



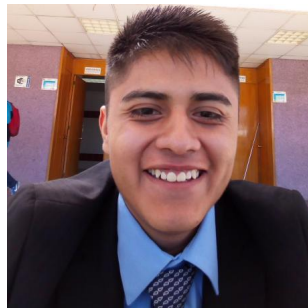
Tarea 06

Tecnica divide y venceras

Alumno:

López Manríquez Ángel

(2017630941)



Índice

1. Maximum subarray sum	3
1.1. Descripción	3
1.2. Código	3
1.3. Explicación	4
1.4. Validación del juez	5
2. Closest pair	5
2.1. Descripción	5
2.2. Código	5
2.3. Explicación	7
2.4. Validación del juez	8
3. Inversion count	8
3.1. Descripción	8
3.2. Código	8
3.3. Explicación	10
3.4. Validación del juez	10
4. Mega inversions	11
4.1. Descripción:	11
4.2. Código	11
4.3. Explicación	12
4.4. Validación del juez	13
5. Cobertura	13
5.1. Descripción	13

5.2. Codigo 13

5.3. Explicacion 14

5.4. Validacion del juez 15

1. Maximum subarray sum

1.1. Descripción

La tarea es simple, dado un arreglo A de números enteros debes imprimir cual es la suma máxima en cualquier subarreglo contiguo.

1.2. Código

```
1 // Angel Lopez Manriquez
2
3 #include <iostream>
4 #include <vector>
5
6 #include <math.h>
7 #include <stdint.h> // int64_t defined
8
9 using namespace std;
10
11 int64_t max_subarr_between(const vector<int64_t> &v, const long l, const long r,
12     const long mid) {
13     int64_t leftSum = INTMAX_MIN; // -infy
14     int64_t rightSum = INTMAX_MIN;
15     int64_t sum = 0;
16
17     // obtain the highest value from the left
18     for (int i = mid; i >= l; --i) {
19         sum += v[i];
20         if (sum > leftSum) leftSum = sum;
21     }
22
23     // obtain the highest value from the right
24     sum = 0;
25     for (int i = mid + 1; i <= r; ++i) {
26         sum += v[i];
27         if (sum > rightSum) rightSum = sum;
28     }
29
30     return leftSum + rightSum;
31 }
32
33 int64_t max_subarr_subseq(const vector<int64_t> &v, const long l, const long r) {
34     if (l == r) {
35         return v[l];
36     }
37
38     // compute the left subarray, right and the middle one and return the maximum
39     const long mid = (r - l) / 2 + 1;
40     const int64_t leftSum = max_subarr_subseq(v, l, mid);
41     const int64_t rightSum = max_subarr_subseq(v, mid + 1, r);
42     const int64_t midSum = max_subarr_between(v, l, r, mid);
43     return max(leftSum, max(rightSum, midSum));
```

```

44 }
45
46 inline int64_t max_subarr_subseq(vector<int64_t> &v) {
47     return max_subarr_subseq(v, 0, v.size() - 1); // begin the recurrence
48 }
49
50 vector<int64_t> fillVec() {
51     long n;
52     cin >> n;
53     vector<int64_t> v(n); // allocate a vector with n elements
54     for (int i = 0; i < n; ++i) cin >> v[i];
55     return v;
56 }
57
58 int main(void) {
59     vector<int64_t> v = fillVec(); // read the vector
60
61     int64_t ans = max_subarr_subseq(v);
62     cout << ans;
63     return 0;
64 }

```

1.3. Explicación

Dado un arreglo $A = \langle a_0, a_1, \dots, a_{n-1} \rangle$, este problema se resolvió con la técnica divide y vencerás (divide and conquer, en inglés) en donde, separamos al vector en dos mitades L, R y obtenemos el valor de estas recursivamente teniendo muy en cuenta que posiblemente la suma máxima se encuentre en el medio de estos dos subvectores, tomamos este caso aparte obteniendo la suma máxima empezando desde el medio hasta la izquierda y de manera analoga con la derecha, solo obtenemos la suma máxima de estas mitades y al final sumamos estas pues al estar concatenadas la suma forma parte de un subarreglo, claro, el caso base es cuando el arreglo A es tal que $|A| \leq 1$ pues es el caso más sencillo. Al final de la recurrencia encontramos el máximo entre sus costados y el medio.

Por lo anterior explicado la complejidad es $O(n \lg n)$.

1.4. Validación del juez

Entrada

8 -2 -5 6 -2 -3 1 5 -6	7
5 1 2 3 4 5	15

Salida

Límites

- $1 \leq N \leq 10^5$
- $|A_i| \leq 10^9$

Fuente: Filiberto Fuentes
 Problema subido por: galloska
[Reportar contenido inapropiado en este problema.](#)
[Calificar el problema](#)

Envíos

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2018-11-01 00:02:01	d46cbe2d	Respuesta correcta	100.00%	cpp11	3.57 MB	0.30 s	

2. Closest pair

2.1. Descripción

Te encuentras con un mapa del cúmulo de estrellas R136. En el mapa, cada estrella aparece como un punto ubicada en un plano cartesiano. Te asalta de pronto una pregunta, ¿cuál será la distancia mínima entre dos estrellas en el mapa?

2.2. Código

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <functional> // copy_if
5  #include <iomanip> // fixed, setprecision
6
7  #include <stdint.h>
8  #include <float.h>
9  #include <math.h>
10 #include <stdlib.h>
11
12 using namespace std;
```

```

13
14 typedef pair<double, double> double_pair;
15
16 inline double dist(const double_pair &p, const double_pair &q) {
17     return hypot((p.first - q.first), (p.second - q.second));
18 }
19
20 int cmp_y_axis(const double_pair &p, const double_pair &q) {
21     return p.second < q.second;
22 }
23
24 double min_dist_brute_force(const vector<double_pair> &pairs) {
25     double min = DBL_MAX;
26     const int n = pairs.size();
27     for (int i = 0; i < n; ++i) {
28         for (int j = i + 1; j < n; ++j) {
29             const double d = dist(pairs[i], pairs[j]);
30             if (d < min) min = d;
31         }
32     }
33     return min;
34 }
35
36 /**
37  * Obtains the minimum distance in the middle of the set of computed points.
38  * We know that if the distance is bigger than the minimum computed in the sides
39  * we don't even have to compute it.
40  */
41 double min_middle(const vector<double_pair> &xsorted, const double &offset, const double &d) {
42
43     double min = DBL_MAX;
44     vector<double_pair> center_pairs;
45     auto in_limit = [&](const double_pair p) { return abs(p.first - offset) <= d; };
46     copy_if(xsorted.begin(), xsorted.end(), back_inserter(center_pairs), in_limit); // it may
47     ↪ be optimized
48
49     const int n = center_pairs.size();
50     for (int i = 0; i < n; ++i)
51         for (int j = i + 1; j < n; ++j) {
52             if (abs(center_pairs[i].second - center_pairs[j].second) < d) {
53                 double curr_dist = dist(center_pairs[j], center_pairs[i]);
54                 if (curr_dist < min) min = curr_dist;
55             }
56         }
57
58     return min;
59 }
60
61 // always returns a positive value
62 double min_dist(const vector<double_pair> &xsort) {
63     const int n = xsort.size();
64
65     // apply brute force for this little case
66     if (n <= 3) return min_dist_brute_force(xsort);
67     const double_pair mid_pair = xsort[n / 2];

```

```

68     vector <double_pair> left;
69     vector <double_pair> right;
70
71     for (auto &p: xsort) {
72         if (p.first <= mid_pair.first) left.push_back(p);
73         else right.push_back(p);
74     }
75
76     const double min_left = min_dist(left), min_right = min_dist(right);
77     const double min_sides = min(min_left, min_right);
78     const double min_mid = min_middle(xsort, mid_pair.first, min_sides);
79
80     return min(min_sides, min_mid);
81 }
82
83 double get_min_dist(vector<double_pair> &pairs) {
84     sort(pairs.begin(), pairs.end()); // it'll sort by the first element by default
85
86     return min_dist(pairs);
87 }
88
89 void print_pairs(const vector<double_pair> &pairs) {
90     for (auto &p: pairs) cout << p.first << ", " << p.second << endl;
91 }
92
93 vector<double_pair> scanv() {
94     int n;
95     cin >> n;
96     vector<double_pair> pairs(n);
97     for (int i = 0; i < n; ++i) {
98         cin >> pairs[i].first >> pairs[i].second;
99     }
100     return pairs;
101 }
102
103 int main(int argc, char const *argv[]) {
104     vector<double_pair> pairs = scanv();
105     cout << fixed << setprecision(3);
106     cout << get_min_dist(pairs);
107     return 0;
108 }
109
110 // compilation:
111 // g++ -g closest_pair.cpp -std=c++11

```

2.3. Explicación

Esta pregunta es un clásico y, como todo programa, puede tener múltiples soluciones, aquí, dado un conjunto de pares ordenados $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$ primero procedemos a ordenarlos con respecto al primer par del eje x , luego nos concentramos en resolver el problema recursivamente para la primera mitad de puntos y para la siguiente mitad, obtenemos el mínimo de ambas junto con la franja que se encuentra en el medio de estas tomando en cuenta que esta franja debe abarcar un ancho tal que es

el mínimo de las soluciones anteriores pues no hay necesidad de buscar más allá por la desigualdad del triángulo. El caso base es cuando tenemos solo tres puntos, aquí procedemos a resolverlos mediante fuerza bruta. La implementación solo funciona si para toda $x \in \mathbf{R}$ le corresponde un único valor, afortunadamente en el juez no se presentó tal caso. La complejidad es $O(n \lg n)$.

2.4. Validación del juez

The screenshot shows the omegaUp website interface. The top navigation bar includes links for Arena, Concursos, Problemas, Rank, Escuelas, Blog, Preguntas, and a user profile for Angelino. The main content area displays the 'Límites' (Limits) for a problem, specifying constraints for n and X, Y . Below this, it lists the source as 'Alain' and the uploader as 'Alain_Acevedo'. A table titled 'Envíos' (Submissions) shows a submission from 2018-11-02 with a GUID of 40a3f13d, which was successful ('Respuesta correcta') with a 100.00% score, using C++11, 3.55 MB of memory, and 0.95 seconds of time.

3. Inversion count

3.1. Descripción

Dado un arreglo $A[0, \dots, n-1]$ de enteros positivos ($n \leq 2 \times 10^5$, $A[i] \leq 10^7$), definimos una *inversión* como un par ordenado (i, j) tal que $i < j$ y $A[i] > A[j]$; es decir, dos elementos distintos que se encuentran “desordenados”. El problema es hallar todas las inversiones en A .

3.2. Código

```

1  #include <iostream>
2  #include <vector>
3  #include <array>
4  #include <functional>
5  #include <stack>
6
7  #include <stdint.h>
8
9  using namespace std;
10
11 template<typename t>
12 int merge(vector<t> &dest, vector<t> &buff, const int left, const int mid, const int right) {
13     int i = left, j = mid, k = left, count = 0;

```

```
14     while ((i <= mid - 1) && (j <= right)) {
15         if (dest[i] <= dest[j]) buff[k] = dest[i++];
16         else {
17             buff[k] = dest[j++];
18             count += mid - i;
19         }
20         ++k;
21     }
22
23     while (i <= mid - 1) {
24         buff[k] = dest[i++];
25         ++k;
26     }
27
28     while (j <= right) {
29         buff[k] = dest[j++];
30         ++k;
31     }
32
33     for (i = left; i <= right; ++i) dest[i] = buff[i];
34     return count;
35 }
36
37 template<typename t>
38 int mergesort(vector<t> &v, vector<t> &buff, const int l, const int r) {
39     if (!(l < r)) return 0;
40     int mid = l + (r - l) / 2;
41     return mergesort(v, buff, l, mid)
42         + mergesort(v, buff, mid + 1, r)
43         + merge(v, buff, l, mid + 1, r);
44 }
45
46 template<typename t>
47 int mergesort(vector<t> &v) {
48     vector<t> buff(v.size());
49     return mergesort(v, buff, 0, v.size() - 1);
50 }
51
52 vector<int64_t> scanv() {
53     int n;
54     cin >> n;
55     vector<int64_t> v(n);
56     for (int i = 0; i < n; ++i)
57         cin >> v[i];
58     return v;
59 }
60
61 template<typename t>
62 void print(vector<t> v) {
63     cout << endl;
64     for (auto &x: v) cout << x << " ";
65     cout << endl;
66 }
67
68 int main(int argc, char const *argv[])
69 {
```

```

70 ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
71 int t;
72 cin >> t;
73
74 while (t--) {
75     vector<int64_t> v = scanv();
76     cout << mergesort(v) << endl;
77 }
78
79
80 return 0;
81
82 }

```

3.3. Explicacion

Para resolver el ejercicio usaremos el algoritmo *mergesort* modificado ligeramente, es bien sabido que el mergesort tiene una complejidad $O(n \lg n)$ por lo que su ejecucion es relativamente rapida. Basicamente, ordenamos como de costumbre solo que esta vez retornamos un entero el cual da el numero de inversiones que hay en el arreglo, cada que llamamos la funcion `merge` le pasamos los indices del arreglo i, j, m en el cual el numero de inversiones es representado por $\sum_i (m - i)$ cuando se encuentra a una inversion.

3.4. Validacion del juez

https://www.spoj.com/status/algoria/


PROBLEMS STATUS RANKS DISCUSS CONTESTS PROFILE

Profile / History of submissions

Not hidden submissions All submissions

: submissions

ID	DATE	PROBLEM	RESULT	TIME	MEM	LANG
22636287	2018-11-04 02:57:56	Inversion Count	accepted edit ideone it	0.06	16M	CPP14
22636279	2018-11-04 02:55:33	Inversion Count	wrong answer edit ideone it	0.05	17M	CPP14
22636254	2018-11-04 02:46:44	Inversion Count	wrong answer edit ideone it	0.12	17M	CPP14
19599214	2017-06-12 19:53:59	Cumulative Sum Query	accepted edit ideone it	0.02	16M	CPP14
19594143	2017-06-11 23:04:01	Cumulative Sum Query	accepted edit ideone it	2.35	16M	CPP14



All the tools your team needs in one place. Slack: Where work happens.

ADS VIA CARBON

4. Mega inversions

4.1. Descripción:

Dado un arreglo $A[0, \dots, n-1]$ de enteros positivos ($n \leq 10^5$, $A[i] \leq n$), definimos una *mega inversión* como una terna ordenada (i, j, k) tal que $i < j < k$ y $A[i] > A[j] > A[k]$. El problema es hallar todas las mega inversiones en A .

4.2. Código

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  #include <stdint.h>
6
7  using namespace std;
8
9  template<typename T>
10 class FenwickTree {
11 private:
12     int n;
13     vector<T> bit;
14
15 public:
16     FenwickTree(int N) {
17         this->n = N;
18         bit.assign(N, 0);
19     }
20
21     void update(int pos, T value) {
22         while(pos < n) {
23             bit[pos] += value;
24             pos |= pos + 1;
25         }
26     }
27
28     T query(int r) {
29         T res = 0;
30         while (r >= 0) {
31             res += bit[r];
32             r = (r & (r + 1)) - 1;
33         }
34         return res;
35     }
36
37     T query(int l, int r) {
38         return query(r) - query(l - 1);
39     }
40 };
41
42 int64_t megaInversions(vector<int> & v) {

```

```

43     int64_t count = 0;
44     int maximum = *max_element(v.begin(), v.end());
45     FenwickTree<int64_t> bit1(maximum + 1);
46     FenwickTree<int64_t> bit2(maximum + 1);
47     for (int & a: v) {
48         bit1.update(a, 1);
49         bit2.update(a, bit1.query(a + 1, maximum));
50         count += bit2.query(a + 1, maximum);
51     }
52     return count;
53 }
54
55 int main() {
56     int n;
57     cin >> n;
58     vector<int> v(n);
59     for (int i = 0; i < n; ++i) cin >> v[i];
60     cout << megaInversions(v) << "\n";
61     return 0;
62 }

```

4.3. Explicación

Usamos dos Fenwick Tree B_1 y B_2 inicializados en cero. ¿Que es un Fenwick tree? Es una estructura de dato tipo arbol T la cual, dado un arreglo A es capaz de obtener $\sum_{i=a}^b a_i$, es decir, la "función acumulada" de A de manera bastante rápida, a diferencia de un arbol de segmentos o usar barridos, en fin, ahora ya explicado el Fenwick Tree tenemos que iterar sobre los elementos de A y definimos:

1. $B_1[a_i]$ es el número de veces que ha aparecido el elemento a_i hasta la i -ésima iteración. Tal y como lo hicimos en el problema anterior.
2. Por lo tanto, $\sum_{j=a_i+1}^n B_1[j]$ nos dirá cuántos elementos a_j a la derecha de a_i son mayores a a_i , es decir, los elementos a_j tal que $i < j$ y $A[i] > A[j]$.

Para el análisis, sea $n = |A|$, así, la construcción del arbol cuesta $O(n)$ tenemos un bucle que incrementa en una unidad en donde actualizamos y consultamos el arbol, cada una de estas operaciones cuesta $\lg n$, por lo que la función `megaInversion` cuesta $O(n + n(4 \lg n)) = O(n \lg n)$.

4.4. Validacion del juez

judge status

<
Previous
1
2
3
4
5
Next
>

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
22663636	2018-11-08 03:02:19	ang3lino	Mega Inversions	accepted edit ideone it	0.06	20M	CPP14
22663631	2018-11-08 03:01:41	supernanren	Matrix Inverse	accepted	0.14	16M	CPP14
22663627	2018-11-08 03:00:17	supernanren	Matrix Inverse	accepted	0.12	16M	CPP14

5. Cobertura

5.1. Descripción

Considera una pieza con forma de L formada por 3 cuadros adyacentes.

Se requiere cubrir una cuadrícula con piezas como la antes descrita dejando un solo cuadro de 1×1 vacío. La cuadrícula consta de 2^n filas (numeradas de 1 a 2^n) por 2^n columnas (numeradas también de 1 a 2^n).

Para cubrir la cuadrícula no es válido colocar una pieza sobre otra ni dejar una o más piezas parcialmente fuera de la cuadrícula. Es posible comprobar que toda cuadrícula de dichas dimensiones puede ser cubierta siguiendo las restricciones anteriores sin importar donde se encuentre el espacio vacío.

5.2. Código

```

1  #include <iostream>
2  using namespace std;
3
4  void solve(int r0, int r1, int c0, int c1, int x, int y) {
5      if (r1 - r0 == 1 && c1 - c0 == 1) { // base case
6          if (!(r0 == x && c0 == y)) cout << r0 << " " << c0 << " ";
7          if (!(r0 == x && c1 == y)) cout << r0 << " " << c1 << " ";
8          if (!(r1 == x && c0 == y)) cout << r1 << " " << c0 << " ";
9          if (!(r1 == x && c1 == y)) cout << r1 << " " << c1 << " ";
10         cout << "\n";
11     } else { // inductive case
12         int mid_r = (r0 + r1) / 2;
13         int mid_c = (c0 + c1) / 2;
14         if (r0 <= x && x <= mid_r && c0 <= y && y <= mid_c) {
15             //first quadrant
16             cout << mid_r << " " << mid_c + 1 << " ";

```

```

17     cout << mid_r + 1 << " " << mid_c << " ";
18     cout << mid_r + 1 << " " << mid_c + 1 << "\n";
19     solve(r0, mid_r, c0, mid_c, x, y);
20     solve(r0, mid_r, mid_c + 1, c1, mid_r, mid_c + 1);
21     solve(mid_r + 1, r1, c0, mid_c, mid_r + 1, mid_c);
22     solve(mid_r + 1, r1, mid_c + 1, c1, mid_r + 1, mid_c + 1);
23 } else if (r0 <= x && x <= mid_r && mid_c + 1 <= y && y <= c1) {
24     //second quadrant
25     cout << mid_r << " " << mid_c << " ";
26     cout << mid_r + 1 << " " << mid_c << " ";
27     cout << mid_r + 1 << " " << mid_c + 1 << "\n";
28     solve(r0, mid_r, c0, mid_c, mid_r, mid_c);
29     solve(r0, mid_r, mid_c + 1, c1, x, y);
30     solve(mid_r + 1, r1, c0, mid_c, mid_r + 1, mid_c);
31     solve(mid_r + 1, r1, mid_c + 1, c1, mid_r + 1, mid_c + 1);
32 } else if (mid_r + 1 <= x && x <= r1 && c0 <= y && y <= mid_c) {
33     //third quadrant
34     cout << mid_r << " " << mid_c << " ";
35     cout << mid_r << " " << mid_c + 1 << " ";
36     cout << mid_r + 1 << " " << mid_c + 1 << "\n";
37     solve(r0, mid_r, c0, mid_c, mid_r, mid_c);
38     solve(r0, mid_r, mid_c + 1, c1, mid_r, mid_c + 1);
39     solve(mid_r + 1, r1, c0, mid_c, x, y);
40     solve(mid_r + 1, r1, mid_c + 1, c1, mid_r + 1, mid_c + 1);
41 } else if (mid_r + 1 <= x && x <= r1 && mid_c + 1 <= y && y <= c1) {
42     //fourth quadrant
43     cout << mid_r << " " << mid_c << " ";
44     cout << mid_r << " " << mid_c + 1 << " ";
45     cout << mid_r + 1 << " " << mid_c << "\n";
46     solve(r0, mid_r, c0, mid_c, mid_r, mid_c);
47     solve(r0, mid_r, mid_c + 1, c1, mid_r, mid_c + 1);
48     solve(mid_r + 1, r1, c0, mid_c, mid_r + 1, mid_c);
49     solve(mid_r + 1, r1, mid_c + 1, c1, x, y);
50 }
51 }
52 }
53
54 int main() {
55     int n, x, y;
56     cin >> n >> x >> y;
57     solve(1, 1 << n, 1, 1 << n, x, y);
58     return 0;
59 }

```

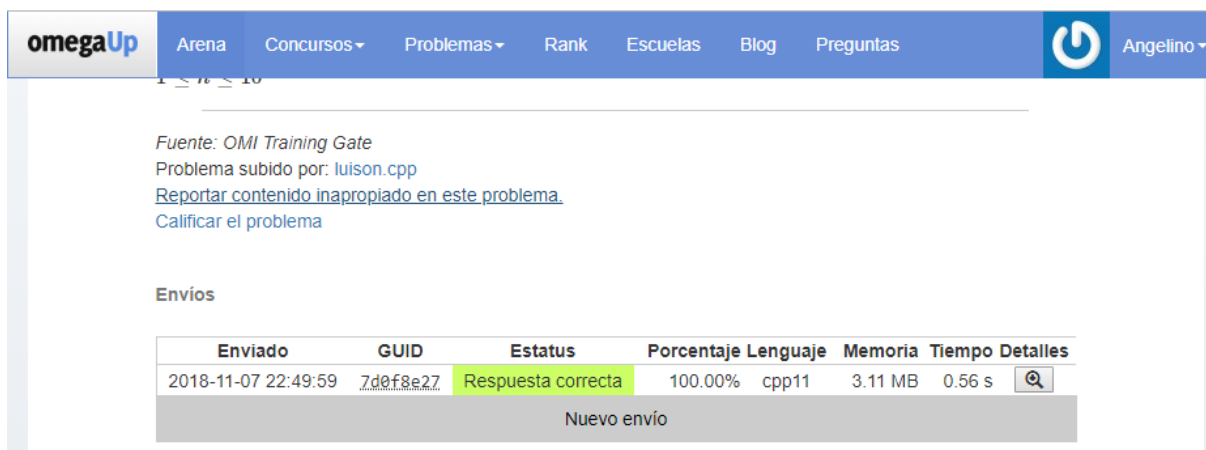
5.3. Explicacion

Para la implementacion del programa se tuvo que hacer uso de la demostracion de un lema, el lema es la descripcion del problema el cual asegura que existe una configuracion tal que un cuadrado de dimension de potencias de dos puede ser llenado usando piezas en forma de L (se llama triomino).

Prueba del paso inductivo: Para $n = k$, tenemos un tablero de ajedrez de tamaño $2^k \times 2^k$, con un azulejo de la esquina eliminado. Podemos dividir este tablero en 4 partes, tres de las cuales son tableros de

ajedrez completos de tamaño $2^{k-1} \times 2^{k-1}$, y uno del mismo tamaño con un azulejo de esquina eliminado. El tablero con el azulejo de la esquina eliminado se puede cubrir con triominos, utilizando la hipótesis inductiva. Los otros tres tableros pueden tener todos los mosaicos, excepto un azulejo de esquina, cada uno cubierto por triominos, nuevamente utilizando la hipótesis inductiva. Ahora tenemos tres tablas, de tamaño $2^{k-1} \times 2^{k-1}$, cada uno con un azulejo de esquina descubierto. Podemos organizar las tres tablas para que las tres baldosas descubiertas se unan para formar una vacante con forma de triomino, y cubrirlo utilizando otro triomino. Entonces, hemos probado que si la declaración se mantiene para $n = k - 1$, también se mantiene para $n = k$. Por lo tanto, la afirmación está probada para todos n usando inducción. La complejidad es $O(2^{2n})$, lo cual explica el porque la máxima n que se nos puede dar es 10.

5.4. Validación del juez



The screenshot shows the omegaUp interface. At the top is a navigation bar with links: Arena, Concursos, Problemas, Rank, Escuelas, Blog, Preguntas, and a user profile for Angelino. Below the navigation bar, the problem title is partially visible as $1 \leq n \leq 10$. The source is cited as "Fuente: OMI Training Gate" and the problem was uploaded by "luison.cpp". There are links to "Reportar contenido inapropiado en este problema" and "Calificar el problema".

Under the "Envíos" (Submissions) section, a table displays the submission details:

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2018-11-07 22:49:59	7d0f8e27	Respuesta correcta	100.00%	cpp11	3.11 MB	0.56 s	

Below the table, there is a button labeled "Nuevo envío" (New submission).