Clasificatorio AdaByron URJC 2022

Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	Categoría
A - La suma de un cuadrado	Ad-hoc
B - Cadena de favores	Grafos, Componentes Fuertemente Conexas
C - La infractuosa búsqueda de un despacho	Bucles
D - Dickie y las pizzas matemáticas	Programación dinámica y Teoría de números
E - Siendo rico gracias a la bolsa	Adh-hoc y Long long

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - La suma de un cuadrado	66	1.57MB
B - Cadena de favores	15	280KB
C - La infractuosa búsqueda de un despacho	11	2.39MB
D - Dickie y las pizzas matemáticas	32	168KB
E - Siendo rico gracias a la bolsa	105	24KB
- **Total**	229	4.4MB (+-)

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - La suma de un cuadrado	Teamto de Verano	40
B - Cadena de favores	Teamto de Verano	9
C - La infractuosa búsqueda de un despacho	Teamto de Verano	24
D - Dickie y las pizzas matemáticas	-	-
E - Siendo rico gracias a la bolsa	DTX	41

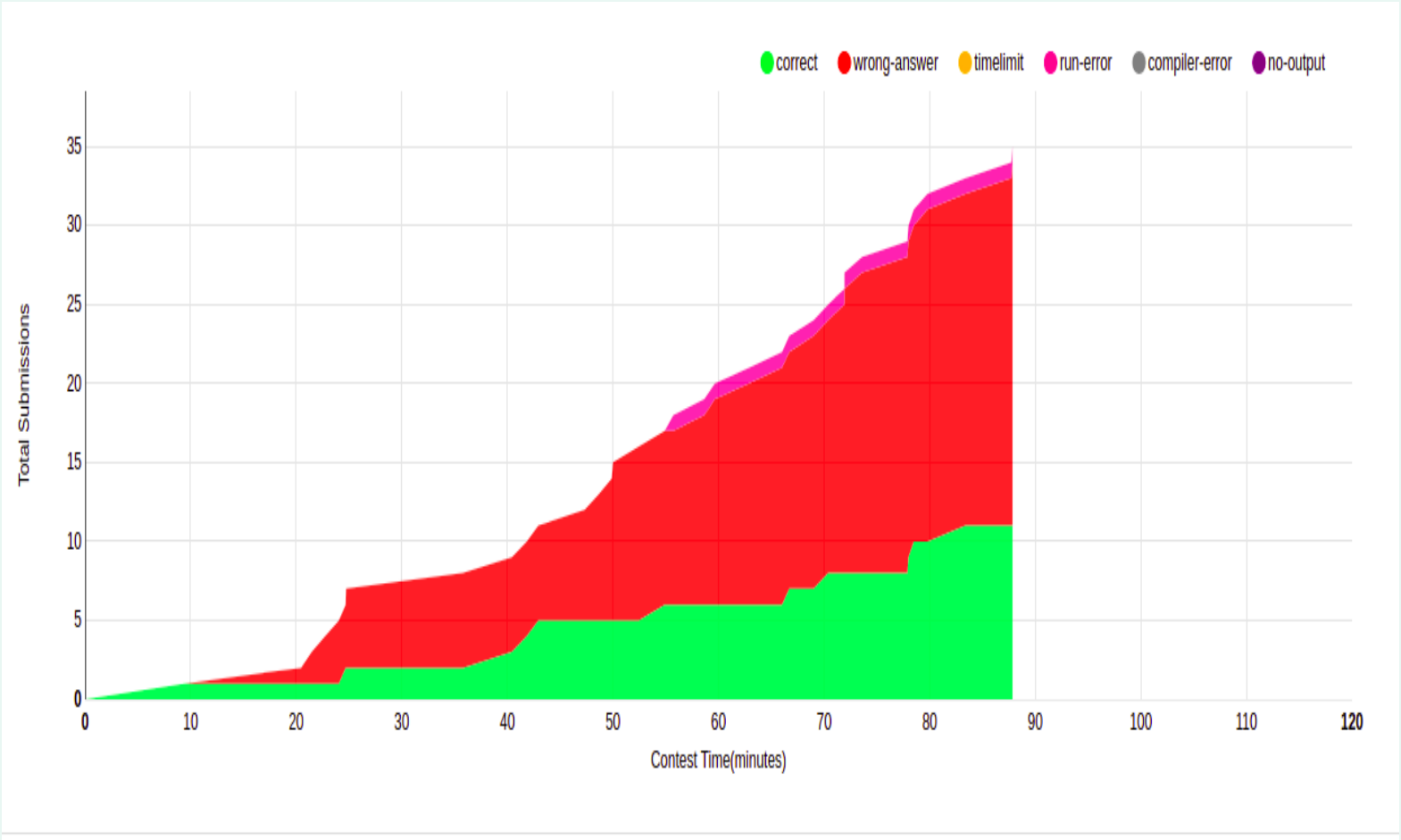
*Antes de congelar el marcador.

Estadísticas*

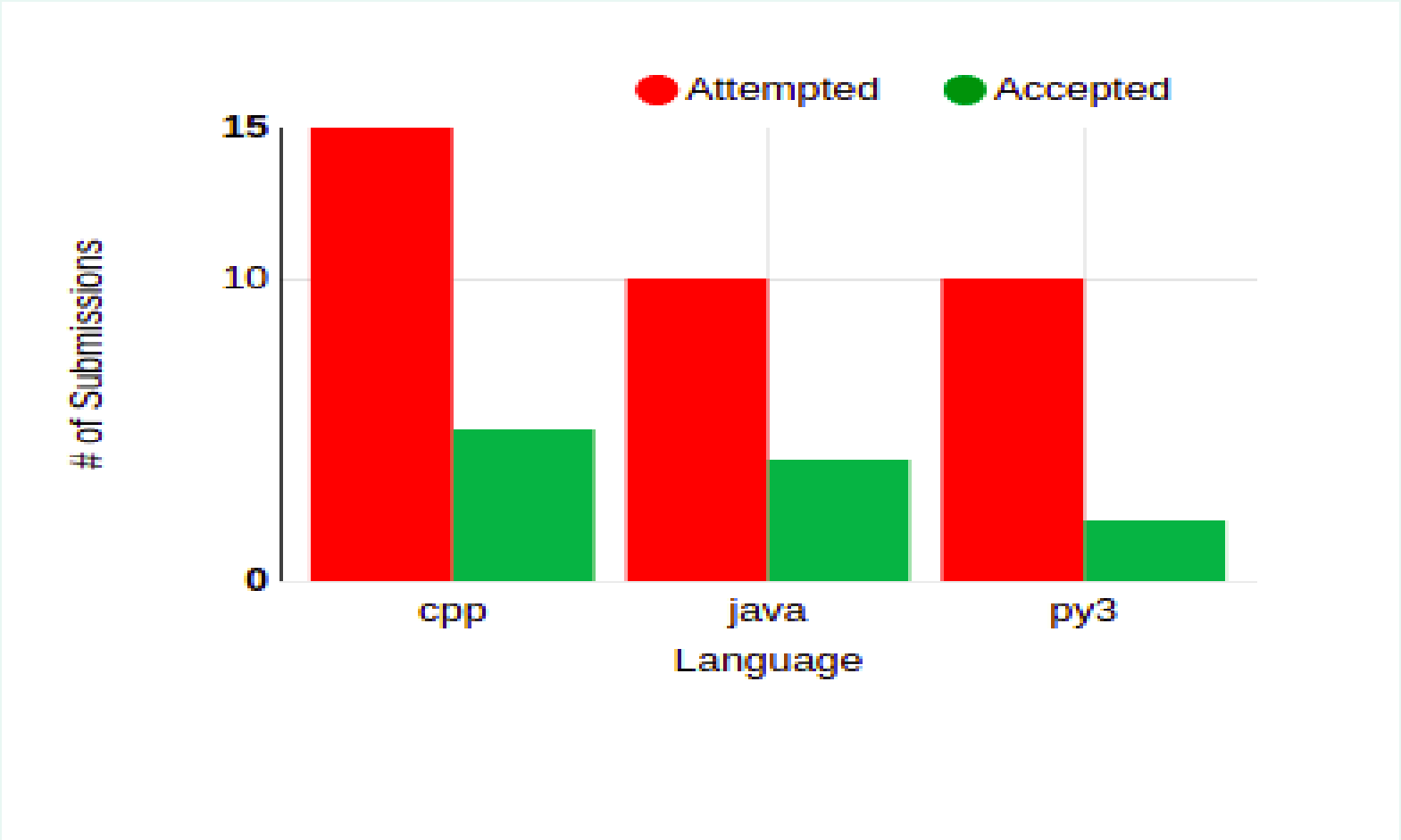
Problema	Envíos	Válidos	% éxito
A - La suma de un cuadrado	2	2	100 %
B - Cadena de favores	2	10	20 %
C - La infractuosa búsqueda de un despacho	5	13	38.50 %
D - Dickie y las pizzas matemáticas	0	0	0 %
E - Siendo rico gracias a la bolsa	2	13	15.40 %

*Antes de congelar el marcador.

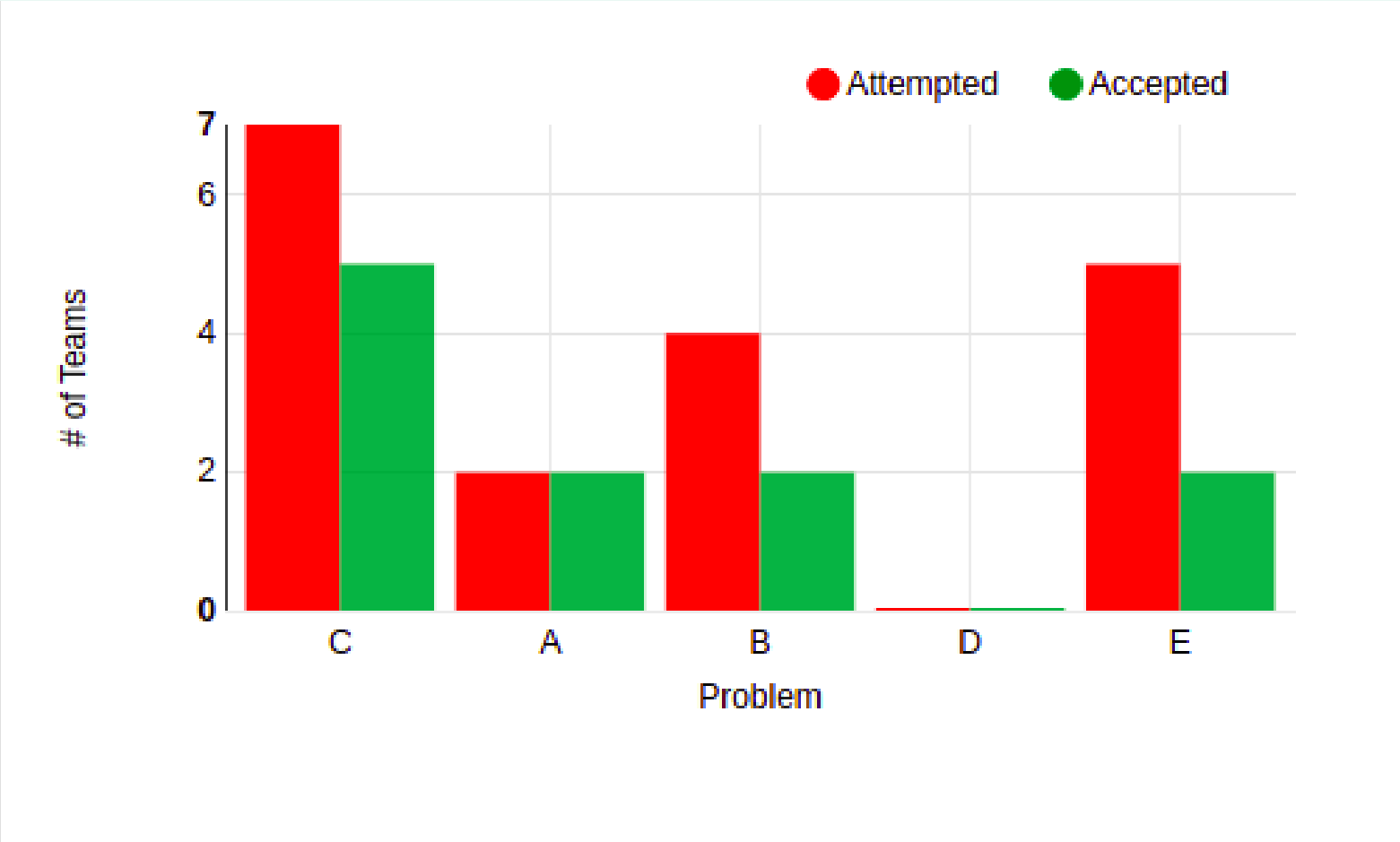
Estadísticas varias



Estadísticas varias



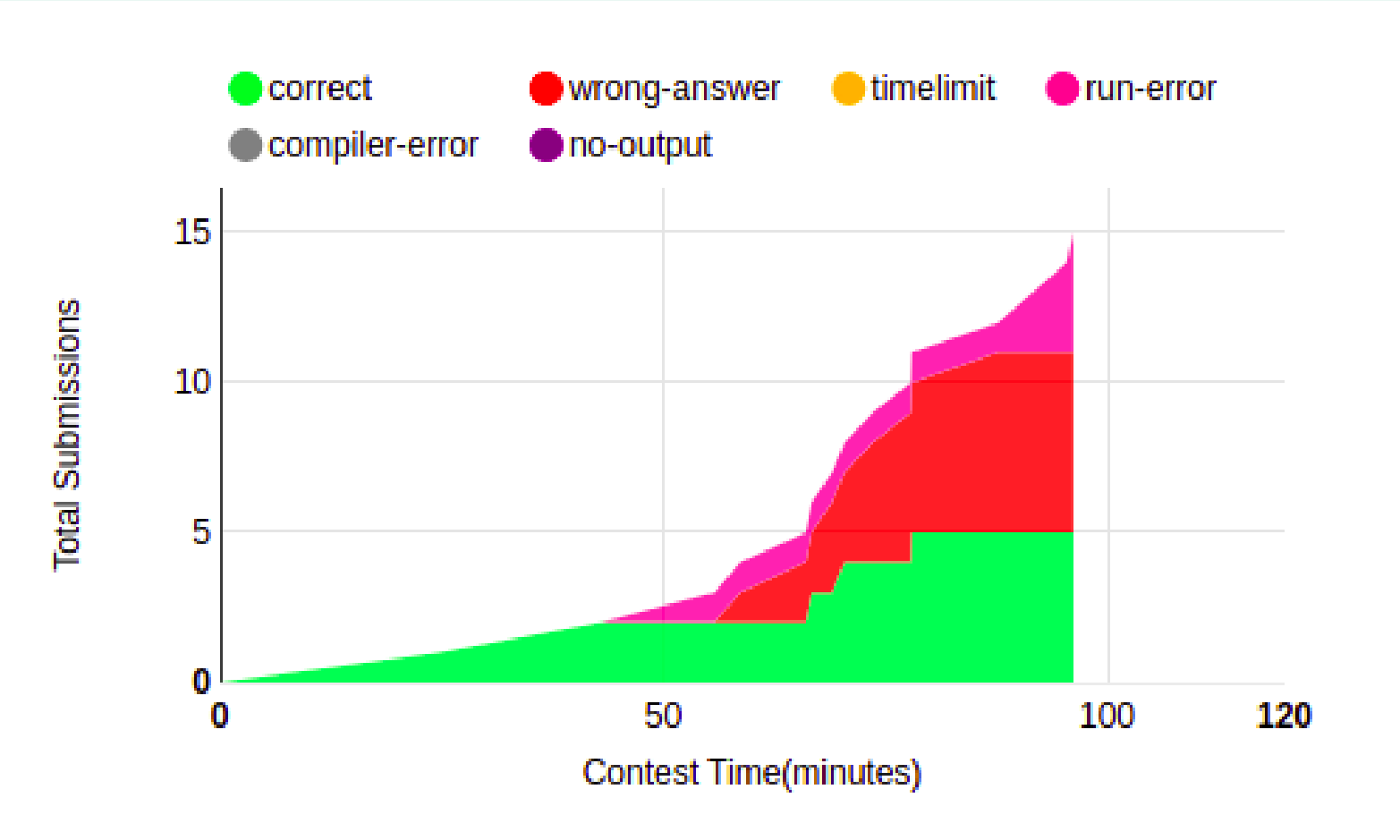
Estadísticas varias



C. La infructuosa búsqueda de un despacho

Envíos	Válidos	% éxito
5	13	38.50 %

C. La infructuosa búsqueda de un despacho



C. La infructuosa búsqueda de un despacho

Este problema se puede resumir en encontrar las listas de números que cumplan dos condiciones:

1. Al menos uno de los números de la lista debe de ser "0".
2. La suma de la lista de números debe ser positiva (> 0).

Una vez se ha determinado qué listas cumplen ambas condiciones, se debe indicar cuál de ellas maximiza la suma de los números de la lista.

Y bien, ¿qué necesitas saber para resolver este problema?

- Leer.
- Conceptos básicos de programación: bucles, condiciones, etc.
- Leer.

C. La infructuosa búsqueda de un despacho

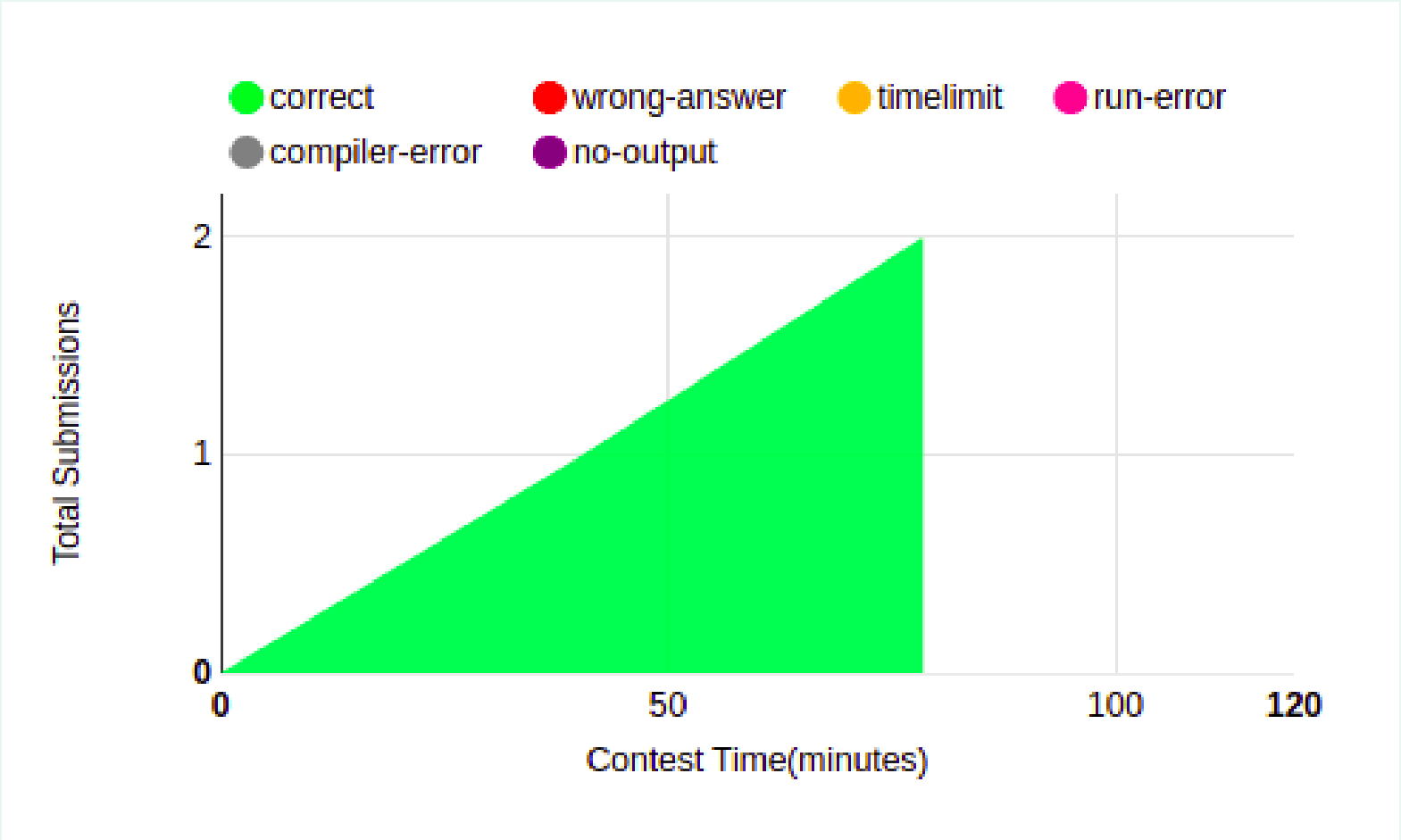
Una resolución genérica del problema planteado se ilustra en el siguiente pseudocódigo:

```
para cada (caso de prueba) {
    para cada (despacho) {
        para cada (sitio) {
            sumo su calidad
            ¿hay un sitio libre?
        }
        ¿es elegible el despacho?
        ¿es el mejor despacho encontrado?
    }
    Mostrar el mejor despacho encontrado
        o indicar que no hay despachos disponibles
}
```

A. La suma de un cuadrado

Envíos	Válidos	% éxito
2	2	100 %

A. La suma de un cuadrado



A. La suma de un cuadrado

Este problema nos pedía maximizar la suma de los elementos de una submatriz de tamaño N/2 x N/2 situada en la esquina superior izquierda de la matríz NxN
Podíamos aplicar, tantas veces quisiéramos, las siguientes operaciones:

- Invertir todos los elementos de una fila r cualquiera
- Invertir todos los elementos de una fila c cualquiera

11	4	8	11
5	12	5	4
1	7	10	4
6	9	11	10

Figura: Visualización de un cuadrado NxN y sus subcuadrados N/2 x N/2

A. La suma de un cuadrado

Para este problema hace falta darnos cuenta de como podemos rotar y qué es a lo que conlleva.
Para el elemento en la posición (2,2) del cuadrado; *12*, solo es posible moverlo a 4 posibles posiciones; (2,3), (3,2), (3,3) o (2,2). O, visto de otra forma:

Rota R	Rota C	Pos.
NO	NO	(2,2)
SI	NO	(3,2)
NO	SI	(2,3)
SI	SI	(3,3)

Visto así, pareciera que habría que aplicar combinaciones desde 0 hasta 2^M siendo M de tamaño (N/2 x N/2). TLE

A. La suma de un cuadrado

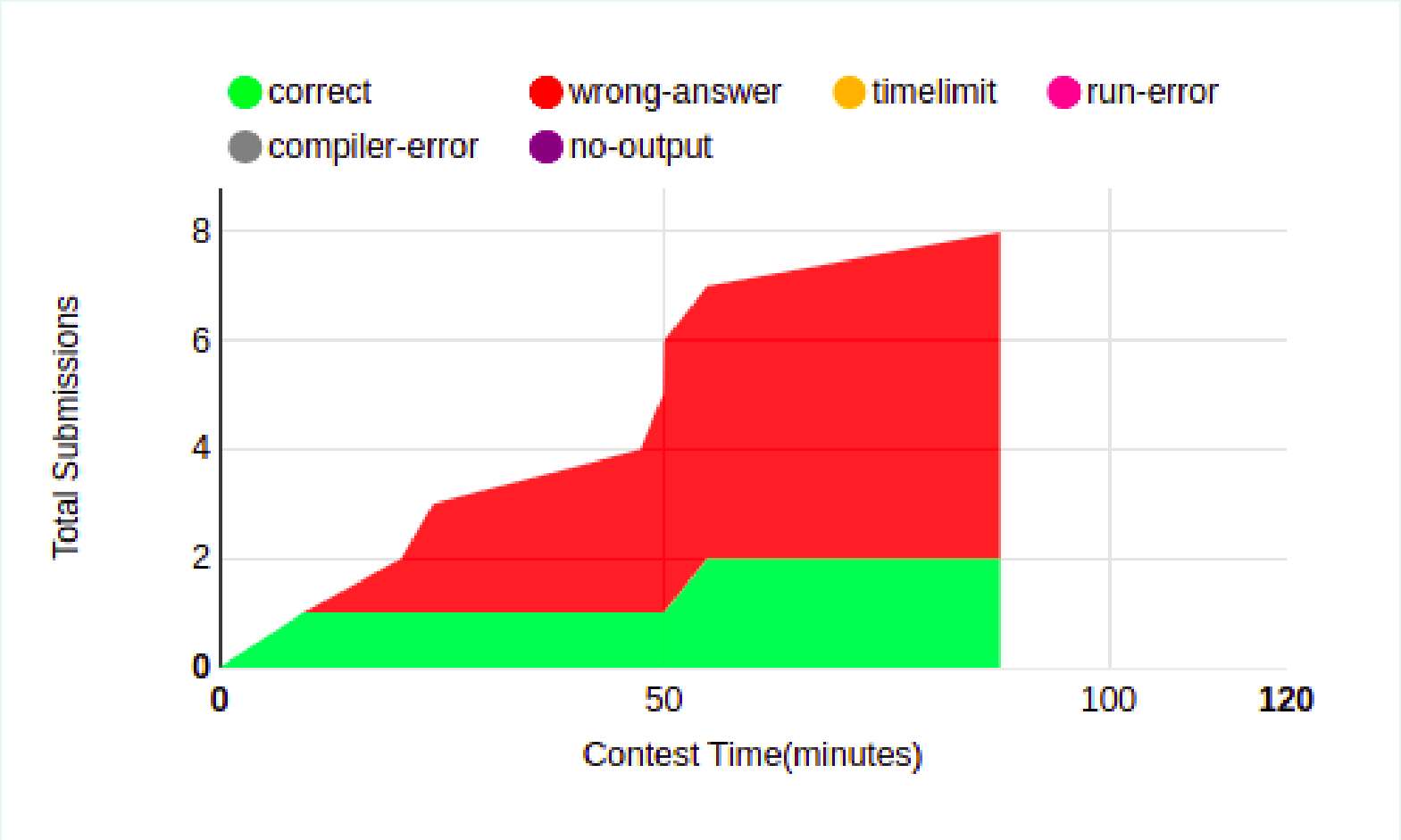
Intentemos pensar, de nuevo, en la posición (2,2) del cuadrado, si la movemos de 4 formas distintas, significa que la fila y la columna adyacente (1,2) y (2,1) respectivamente tambien se moverán, pero estas, a su vez, se pueden mover independientemente en 4 posiciones.
Esta idea la podemos extender de tal forma que no hace falta implementar la ïnversión"de columnas y filas, simplemente, el problema se reduce a saber quien es el elemento mayor entre los 4 posibles movimientos.
Para (1,1), las posibles casillas que se quedarían en esa posición serían (1,1), (1,4), (4,1) y (4,4). Similarmente para (2,2) con el ejemplo ya tomado.
Visto así, el problema se resuelve tal que:

```
M = N/2
ans = 0
for i in range(0, M):
  for j in range(0, M):
    ans += max(mat[i][j], mat[N-1-i][j],
               mat[i][N-1-j], mat[N-1-i][N-1-j])
```

B. Cadena de favores

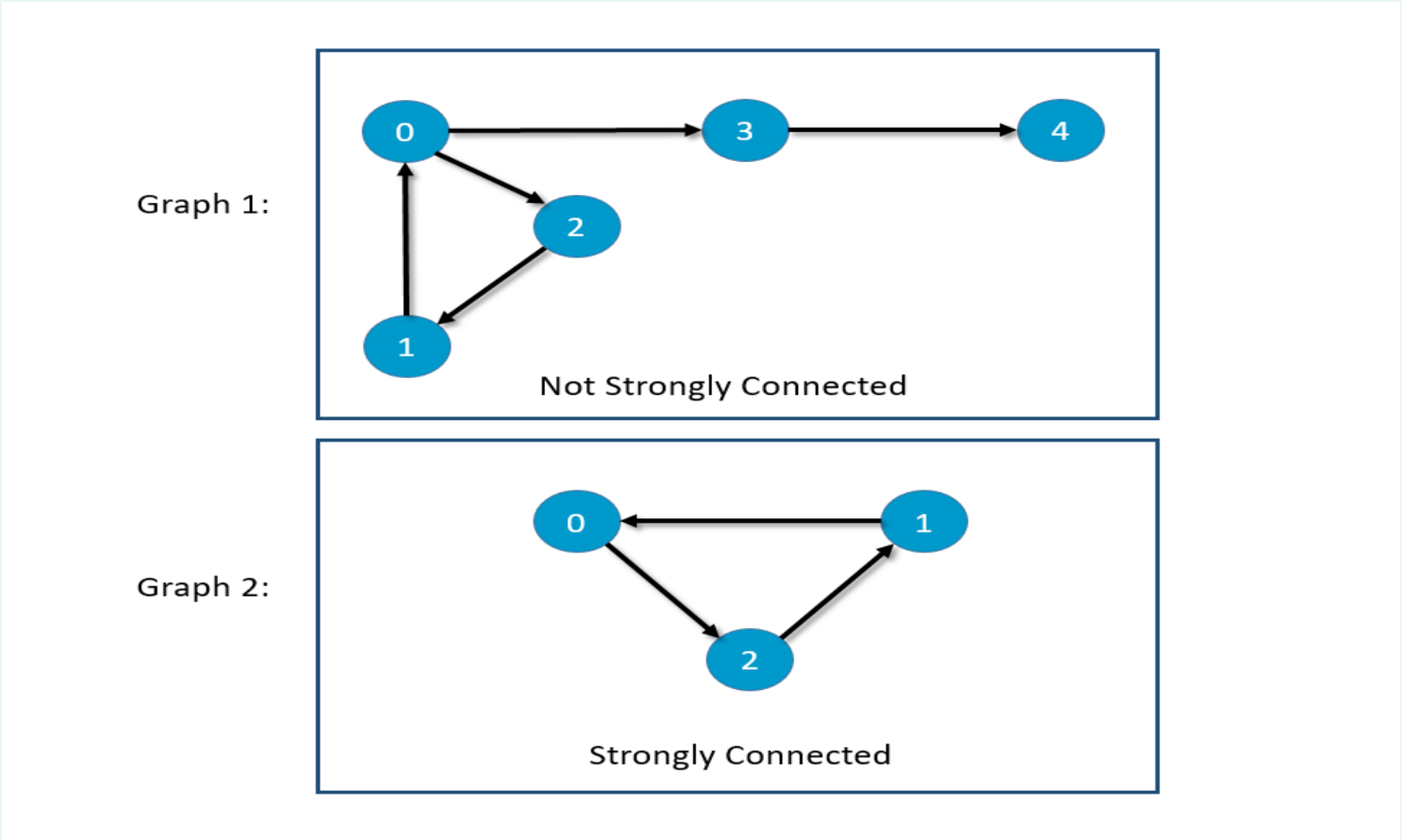
Envíos	Válidos	% éxito
2	10	20 %

B. Cadena de favores



B. Cadena de favores

En este problema se nos pide aplicar una operación bien conocida sobre grafos: encontrar componentes fuertemente conexas. En concreto, el problema era aún más específico: debemos devolver SI en caso de que el grafo conforme **una única componente fuertemente conexa** y NO en caso contrario.



B. Cadena de favores

Para resolver este problema podemos seguir varios enfoques:

- Lanzar un BFS desde cualquier nodo, si todos los nodos se visitan devuelvo $SI \rightarrow$ Válido para grafos no dirigidos, en este caso tenemos un grafo dirigido, por lo tanto **ERROR**
- Utilizar Floyd-Warshall para encontrar el camino más corto que conecte todos los nodos $\rightarrow$ Complejidad $O(n^3)$, por lo tanto **ERROR**
- Lanzar un DFS/BFS desde todos los vértices. En este caso concreto, esta idea puede funcionar $\rightarrow$ Complejidad $O(V * (V + E))$, por lo que sería válida en este caso, **ACIERTO**

B. Cadena de favores

Sin embargo, existe una mejor opción: el algoritmo DFS basado en Kosaraju. Los pasos a seguir son:

- Marcar todos los vértices como **no visitados**
- Lanzar un DFS empezando por cualquier vértice v. Si con el DFS no visitamos todos los vértices, entonces devolvemos NO.
- De lo contrario, invertimos todas las aristas del grafo (intercambiando orígenes y destinos) y volvemos a marcar todos los nodos como **no visitados**
- Lanzaar un DFS sobre el grafo invertido, empezando desde el mismo vértice v. Si el DFS no visita todos los vértices, entonces se devuelve NO. En caso contrario, se devuelve SI

B. Cadena de favores

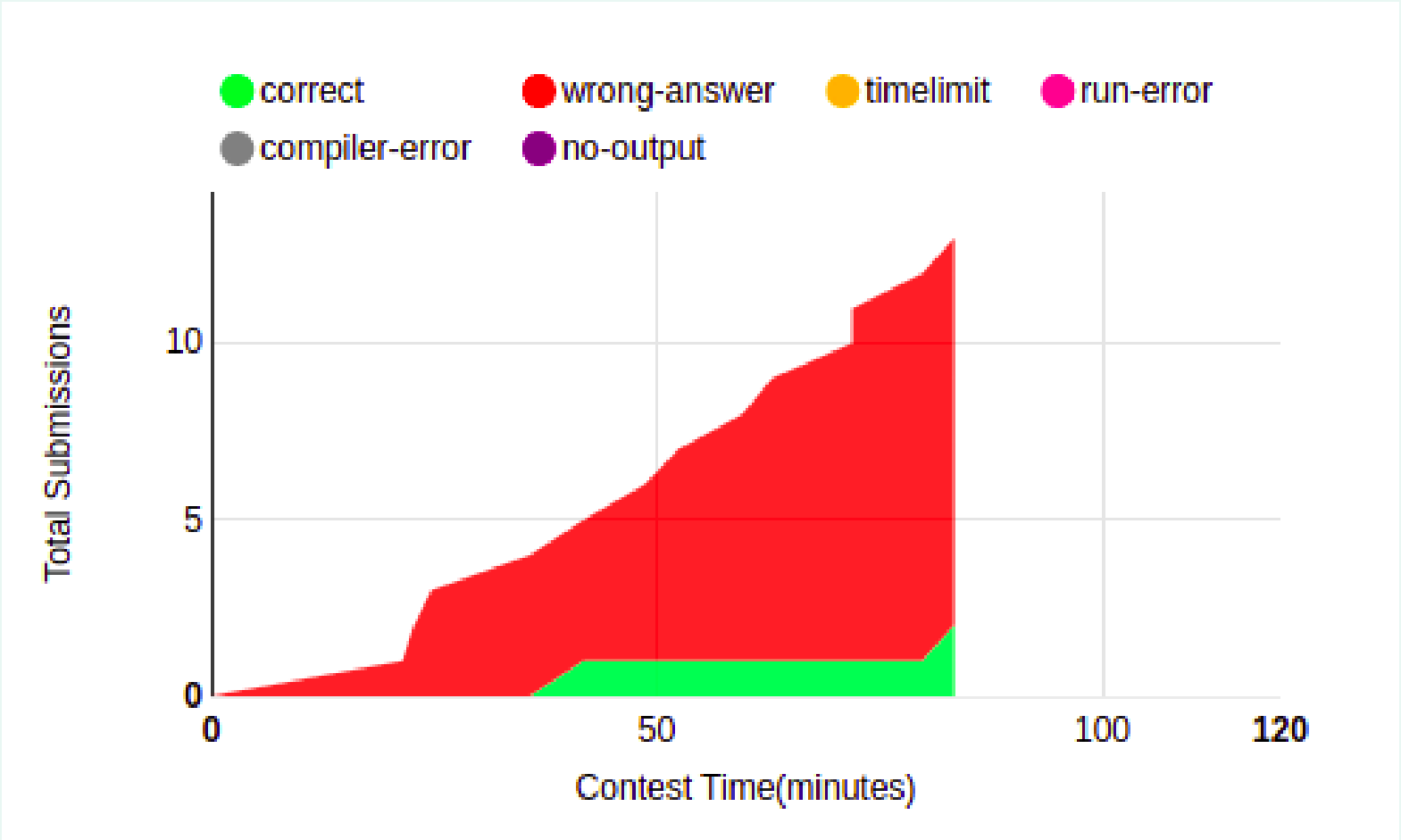
Otras ideas que surgieron probando los problemas:

- Contar el número de destinos únicos que existen en el grafo. Si este número es igual al número de nodos, entonces devuelvo $SI \rightarrow$ Falla cuando hay más de una componente conexa.

E. Siendo rico gracias a la bolsa

Envíos	Válidos	% éxito
2	13	15.40 %

E. Siendo rico gracias a la bolsa



E. Siendo rico gracias a la bolsa

Se va a invertir una cantidad de dinero D de la siguiente forma.

- La primera vez se comprará sea el precio que sea.
- En el momento que el valor de la moneda cuando invirtió sea o supere el doble el venderá (de hecho el aunque sea superior, venderá siempre cuando llegue al doble).
- Volverá a comprar cuando el valor de la moneda cuando vendió sea la mitad.

¿Cuántas ganancias se obtendrá?

E. Siendo rico gracias a la bolsa

El mayor beneficio posible se calculaba de la siguiente forma. D multiplicado por 2^{25} (50 valores en el histórico dónde, 25 compra y 25 vende). D el valor máximo es 1000 por 2^{25}, nos da 1000 * 33.554.432 = 33.554.432.000. El valor máximo de un entero es 2.147.483.647, si no se usaba el tipo long -> WA.
¡Cuidado con el uso de divisiones! (aunque hemos sido buenos..)

E. Siendo rico gracias a la bolsa

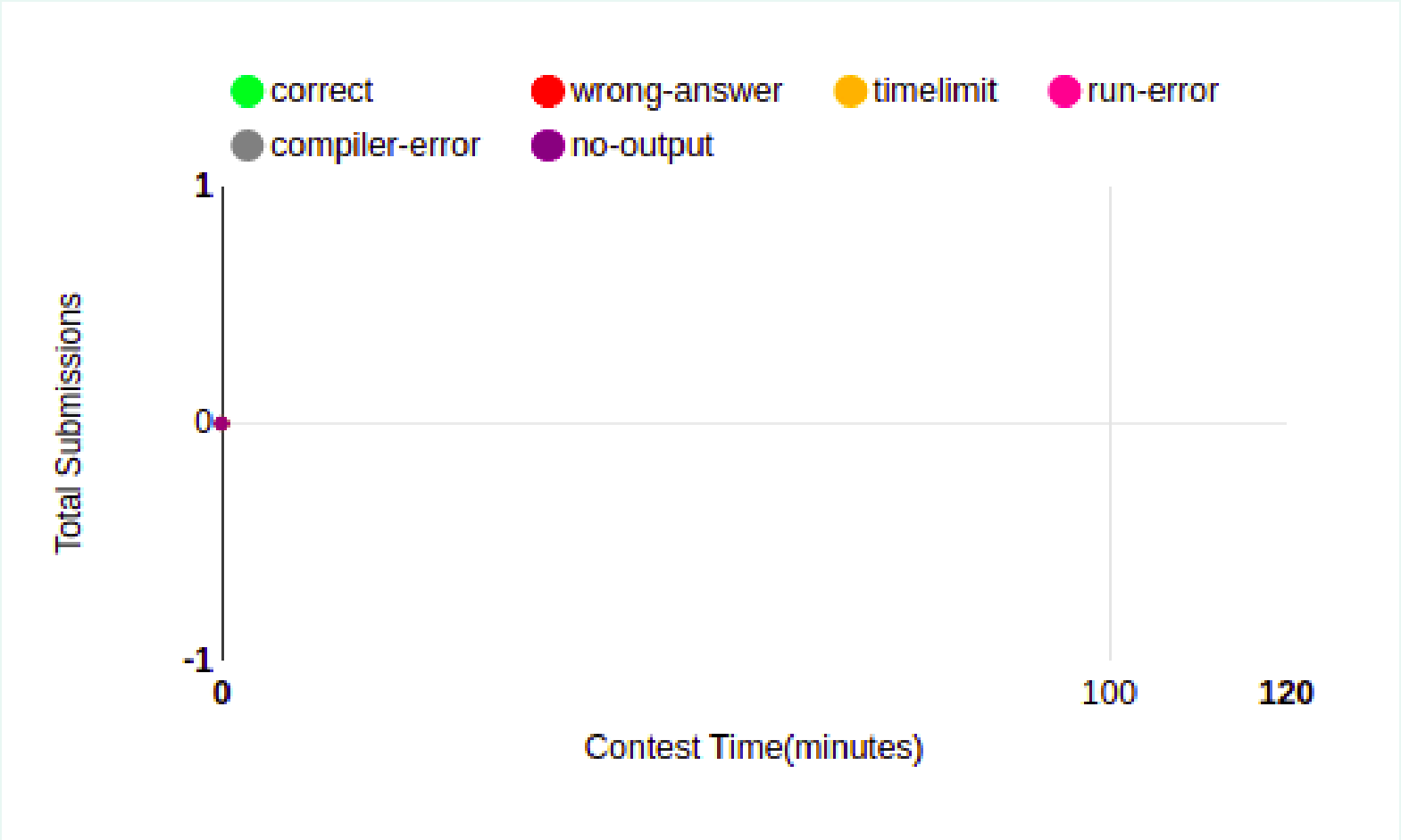
La solución era simularlo, lo ideal sin usar divisiones (aún así, la división se puso que fuese módulo 2, para que todos los valores fuesen pares y cualquier valor entre 2 fuese par).

```
long dinero = initial = D;
int venta = -1, compra = -1, V, valor;
bool vender;
Para cada valor que tenemos un histórico (V)
    Leer el valor
    Si i es 0
        compra = valor;
        vender = true;
    Si no comprobar compra*2<=price && vender
        dinero*=2; venta = valor; flag = !flag;

    Si no comprobar venta<=valor*2 && !vender //venta/2>=valor
        compra = valor;  vender = !vender;

Solución -> dinero-inicial
```

D. Dickie y las pizzas matemáticas

Envíos	Válidos	% éxito
0	0	0 %

D. Dickie y las pizzas matemáticas



D. Dickie y las pizzas matemáticas

Tl;dr: Este problema se soluciona con una mochila (programacion dinamica). El peso y la felicidad son funciones matematicas que deben ser encontradas. En principio, debemos deducir $h(x)$ y $w(x)$.

- h(x) es igual al x-esimo numero triangular + 1. Es decir $h(x) = x(x+1)/2+1$. Esto se puede demostrar luego de dibujar ejemplos pequeños (e.g. x = 0 => h(x) = 1, x = 1 => h(x) = 2...) y estableciendo una relacion de recurrencia para luego derivar la formula cerrada.
- w(x) es equivalente al x-esimo numero de Catalan. La formula para hallar el x-esimo numero de Catalan es: $cat(x) = (2x)!/(x!(x+1)!)$.
- Sin embargo, calcular factoriales puede ser "trickyütilizando aritmetica modular y tomando en cuenta que $2x$ puede ser mayor que 1009 (calcular un $n!\%p$ donde $p < n$ no es trivial).
- Hay una forma recursiva para calcular los numeros de Catalan: $C_{n+1} = \frac{2(2n+1)}{n+2} \times C_n$ partiendo de que $C_0 = 1$
- Otra alternativa para calcular el x-esimo numero de Catalan es: $cat(x) = \binom{2x}{x} - \binom{2x}{x+1}$. Dado que la funcion debe ser calculada con modulo 1009, es posible calcular estos numeros utilizando el triangulo de pascal para calcular coeficientes binomiales.

D. Dickie y las pizzas matemáticas

Por lo anterior podemos definir que:

- $h(x) = x(x+1)/2 \,\%1009$
- $w(x) = cat(x) \,\%1009$

D. Dickie y las pizzas matemáticas

Para calcular estas funciones vamos a necesitar utilizar propiedades de la aritmetica modular:

- $modSum(a,b,m) = (a+b)\,\%m = ((a\,\%m) + (b\,\%m))\,\%m$ - $O(1)$
- $modMult(a,b,m) = (a \times b)\,\%m = ((a\,\%m) \times (b\,\%m))\,\%m$ - $O(1)$
- $modPow(a,b,m) = modMult(a, modMult(a,...,m),...,m)$ - $O(b)$ Pero si se hace exponenciacion rapida termina siendo $O(log(b))$
- $modInv(a,m) = modPow(a, m-2)$ Esto se cumple siempre y cuando m sea primo por el pequeño teorema de Fermat (1009 es primo) - $O(log(m-2))$ Con exponenciacion rapida. Tambien se puede calcular la inversa modular de un numero utilizando el algoritmo de Euclides extendido.

D. Dickie y las pizzas matemáticas

Apoyandonos en las funciones anteriores podriamos calcular $h(x)$ y $w(x)$

- $(*)h(x) = modMult(modMult(x, x+1), modInv(2))$ - $O(1)$
- $(*)w(x) = modMult(modMult((4n+2), modInv(n+2)), w(x-1))$ y si $x == 0$ entonces $w(x) = 1$
- $(*)$: Omiti el parametro m del modulo por comodidad al escribir.

En este punto podemos calcular el peso y la felicidad individual de cada ingrediente y almacenarlo en un arreglo.
Luego de calcular el peso y la felicidad de cada ingrediente podemos aplicar la tecnica de la mochila para resolver el problema.
La complejidad total es $O(N + NlogN + N^2) = O(N^2)$.

D. Dickie y las pizzas matemáticas

```
// global scope
typedef long long ll;

const int MOD = int(1e3) + 9;
const int INF = int(1e9) + 100;

ll ww[1500], hh[1500], cat[1500];;
ll memo[1500][1500];
int n;
```

D. Dickie y las pizzas matemáticas

```
// main
int main() {
    int c;
    cin >> n >> c;
    ll x;
    for(int i = 0; i < n; i++) cin >> x, hh[i] = h(x);
    memset(cat, -1, sizeof(cat));
    cat[0] = 1;
    cat[1] = 1;
    for(int i = 0; i < n; i++) cin >> x, ww[i] = w(x);
    memset(memo, -1, sizeof(memo));
    ll ans = dp(0, c);
    cout << ans << endl;
    return 0;
}
```

D. Dickie y las pizzas matemáticas

```
// mochila
ll dp(int i, int c) {
    if (c < 0)
        return -INF;

    if (i == n)
        return 0;

    if (memo[i][c] != -1) {
        return memo[i][c];
    }

    memo[i][c] = max(dp(i+1, c), dp(i+1, c-ww[i]) + hh[i]);

    return memo[i][c];
}
```

D. Dickie y las pizzas matemáticas

```
// funciones h(x) y w(x)

int h(ll x) {
    ll num = mod_mult(x, x+1);
    ll den = mod_inv(2);
    ll tot = mod_mult(num, den);
    return mod_sum(tot, 1);
}
int w(ll x) {
    if (cat[x] != -1)
        return cat[x];
    ll num = 4*(x-1)+2;
    ll den = mod_inv(mod_sum(x, 1));
    ll prev = w(x-1);
    ll aux = mod_mult(num, den);
    return cat[x] = mod_mult(aux, prev);
}
```

D. Dickie y las pizzas matemáticas

```
// aritmetica modular - parte I
ll mod_sum(ll a, ll b) {
    return ((a%MOD) + (b%MOD)) % MOD;
}
ll mod_mult(ll a, ll b) {
    return ((a%MOD) * (b%MOD)) % MOD;
}
ll mod_pow(ll a, ll b) {
    ll ret = 1;
    ll y = a;
    while (b) {
        if (b&1) {
            ret = mod_mult(ret, y);
        }
        y = mod_mult(y, y);
        b >>= 1;
    }
    return ret;
}
```

D. Dickie y las pizzas matemáticas

```
// aritmetica modular - parte II
ll mod_inv(ll a) {
    return mod_pow(a, MOD-2);
}
```