The editorials

By Mourad NOUAILI

https://www.facebook.com/NOUAILI.Mourad

https://twitter.com/NOUAILI_Mourad

https://github.com/Mourad-NOUAILI

Certainly, you can find other solutions. But all solutions have the same principles.

Problem #1: Les clips

The videos never lap iif, $b \leq c$ or $d \leq a$. otherwise, we get a lap.

Code C++

```
//Les clips
/*
 * Constest: TOP2018 - Pre-selection round #2
 * Subject: Les clips
 * Lang: C++11
 * Submission result: Accepted
 *Time complexity: O(1)
*/
#include <bits/stdc++.h>

using namespace std;

int main() {

  int a, b, c, d;
  cin >> a >> b >> c >> d;

  (b <= c  || d <= a ) ? cout << "NO" : cout << "YES";

 return 0;
}
```

Submission of Les clips

View source
Resubmit

Execution Results

Test case #1: AC [0.012s, 1.23 MB] (10/10)
Test case #2: AC [0.015s, 1.23 MB] (10/10)
Test case #3: AC [0.013s, 1.23 MB] (10/10)
Test case #4: AC [0.013s, 1.23 MB] (10/10)
Test case #5: AC [0.009s, 1.23 MB] (10/10)
Test case #6: AC [0.013s, 1.23 MB] (10/10)
Test case #7: AC [0.012s, 1.23 MB] (10/10)
Test case #8: AC [0.011s, 1.23 MB] (10/10)
Test case #9: AC [0.008s, 1.23 MB] (10/10)
Test case #10: AC [0.012s, 1.23 MB] (10/10)

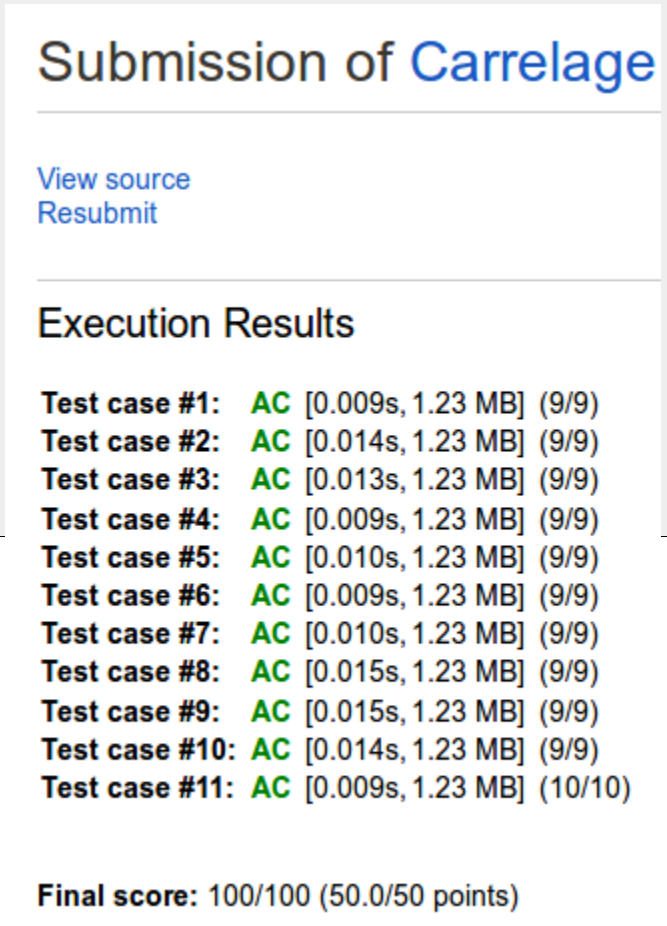
Final score: 100/100 (50.0/50 points)

Problem #2: Carrelage

To know the number of tiles to put on the surface of the room. Just multiply the height of the room times its width and divide all over the square of tile's side.

Code C++

```
//Carrelage
/*
 * Constest: TOP2018 - Pre-selection round #2
 * Subject: Carrelage
 * Lang: C++11
 * Submission result: Accepted
 *Time complexity: O(1)
*/
#include <bits/stdc++.h>

using namespace std;

int main() {

  int h, w;
  cin >> h >> w;

  int side;
  cin >> side;

  cout << (h / side) * (w / side) << '\n';

 return 0;
}
```



Submission of Carrelage

View source
Resubmit

Execution Results

Test case #1:	**AC**	[0.009s, 1.23 MB]	(9/9)
Test case #2:	**AC**	[0.014s, 1.23 MB]	(9/9)
Test case #3:	**AC**	[0.013s, 1.23 MB]	(9/9)
Test case #4:	**AC**	[0.009s, 1.23 MB]	(9/9)
Test case #5:	**AC**	[0.010s, 1.23 MB]	(9/9)
Test case #6:	**AC**	[0.009s, 1.23 MB]	(9/9)
Test case #7:	**AC**	[0.010s, 1.23 MB]	(9/9)
Test case #8:	**AC**	[0.015s, 1.23 MB]	(9/9)
Test case #9:	**AC**	[0.015s, 1.23 MB]	(9/9)
Test case #10:	**AC**	[0.014s, 1.23 MB]	(9/9)
Test case #11:	**AC**	[0.009s, 1.23 MB]	(10/10)

Final score: 100/100 (50.0/50 points)

Problem #3: Le chiffre de la chance

The number to reduce is very big, so we have to read it as a string.

I wrote two function, a function to reduce the input string. The other to reduce a number.

Code C++

```
//Le chiffre de la chance
/*
 * Constest: TOP2018 - Pre-selection round #2
 * Subject: Le chiffre de la chance
 * Lang: C++11
 * Submission result: Accepted
 *Time complexity: O(m*n)
*/
#include <bits/stdc++.h>

using namespace std;


int reduce1(string n) {
  int s = 0;
  int len = n.size();
  for (int i = 0 ; i < len ; ++i) {
    int d = n[i] - '0';
    s += d;
  }

  return s;
}

int reduce2(int n) {
  int s = 0;
   while (n != 0) {
      s += n % 10;
      n /= 10;
    }
   return s;
}
```

```
int main() {

 int m;
 cin >> m;

 while (m--) {
  string x;
  cin >> x;

  int s = reduce1(x);


  do {
    s = reduce2 (s);
  }while (s >= 10);



  cout << s << '\n';

 }

return 0;
}
```

Submission of Le Chiffre de la chance

View source
Resubmit

Execution Results

Test case #1: **AC** [0.008s, 1.23 MB] (1/1)

Final score: 1/1 (100.0/100 points)

Problem #4: L’extraterrestre

The goal is to find the exact pressure in 31 tries. Such as the pressure ∈ [1, 2.10^9].

The only information that we have:

- if we give a pressure greater than the one we search, the computer ship answer will be "FLOATS".
- if we give a pressure lower than the one we search, the computer ship answer will be "SINKS".

We have to write a program that search automatically the adequate pressure.

How our code must work ?

To simplify, let search a the value 13 in set of values [1, 20].

Naive approach

The algorithm will pick all the values until find the value that we search. In the worst case we an pick them all. The time complexity is linear: O(n).

Faster approach

Our program must work smart. Such as, we search a value in [1, 20], which mean we search a value in an ordered set of value, we can do a **binary search**.

```
Repeat
        Pick the middle
        if the middle is greater than the searched value then lower bound will come the middle + 1
        else upper bound will come the middle – 1
until (the middle = searched value or upper bound < lower bound)
```

Time complexity = O(log N)

example:

First our program pick the middle of the set (1 + 20) / 2 = 10, which is lower that 13.

so, our code pick the middle of [11, 20] = 15 > 13

then the middle of [11, 15] = 13 = 13.

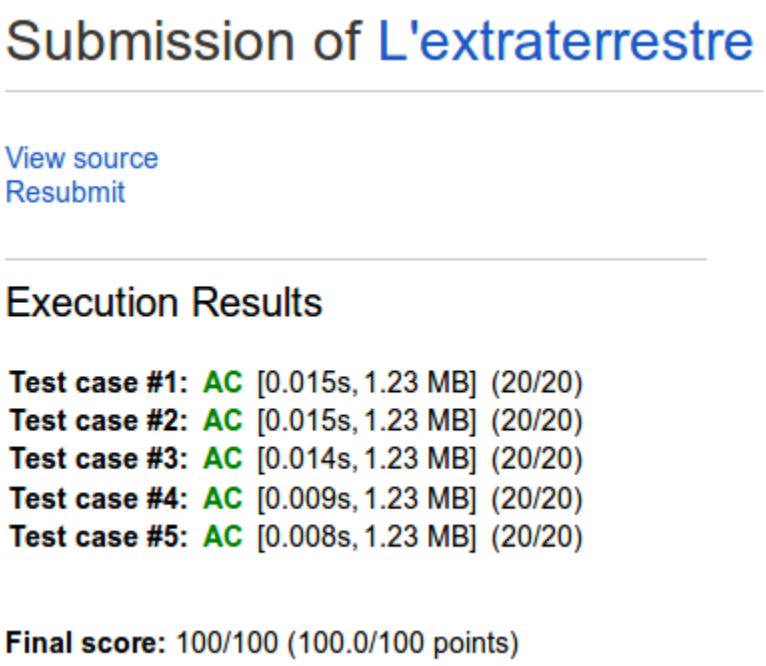
In our problem we have always a solution that we must found in 31 tries or less.

The code will be:

Code C++

```
//L'extraterrestre
/*
 * Constest: TOP2018 - Pre-selection round #2
 * Subject: L’extraterrestre
 * Lang: C++11
 * Submission result: Accepted
 *Time complexity: O(log n)
*/
#include <bits/stdc++.h>

using namespace std;

const int MAX_N = 31;
const long long MAX_PRESSURE = 2e9;

set<long long> hasChosen;

int main() {
  ios::sync_with_stdio(false);

  srand(time(NULL));

  int t = 1;
  long long up = MAX_PRESSURE;
  long long down = 1;
  while (t++ <= 31) {

    long long m = (up + down) / 2;

    cout << m << '\n';

    string ans;
    cin >> ans;

    if (ans.compare("FLOATS") == 0)
      up = m - 1;

    if (ans.compare("SINKS") == 0)
      down = m + 1;
```

```
    if (ans.compare("OK") == 0)
      break;
  }

  return 0;
}
```



Submission of L'extraterrestre

View source
Resubmit

Execution Results

Test case #1: **AC** [0.015s, 1.23 MB] (20/20)
Test case #2: **AC** [0.015s, 1.23 MB] (20/20)
Test case #3: **AC** [0.014s, 1.23 MB] (20/20)
Test case #4: **AC** [0.009s, 1.23 MB] (20/20)
Test case #5: **AC** [0.008s, 1.23 MB] (20/20)

Final score: 100/100 (100.0/100 points)

Problem #5: jeu de cartes

Our goal is to compute the maximum product of a list of integers as fast as possible. Which means in linear time.

- If we have one card, take it.

Example #1

INPUT
1
-5
OUTPUT
-5

Example #2

INPUT
1
5
OUTPUT
5

Example #3

INPUT
1
0
OUTPUT
0

- If there is a zeros and one negative integer, take the zero.

Example #1

INPUT
3
0 -5 0
OUTPUT
0

- If there is only positive integers, take them all

Example #1

INPUT
3
2 2 3
OUTPUT
12

- If there is one positive integer and one negative integer, take the positive one.

Example #1

INPUT
2
-5 6
OUTPUT
6

- If we count an odd number of negative integers, take them all except the nearest one to zero. And take also all positive integers.

Example #1

INPUT
10
-5 -3 -4 -2 -10 2 3 4 0 0
OUTPUT
14400

- If there is a even number of negative integers, take them with all positive integers.

Example #1

INPUT
9
-5 -3 -4 -2 2 3 4 0 0
OUTPUT
1440

C++ code: O(n)

```
//Jeu de cartes
/*
 * Constest: TOP2018 - Pre-selection round #2
 * Subject: Jeu de cartes
 * Lang: C++11
* Submission result: Accepted
*Time complexity: O(n)
*/
#include <bits/stdc++.h>

using namespace std;

multiset <int> all, allPositive, allNegative;
set <int> zero;

int main() {
  int n;
  cin >> n;
```

```
for (int i = 0 ; i < n ; ++i) {
  int val;
  cin >> val;
  if (val == 0) zero.insert(val);
  if (val > 0) allPositive.insert(val);
  if (val < 0) allNegative.insert(val);
}

all.insert(allPositive.begin(), allPositive.end());
all.insert(allNegative.begin(), allNegative.end());


if (n == 1) {
  cout << *all.begin() << '\n';
  return 0;
}

if (allNegative.size() == 0 && allPositive.size() == 0 && zero.size() == 1) {
  cout << "0\n";
  return 0;
}

if (zero.size() == 1 && allNegative.size() == 1 && allPositive.size() == 0) {
  cout << "0\n";
  return 0;
}

if (allNegative.size() == 1 && allPositive.size() == 1) {
  cout << *allPositive.begin() << '\n';
  return 0;
}
```

```
 if (allNegative.size() > 1 && allNegative.size() % 2 != 0) {
   int p = 1;
   auto it = allNegative.end();
   --it;
   allNegative.erase(it);

   for(auto e: allNegative)
     p *= e;

   for(auto e: allPositive)
     p *= e;

   cout << p << '\n';
   return 0;
 }

 if (allNegative.size() > 1 && allNegative.size() % 2 == 0) {
   int p = 1;
   for(auto e: all)
     p *= e;

   cout << p << '\n';
   return 0;
 }

 if (allNegative.size() == 0 && allPositive.size() >= 1) {
   int p = 1;
   for(auto e: allPositive)
     p *= e;

   cout << p << '\n';

 }

return 0;
}
```

Submission of Le jeu de cartes

View source
Resubmit

Execution Results

Test case #1: AC [0.013s, 1.23 MB] (10/10)
Test case #2: AC [0.014s, 1.23 MB] (10/10)
Test case #3: AC [0.014s, 1.23 MB] (10/10)
Test case #4: AC [0.014s, 1.23 MB] (10/10)
Test case #5: AC [0.015s, 1.23 MB] (10/10)
Test case #6: AC [0.008s, 1.23 MB] (10/10)
Test case #7: AC [0.014s, 1.23 MB] (10/10)
Test case #8: AC [0.014s, 1.23 MB] (10/10)
Test case #9: AC [0.013s, 1.23 MB] (10/10)
Test case #10: AC [0.010s, 1.23 MB] (10/10)

Final score: 100/100 (100.0/100 points)

Problem #6: Le chemin

It’s a graph theory problem.

Houses are vertices and roads are edges. We search if there is a path from A to B. To find that we can do a DFS.

See my write up of graph theory:

https://github.com/Mourad-NOUAILI/Graph-theory

MIT OCW:

https://www.youtube.com/watch?v=s-CYnVz-uh4&index=13&list=PLUl4u3cNGP61Oq3tWYp6V_F-5jb5L2iHb

```
//Le chemin
/*
 * Constest: TOP2018 - Pre-selection round #2
 * Subject: Le chemin
 * Lang: C++11
* Submission result: Accepted
*Time complexity: θ(n+m)
*/
#include <bits/stdc++.h>
using namespace std;

int n, m, a, b;
list<int> *adjList = NULL;
```

```
void createAdjList(int nbEdges) {
  for (int edge = 0 ; edge < nbEdges ; ++edge) {
    int vertex1 , vertex2;
    cin >> vertex1 >> vertex2;

    vertex1--;
    vertex2--;

    adjList[vertex1].push_back(vertex2);
    adjList[vertex2].push_back(vertex1);
  }
}

void dfs_visit(int v, bool visited[]){
  visited[v] = true;

  if (v == b) {
    printf("GO SHAHIR!\n");
    exit(0);
  }

  list<int>::iterator i;
  for (i = adjList[v].begin(); i != adjList[v].end(); ++i)
    if (!visited[*i])
      dfs_visit(*i, visited);
}

void dfs(int v){
  bool *visited = new bool[n];
  for (int i = 0; i < n; ++i)
    visited[i] = false;

  dfs_visit(v, visited);
}
```

```
int main() {
 ios::sync_with_stdio(false);

 cin >> n >> m >> a >> b;

 a--;
 b--;

 adjList = new list<int>[n];

 createAdjList(m);

 dfs(a);

 printf("NO SHAHIR!\n");

 return 0;
}
```

Submission of Le chemin

View source
Resubmit

Execution Results

Test case #1:	**AC**	[0.012s, 1.50 MB]	(10/10)
Test case #2:	**AC**	[0.005s, 1.50 MB]	(10/10)
Test case #3:	**AC**	[0.005s, 1.50 MB]	(10/10)
Test case #4:	**AC**	[0.005s, 1.50 MB]	(10/10)
Test case #5:	**AC**	[0.005s, 1.50 MB]	(10/10)
Test case #6:	**AC**	[0.007s, 1.50 MB]	(10/10)
Test case #7:	**AC**	[0.005s, 2.98 MB]	(10/10)
Test case #8:	**AC**	[0.006s, 2.98 MB]	(10/10)
Test case #9:	**AC**	[0.008s, 2.98 MB]	(10/10)
Test case #10:	**AC**	[0.005s, 2.98 MB]	(10/10)

Final score: 100/100 (100.0/100 points)

Problem #7: Le chocolat

It’s a D.P. (Dynamic Programming) problem.

Let’s call $F(N)$ that return the maximum number of given numbers$(X, Y and Z)$ that could make the sum of a given number N.

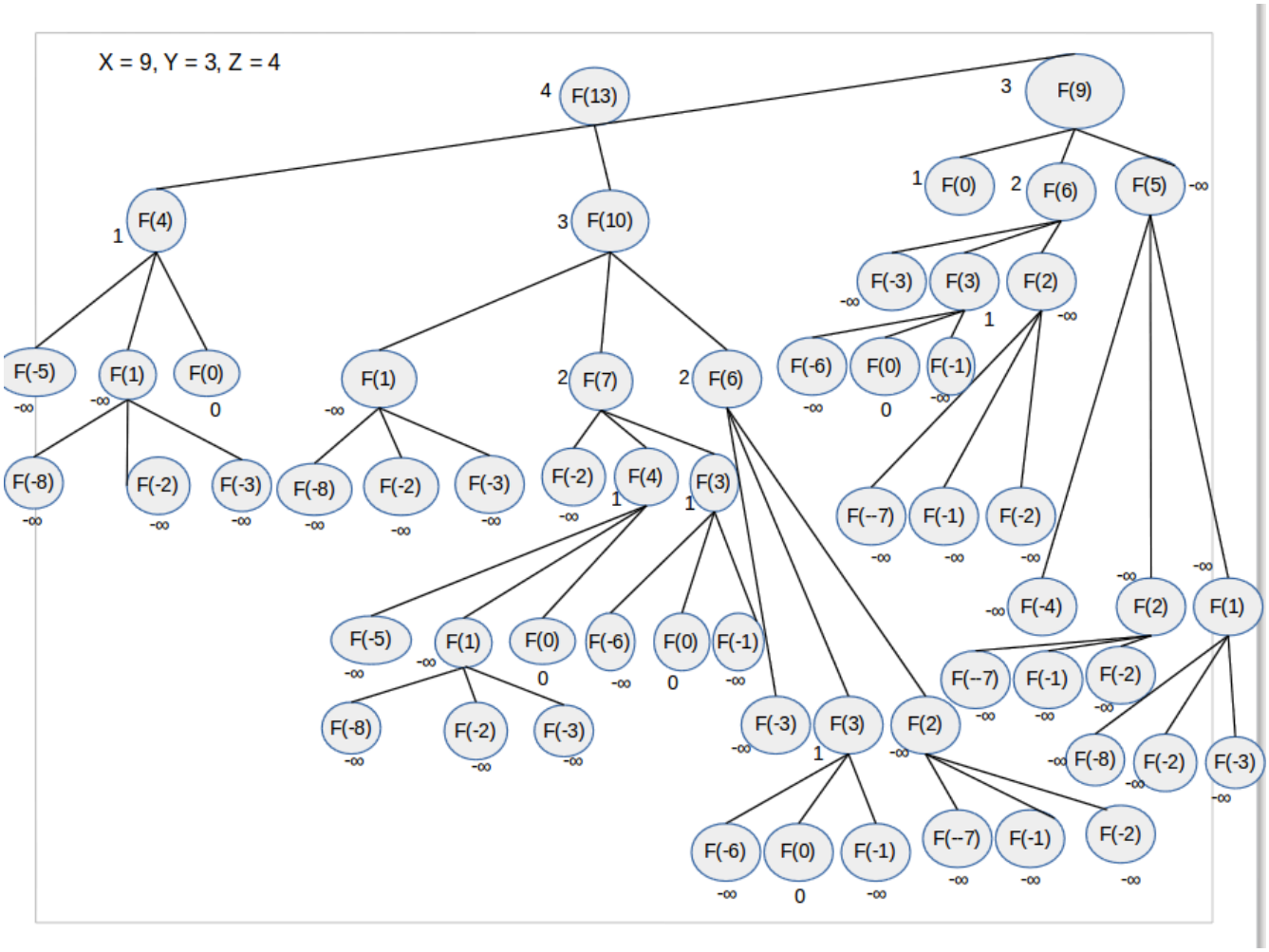
for example:

$N = 13, X = 9, Y = 3 \; and \; Z = 4.$

$13 = 4 + 3 + 3 + 3 ==> \quad F(13) = 4$

$$F(N) = \begin{cases} 0, if \; N = 0 \\ -\infty, if \; N < 0 \\ max(F(N-X), F(N-Y), F(N-Z)) + 1, otherwise \end{cases}$$

Naive recursive algorithm



As you see, the computation of the same value could be done many times. That cause the TLE.

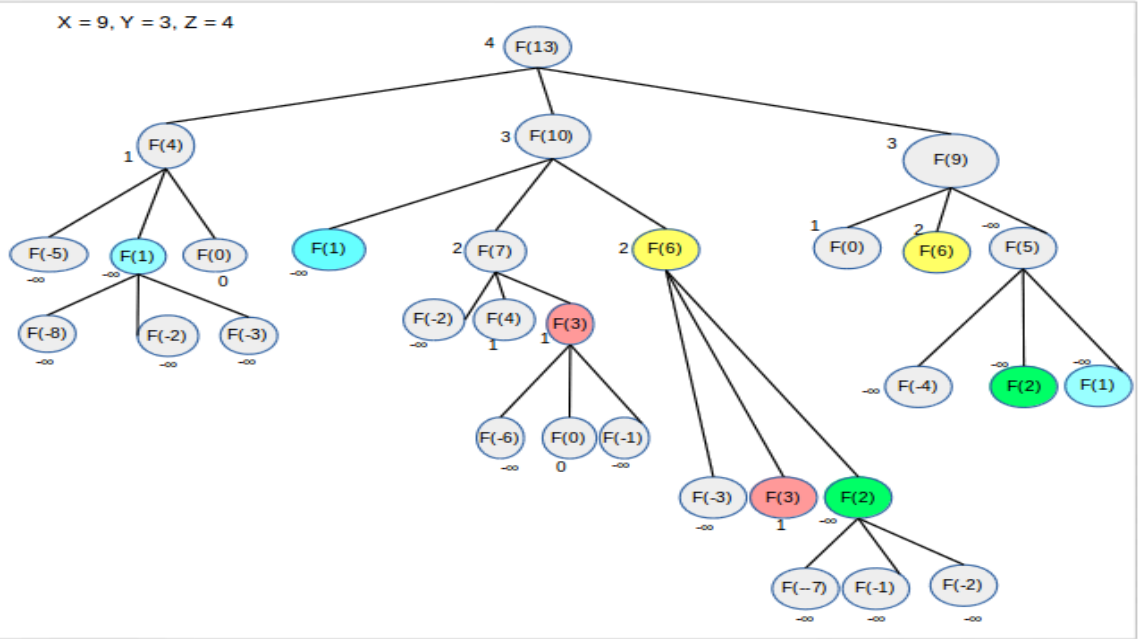
```
//Le chocolat
/*
 * Constest: TOP2018 - Pre-selection round #2
 * Subject: Le chocolat
 * Lang: C++11
* Submission result: TLE (recursion without memoization)
* Time complexity:  θ(2^(n/3))
*/
#include <bits/stdc++.h>
using namespace std;

const int MAX_N = 31;
const int INF = -1e9;

map<int, int> memo;

int x, y, z;
```

```
//Naive recursion
int f(int n) {
  if (n == 0) return 0;
  if (n < 0) return INF;
  return max(max(f(n-x), f(n-y)), f(n-z)) + 1;
}
```

```
int main() {
  ios::sync_with_stdio(false);

  int n;
  cin >> n;
  cin >> x >> y >> z;

  int ans = f(n);
    cout << ans << '\n';

  return 0;
}
```

Memoized recursive algorithm

The idea is, once F of some value was computed, there is no need to recomputed one more time. Just store it and use it: it's memoization.



Reference to DP memoization (MIT OCW):

https://www.youtube.com/watch?v=OQ5jsbhAv_M&list=PLUl4u3cNGP61Oq3tWYp6V_F-5jb5L2iHb&index=19

- f(k) only recurses first time it's called, ∀ k.
- memoized calls cost θ(1)
- # nonmemoized calls is n (in worst case): f(n), f(n-1), f(n-2), .., f(0), f(<0)
- Non recursive work coast: θ(1)

=> Time complexity is θ(n)

```
//Le chocolat
/*
 * Constest: TOP2018 - Pre-selection round #2.
 * Subject: Le chocolat (with memoization)
 * Lang: C++11.
* Submission result: Accepted.
* Time complexity:  θ(n)
*/
#include <bits/stdc++.h>

using namespace std;

const int INF = -1e9;

map<int, int> memo;

int x, y, z, r;
```

```
int f(int n)  {
 auto it = memo.find(n);
  if (it != memo.end())
      return memo[n];

  if (n == 0) r = 0;
  else if (n < 0) r = INF;
  else r =  max(max(f(n-x), f(n-y)), f(n-z)) + 1;

  memo.insert(pair<int, int>(n, r));

  return r;
}
```

```
int main() {
  ios::sync_with_stdio(false);

  int n;
  cin >> n;
  cin >> x >> y >> z;

  int ans = f(n);

  cout << ans << '\n';

  return 0;
}
```

Submission of Le chocolat

View source
Resubmit

Execution Results

Test case #1: AC [0.017s, 2.26 MB] (20/20)
Test case #2: AC [0.008s, 1.75 MB] (20/20)
Test case #3: AC [0.028s, 3.29 MB] (20/20)
Test case #4: AC [0.007s, 1.23 MB] (20/20)
Test case #5: AC [0.009s, 1.23 MB] (20/20)

Final score: 100/100 (100.0/100 points)