PRÁCTICA IV: BACKTRACKING

By:Elena Merelo

```cpp
#include <iostream>
#include <stdlib.h> //srand, rand
#include <time.h> //time
#include <vector>
#include <algorithm> //count

using namespace std;

class matriz_de_conveniencia{
private:
  vector<vector<int> > m;
  vector<int> solucion;

public:
  //Crea una matriz n*n con enteros aleatorios y la diagonal principal rellena de ceros
  matriz_de_conveniencia(int n){
    int i, j;
    //Inicializamos la semilla para generar los números aleatorios
    srand(time(NULL));

    m.resize(n);

    for(i= 0; i< n; i++)
      m[i].resize(n);

    for(i= 0; i< n ; i++)
      for(j= 0; j< n; j++)
        m[i][j]= i != j? rand() % 100: 0; //el número generado es un entero entre 0 y 100

  }
```

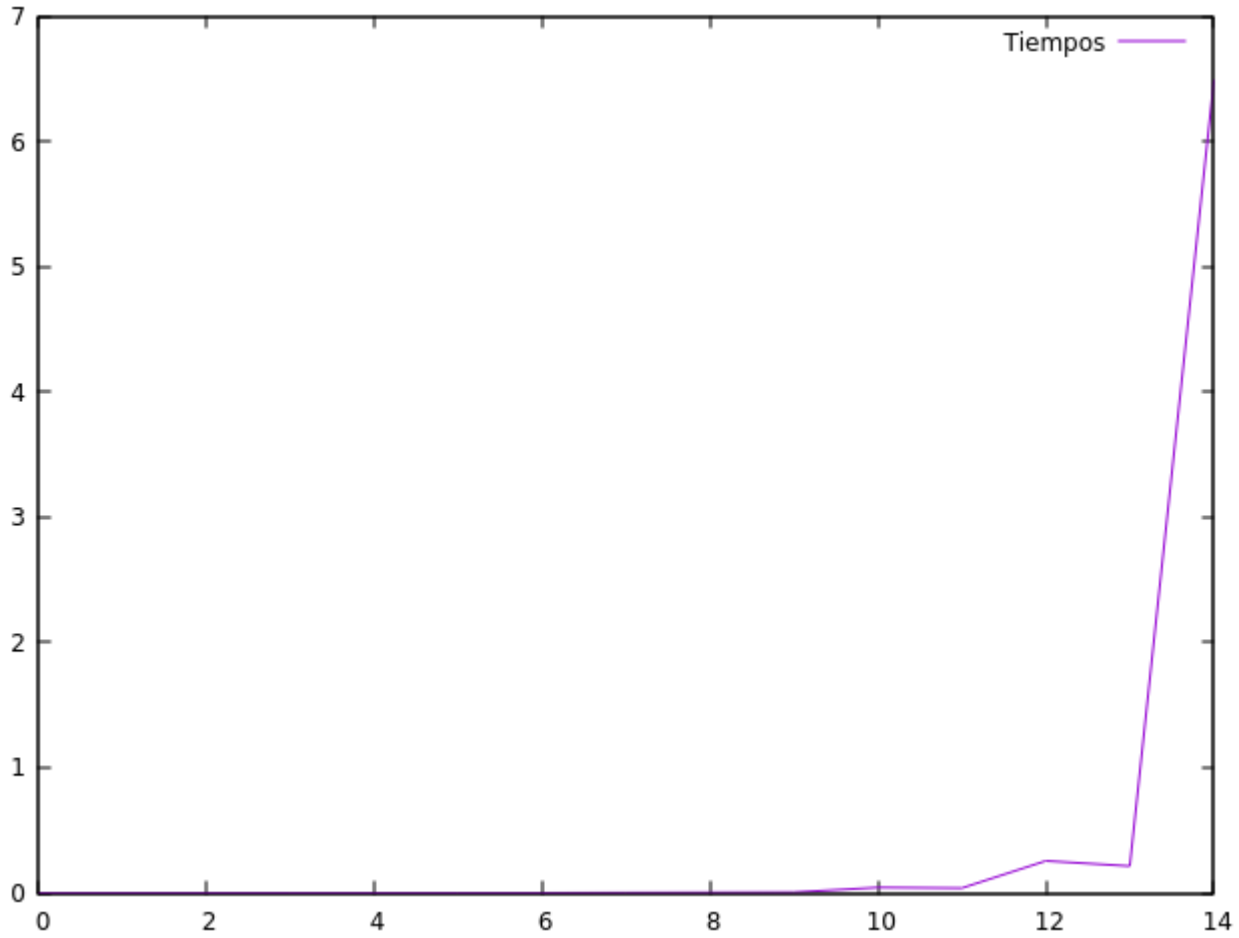
```
vector<int> get_solucion(){
  return solucion;
}

int conveniencia_total(vector<int> v){
  int sum= 0, n= v.size();

  for(int i= 1; i<n-1; i++)
    sum+=m[ v[i] ][ v[i+1] ] + m[ v[i] ][ v[i-1] ];

  sum+=m[ v[n-1] ][ v[n-2] ] + m[ v[n-1] ][ v[0] ];
  sum+=m[ v[0] ][ v[1] ] + m[ v[0] ][ v[n-1] ];

  return sum;
}

//Comprueba si el invitado x ya ha sido considerado
bool sentado(vector<int> v, int x){
  return count(v.begin(), v.end(), x) == 1;
}
```

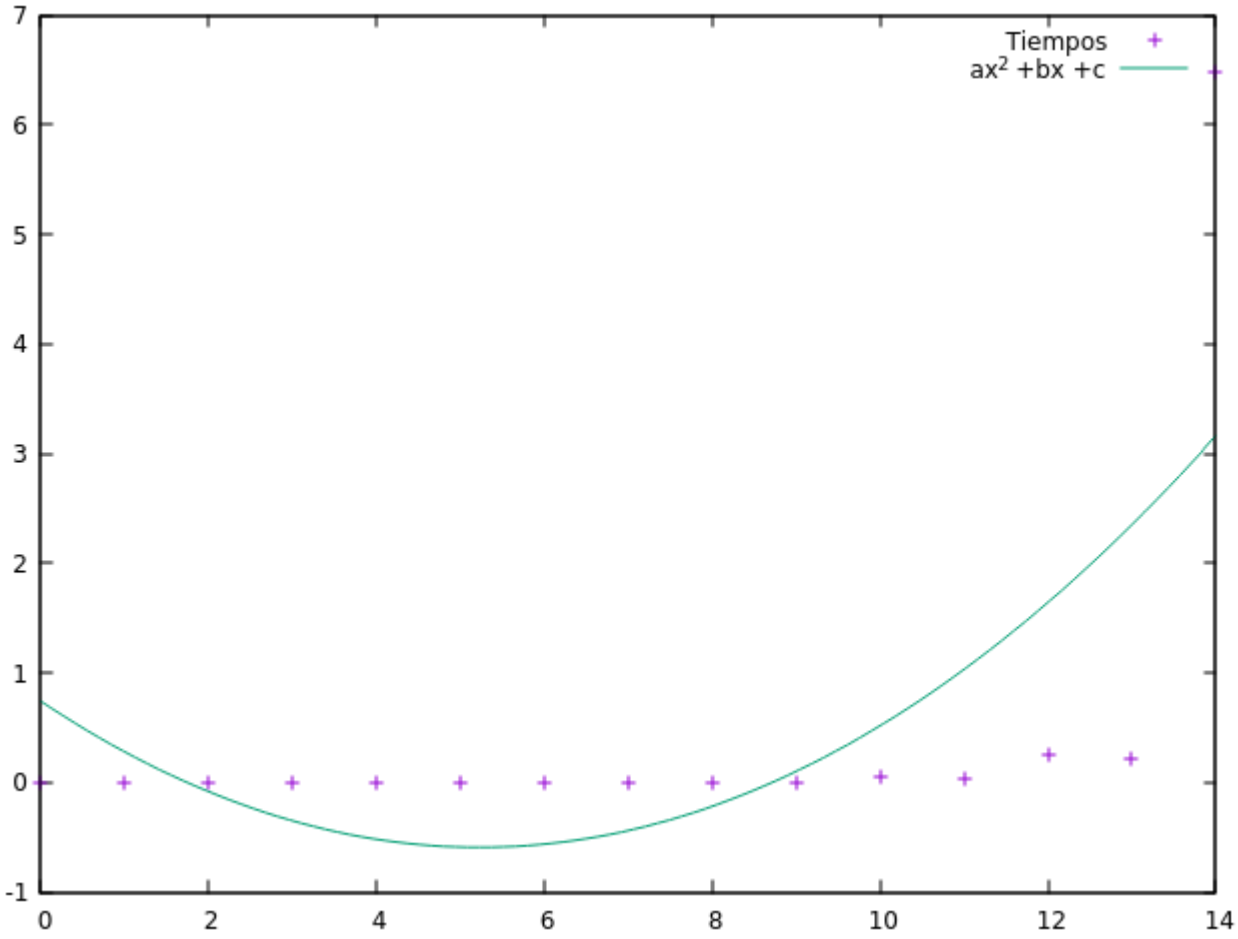
```
vector<int> complementario(vector<int> v, int n){
  vector<int> result;
  for(int i= 0; i< n; i++)
    if(!sentado(v, i))
      result.push_back(i);

  return result;
}
```

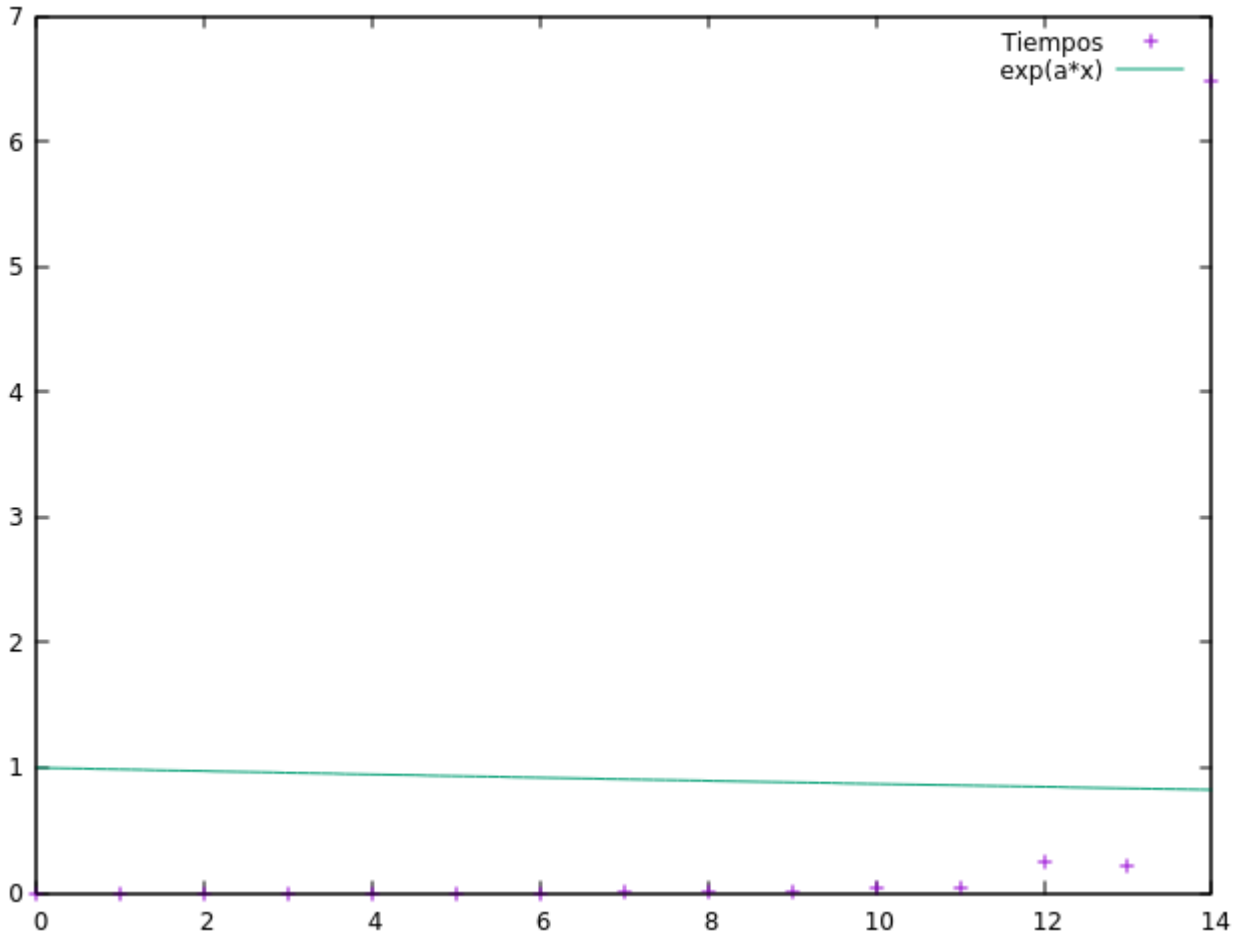
Algoritmo backtracking

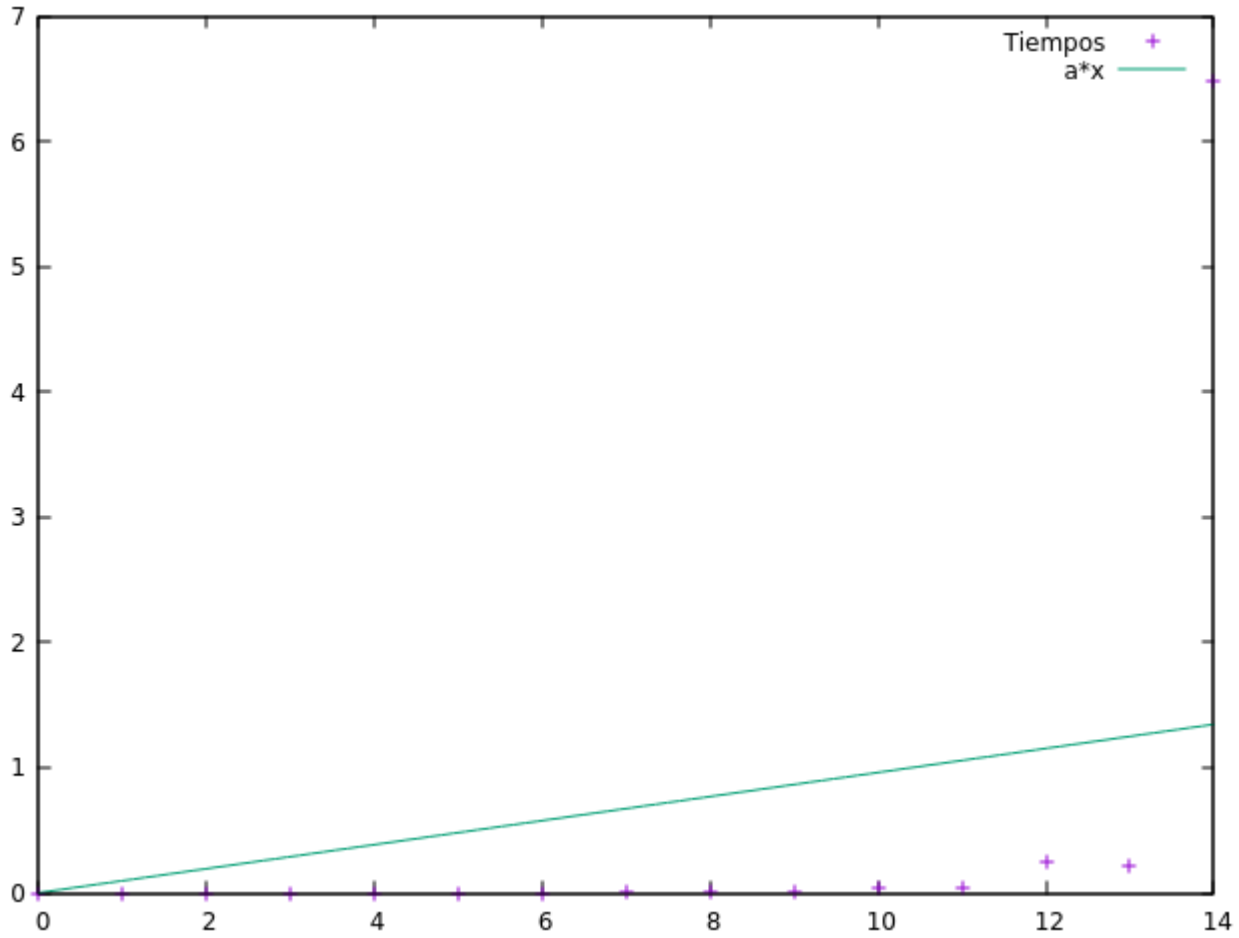
```
int max_nivel_conv(vector<int> v){
  int max_conv= 0, conv= 0;
  vector<int> por_sentar= complementario(v, m.size());

  for(int i= 0; i< por_sentar.size(); i++){
    v.push_back(por_sentar[i]);

    conv= conveniencia_total(v);

    if(conv > max_conv){
      max_conv= conv;
      solucion= v;
      max_nivel_conv(v);
    }

    v.pop_back();
  }
  return max_conv;
}
```

```cpp
#include "matriz_de_conveniencia.cpp"
using namespace std;

int main(int argc, char **argv){
  struct timespec cgt1, cgt2;
  double ncgt;  //para tiempo de ejecución

  if(argc != 2){
    cout << "\nNúmero de argumentos incorrecto";
    exit(-1);
  }

  int n= atoi(argv[1]), max_conv;
  matriz_de_conveniencia m(n);
  vector<int> v, f;

  clock_gettime(CLOCK_REALTIME, &cgt1);
  max_conv=  m.max_nivel_conv(v);
  clock_gettime(CLOCK_REALTIME, &cgt2);

  //cout << "\nEl máximo nivel de conveniencia es: " << max_conv;
  f= m.get_solucion();
  //cout << "\nPara ello los invitados han de estar sentados en el orden: ";
  for(int i= 0; i< f.size(); i++)
    // cout << f[i] << " ";

  ncgt= (double)(cgt2.tv_sec - cgt1.tv_sec) + (double) ((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
  cout << "\n" << ncgt;
}
```


Tiempos
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12
14


Tiempos
ax² +bx +c
7
6
5
4
3
2
1
0
-1
0
2
4
6
8
10
12
14


Tiempos
exp(a*x)
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12
14


Tiempos
a*x
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12
14

FIN