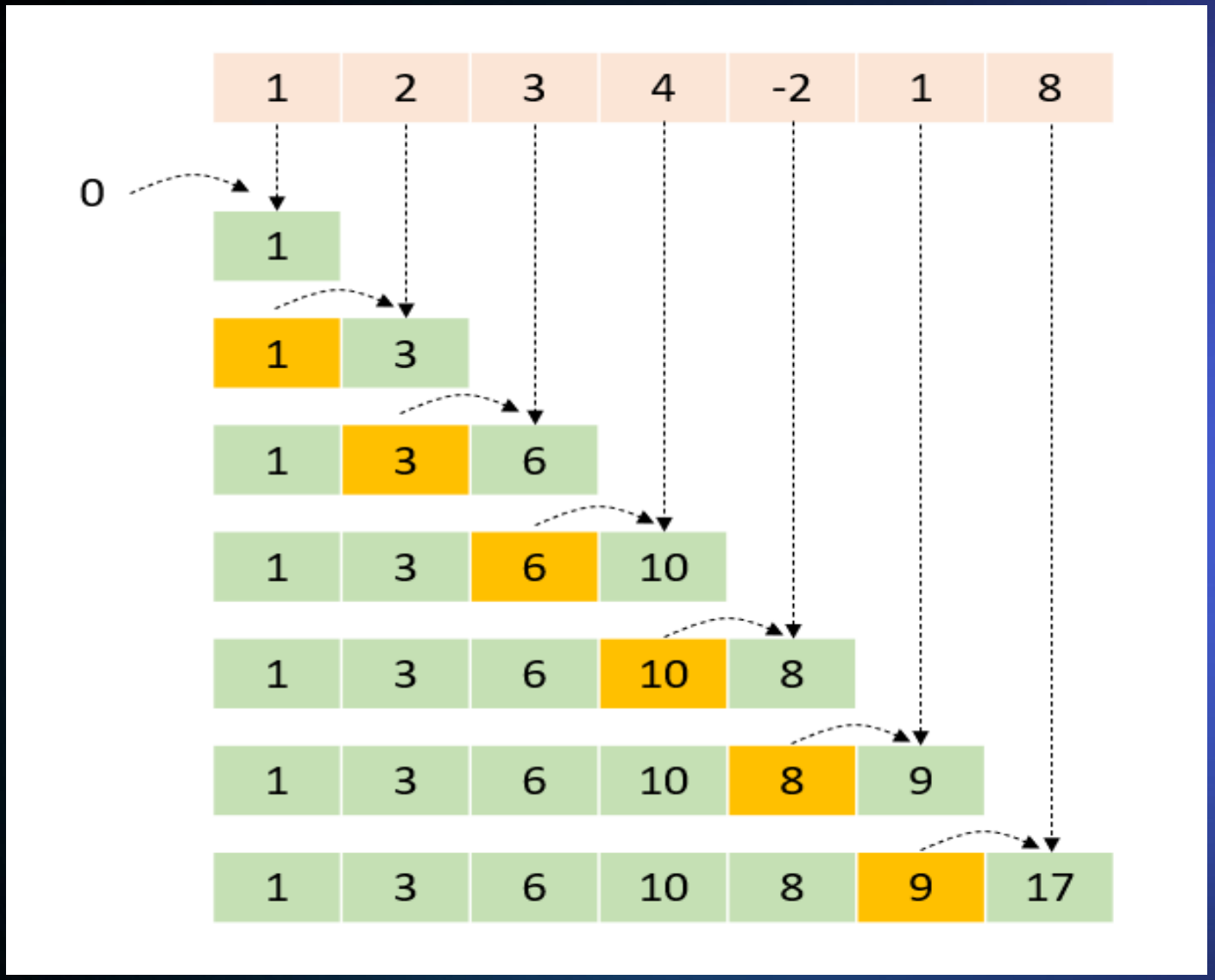
```
1
2
2
```

Sample Input 5

```
4 5
1 1 1 1
1 2 3 4 5
```

Sample Output 5

```
0
1
2
3
impossible
```



Prefix sum array technique

submissions > accepted > solution.py

```python
N, Q = map(int, input().split())
piles = list(map(int, input().split()))
q = list (map(int, input().split()))

prefix = []
cur = 0
for value in piles:
    cur += value
    prefix.append(cur)

def binary_search(arr, target):
    lo, hi = 0, len(arr)
    while lo < hi:
        mid = (lo + hi) // 2
        if arr[mid] >= target:
            hi = mid
        else:
            lo = mid + 1
    return lo

total = prefix[-1]
for query in queries:
    if query > total or query < 1:
        print("impossible")
    else:
        print(binary_search(prefix, query))
```

Time Complexity : O(n + log n)

Worst Case : O(n + log n)

Best Case : O(1) -with O(n) prefix array overhead

Space Complexity : O(N +Q)

Accepted Solution

submissions › time_limit_exceeded › solution.py

```python
N, Q = map(int, input().split())
piles = list(map(int, input().split()))
queries = list(map(int, input().split()))

for target in queries:
    running = 0
    for i, coins in enumerate(piles):
        running += coins
        if running >= target:
            print(i)
            break
    else:
        print("impossible")

```

Time Complexity : O(n)

Worst Case : O(N*Q)

Best Case : O(1)

Space Complexity : O(1)

Time-limit Exceeded Solution

Description.txt

```
Sample test cases
=================
- `sample01`: `[3,1,10,2,6]` with full-range queries to show hits in early, mid, last, and unreachable positions.
- `sample02`: `[5]` double-checks single-pile boundaries (first vs. last coin).
- `sample03`: `[2,8,4]` hits each pile boundary exactly to confirm prefix handling.
- `sample04`: `[4,2,5]` proves interior hits and an out-of-range query produce mixed answers.
- `sample05`: `[1,1,1,1]` keeps piles uniform so graders can sanity-check straight counting.

Secret test cases (handpicked)
==============================
- `secret01`: `[10]`, queries `[1,10,11]` for the minimal full case plus impossible.
- `secret02`: `[1_000_000]*5` stresses large counts and an extra query at 5,000,001.
- `secret03`: `[1]*1000` scales $N$ and probes first/middle/last/impossible slots.
- `secret04`: `[10,1,10,1]` alternates pile sizes to flush out off-by-one errors.
- `secret05`: `[500]` with queries `1..501` checks dense output and line formatting.
- `secret06`: 25k piles of 1 with queries for 25,001 coins triggers TLE in naive $O(NQ)$.

Random tests
============
- `secret07+`: pseudo-random cases within the byte budget
```

Test Cases

test_case_generator › generate.py

```
62  def main():
67      for i, (c, q) in enumerate(SAMPLES, 1):
68          write_case("../data/sample", "sample", i, c, q)
69
70      curr_bytes = 0
71      secret_idx = 1
72
73      print("Writing Handpicked Secrets (Including TLE case)...")
74      for c, q in HANDPICKED_SECRETS:
75          curr_bytes += write_case("../data/secret", "secret", secret_idx, c, q)
76          secret_idx += 1
77
78      print("Writing Random Secrets...")
79      while curr_bytes < MAX_BYTES or secret_idx <= MIN_SECRETS:
80          n_piles = RNG.randint(1, 2000)
81          # 20% chance of a large pile
82          max_val = 100000 if RNG.random() < 0.2 else 1000
83          counts = [RNG.randint(1, max_val) for _ in range(n_piles)]
84
85          # 10% chance of a large query
86          max_query = 1000000000000000000 if RNG.random() < 0.1 else 1000000000
87          n_queries = RNG.randint(1, 2000)
88          queries = [RNG.randint(1, max_query) for _ in range(n_queries)]
89
90
91          curr_bytes += write_case("../data/secret", "secret", secret_idx, counts, queries)
92          secret_idx += 1
93
```

Generator

input_format_validators > validate.ctd

```
# Validate header: N (1..1e9) and Q (1..1e9), separated by a single space.
INT(1,1000000000,N) SPACE INT(1,1000000000,Q) NEWLINE

# Validate the line containing N pile sizes gi, each in [1, 1e9],
# separated by single spaces and terminated by a newline.
INT(1,1000000000) {SPACE INT(1,1000000000)}* NEWLINE

# Validate the line containing Q query values kj, each in [1, 1e9],
# separated by single spaces and terminated by a newline.
INT(1,1000000000) {SPACE INT(1,1000000000)}* NEWLINE

# No trailing tokens or blank lines are allowed.
EOF

```

Validator

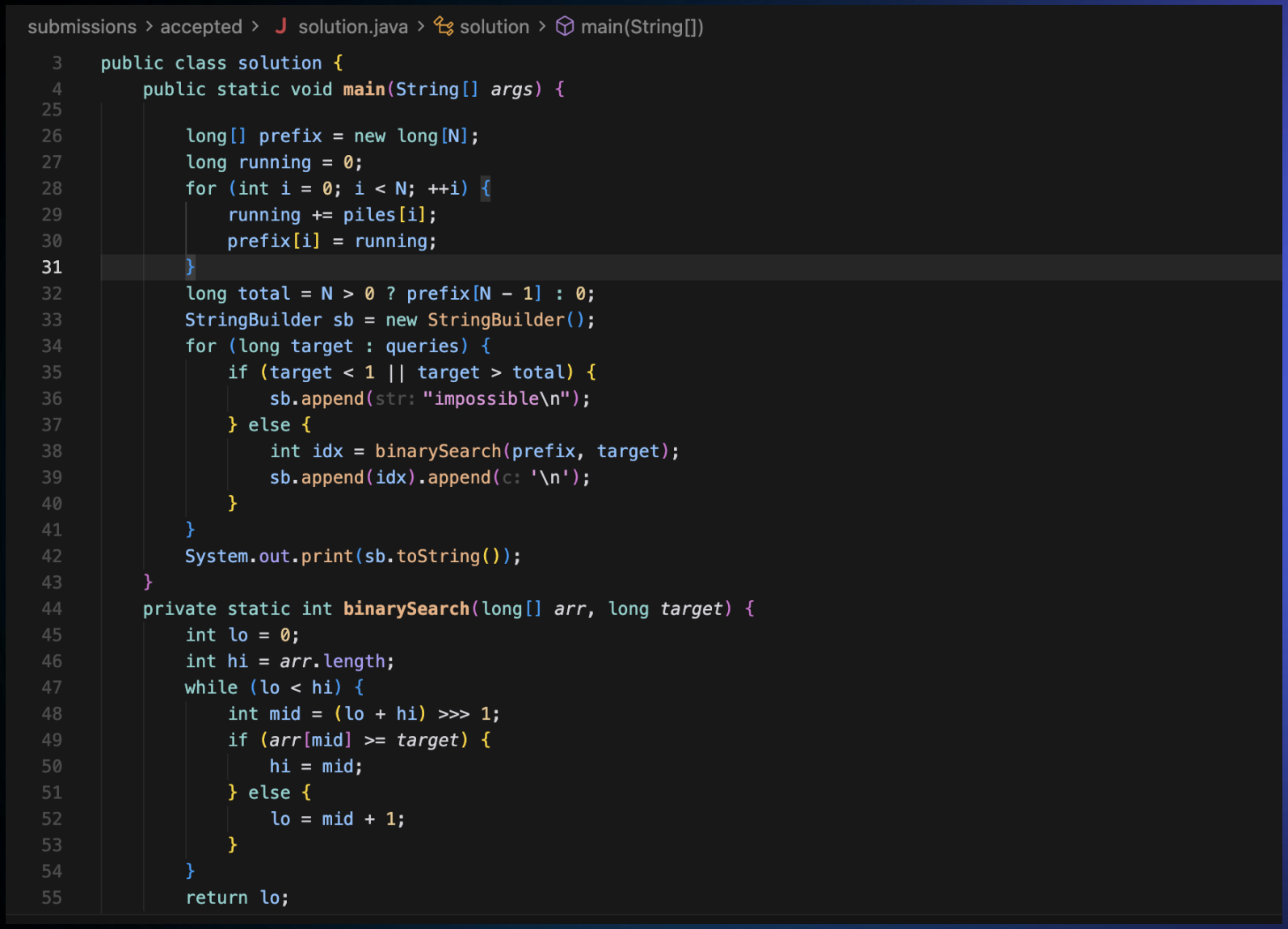
submissions > accepted > solution.cpp

```cpp
int main() {
    }

    std::vector<long long> prefix(N);
    long long running = 0;
    for (int i = 0; i < N; ++i) {
        running += piles[i];
        prefix[i] = running;
    }
    long long total = prefix.back();
    for (long long target : queries) {
        if (target < 1 || target > total) {
            std::cout << "impossible\n";
        } else {
            int idx = binarySearch(prefix, target);
            std::cout << idx << "\n";
        }
    }

    return 0;
}
int binarySearch(const std::vector<long long>& arr, long long target) {
    int lo = 0;
    int hi = static_cast<int>(arr.size());
    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        if (arr[mid] >= target) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
```

C++


submissions > accepted > solution.java > solution > main(String[])
public class solution {
public static void main(String[] args) {
long[] prefix = new long[N];
long running = 0;
for (int i = 0; i < N; ++i) {
running += piles[i];
prefix[i] = running;
}
long total = N > 0 ? prefix[N - 1] : 0;
StringBuilder sb = new StringBuilder();
for (long target : queries) {
if (target < 1 || target > total) {
sb.append(str: "impossible\n");
} else {
int idx = binarySearch(prefix, target);
sb.append(idx).append(c: '\n');
}
}
System.out.print(sb.toString());
}
private static int binarySearch(long[] arr, long target) {
int lo = 0;
int hi = arr.length;
while (lo < hi) {
int mid = (lo + hi) >>> 1;
if (arr[mid] >= target) {
hi = mid;
} else {
lo = mid + 1;
}
}
return lo;

Java

Thank you !