URL VIDEOS

Explicación enunciado del problema: https://www.youtube.com/watch?v=VuOrD06BQ3k
Explicación código: https://www.youtube.com/watch?v=SpYsGIvF32M
Análisis de complejidad: https://youtu.be/HBfcu15wCCc

ENUNCIADO DEL PROBLEMA

Es la hora de comer para Topo. Su amigo, Marmota, le ha preparado un bonito juego para comer. Marmota le ha traído a Topo n montones ordenados de gusanos, de modo que el i-ésimo montón contiene ai gusanos. Etiquetó todos estos gusanos con números enteros consecutivos: los gusanos del primer montón están etiquetados con los números del 1 al a1, los gusanos del segundo montón están etiquetados con los números del a1 + 1 al a1 + a2 y así sucesivamente.

Topo no puede comerse todos los gusanos y, como todos sabemos, Topo es ciego, así que Marmota le dice las etiquetas de los gusanos más jugosos. Marmota sólo le dará un gusano a Topo si éste dice correctamente en qué montón se encuentra dicho gusano.

Entrada

La primera línea contiene un único entero n (1 ≤ n ≤ 10^5), el número de montones.

- La segunda línea contiene n enteros a1, a2, ..., an (1 ≤ ai ≤ 103, a1 + a2 + ... + an ≤ 10^6), donde ai es el número de

gusanos en la i-ésima pila.

- La tercera línea contiene un único entero m (1 ≤ m ≤ 10^5), el número de gusanos jugosos dicho por Marmot.
- La cuarta línea contiene m enteros q1, q2, ..., qm , las etiquetas de los gusanos jugosos.

Salida

Imprime m líneas en la salida estándar. La i-ésima línea debe contener un número entero, que representa el número de la pila donde está el gusano etiquetado con el número qi.

URL DE GITLAB O GITHUB PROYECTO EN FORMA PRIVADA, USUARIO madarme@ufps.edu.co, con rol maintener

https://gitlab.com/ejercicios-proyecto-ada/ejercicio-9-worms.git

MÉTODO 1

```java
public int[] metodoCandido(int[] gusanos, int[] jugosos) {
    int n = gusanos.length;
    int m = jugosos.length;
    int[] pilas = new int[n];
    pilas[0] = gusanos[0];
    for (int i = 1; i < n; i++) {
        pilas[i] = pilas[i - 1] + gusanos[i];
    }

    int[] result = new int[m];
    for (int j = 0; j < m; j++) {
        for (int i = 0; i < n; i++) {
            if (jugosos[j] <= pilas[i]) {
                result[j] = i + 1;
                break;
            }
        }
    }
    return result;
}
```

Eficacia	El método es eficaz, puesto a que tiene una solución para todas las entradas válidas ingresadas.
	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3 Consultas: 3 Resultado de la invocación: 1 2 3
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11

Eficiencia	El método en entradas pequeñas es eficiente, sin embargo en números grandes y cuando m y n son valores cercanos, se vuelve demasiado lento al realizar tantas comparaciones para la búsqueda.
	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3 Consultas: 3 Resultado de la invocación: 1 2 3 **Tiempo: 107 microsegundos**
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices **Tiempo: 2536509 microsegundos**
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11 **Tiempo: 40 microsegundos**

Correctitud	El algoritmo es correcto, ya que da un resultado esperado para todas las entradas posibles
	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3 Consultas: 3 Resultado Esperado: 1 2 3 Resultado de la invocación: 1 2 3
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado esperado: 52058 7422 48687 28152 89723 84218 12108 19196… Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado esperado: 2 4 9 10 11 11 11 11 11 11 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11

Completitud	El método no es completo ya que aunque la lógica sirve para hallar el resultado, es demasiado lento para producir el resultado esperado.
	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3 Consultas: 3 Resultado de la invocación: 1 2 3
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11

MÉTODO 2

```java
public int[] metodoOptimo(int[] gusanos, int[] jugosos) {
    int n = gusanos.length;
    int m = jugosos.length;
    int[] pilas = new int[n];
    pilas[0] = gusanos[0];
    for (int i = 1; i < n; i++) {
        pilas[i] = pilas[i - 1] + gusanos[i];
    }

    int result[] = new int[m];
    for (int j = 0; j < m; j++) {
        int index = binarySearch(pilas, jugosos[j]);
        if (index < 0) {
            index = 0;
        }
        result[j] = index + 1;
    }
    return result;
}
```

```java
private int binarySearch(int[] pila, int gusano) {
    int izq = 0, der = pila.length - 1;
    int resultado = -1;

    while(izq<=der){
        int mid = izq + (der - izq)/2;

        if(pila[mid] >= gusano){
            resultado = mid;
            der = mid - 1;
        }else{
           izq = mid + 1;
        }
    }
    return resultado;
}
```

Eficacia	El método es eficaz para todas las entradas posibles.
	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3

	Consultas: 3 Resultado de la invocación: 1 2 3
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11
	El método es eficiente ya que al usar búsqueda binaria reduce significativamente la cantidad de iteraciones al buscar.

Eficiencia	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3 Consultas: 3 Resultado de la invocación: 1 2 3 **Tiempo: 24 microsegundos**
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices **Tiempo: 23103 microsegundos**
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11 **Tiempo: 28 microsegundos**
Correctitud	El método devuelve salidas esperadas para todas las entradas posibles.

	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3 Consultas: 3 Resultado Esperado: 1 2 3 Resultado de la invocación: 1 2 3
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado esperado: 52058 7422 48687 28152 89723 84218 12108 19196… Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado esperado: 2 4 9 10 11 11 11 11 11 11 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11
Completitud	El método es completo ya que con la lógica usada se asegura una respuesta correcta si existe.

	Instancia 1: 3 2 4 1 4 9 Tamaño pilas: 3 Consultas: 3 Resultado de la invocación: 1 2 3
	Instancia 2: Tamaño pilas: 100000 Consultas: 100000 Resultado de la invocación: 52058 7422 48687 28152 89723 84218 12108 19196… Cantidad muy grande de índices
	Instancia 3: 1 1 1 2 1 1 1 1 1 1 100 1 99 2 5 10 11 13 15 20 30 50 51 70 Tamaño pilas: 13 Consultas: 11 Resultado de la invocación: 2 4 9 10 11 11 11 11 11 11 11