Analyse algorithme dynamique

```
#include "allAlgo.h"
#include <algorithm>
#include <bitset>

vector<vector<double>> getSimpleDistanceVector(vector<XY> & points);
void getDistanceVector(vector<vector<double>> & D,
vector<vector<list<int>>> & ordre, vector<vector<double>> & dist,
vector<int> & S);
list<int> getFinalDistance(vector<XY> & points, vector<vector<double>>
& D, vector<vector<list<int>>> & ordre, vector<int> & S);

void setS(vector<int> & tmp, vector<int> & S, vector<int> & num, int k,
int offset = 0)
{
   if (k == 0) {
       S.push_back(1 << tmp[0]);
       for (size_t i = 1; i < tmp.size(); i++)                  n
       {
               S.back() |= 1 << tmp[i];
       }
       return;
   }
   for (int i = offset; i <= num.size() - k; i++) {             n
       tmp.push_back(num[i]);
       setS(tmp, S, num, k-1, i+1);                             n^2
       tmp.pop_back();
   }
}

list<int> dynamique(vector<XY> & points)
{
   vector<int> num;
   for (int i = 0; i < points.size() - 1; i++)                  n
   {
       num.push_back(i);                                        n
   }

   vector<int> S;
   S.push_back(0);
   for (size_t i = 1; i < num.size(); i++)                      n
   {
       vector<int> tmp;
```

```
        setS(tmp, S, num, i);                                   n^3
    }

    vector<vector<double>> dist = getSimpleDistanceVector(points);    n^2

    vector<vector<double>> D;
    vector<vector<list<int>>> ordre;
    for (int i = 1; i < points.size(); i++)                          n
    {
        vector<double> di(S.size());
        di[0] = sqrt(pow(points[0].x - points[i].x, 2) + pow(points[0].y
- points[i].y, 2));
        D.push_back(di);

        vector<list<int>> oi(S.size());
        list<int> f = {i, 0};
        oi[0] = f;
        ordre.push_back(oi);
    }

    getDistanceVector(D, ordre, dist, S);                        n^2 2^2n

    return getFinalDistance(points, D, ordre, S);                n^2
}

vector<vector<double>> getSimpleDistanceVector(vector<XY> & points)
{
    vector<vector<double>> D;

    int size = points.size();

    for(int i = 1; i < size; i++)                                    n
    {
        vector<double> di;

        for (size_t j = 1; j < size; j++)                            n^2
        {
            double toAdd = -1;
            if(i != j)
            {
                toAdd = sqrt(pow(points[i].x - points[j].x, 2) +
pow(points[i].y - points[j].y, 2));
            }
```

```
            di.push_back(toAdd);
        }

        D.push_back(di);
    }

    return D;
}

vector<int> findBitPosition(int n)
{
    vector<int> posVec;
    int pos = 0;
     while (n) {                                                    n
        if(n & 1)
        {
            posVec.push_back(pos);
        }

        n = n >> 1;
        pos++;
    }
     return posVec;
}

void getDistanceVector(vector<vector<double>> & D,
vector<vector<list<int>>> & ordre, vector<vector<double>> & dist,
vector<int> & S)
{
    int rowSize = D.size();
    int colSize = S.size();

    for(int i = 1; i < colSize; i++)                                    2^n
    {
        vector<int> pos = findBitPosition(S[i]);

        for (size_t j = 0; j < rowSize; j++)                        n 2^n
        {
            double toAdd = -1;
            list<int> ordreToAdd = {-1};
            if(!(S[i] & (1 << j)))
            {
                vector<double> results;
```

```
                vector<list<int>> resultsOrdre;
                int top = pos.size() - 1;
                for (int k = 0; k < pos.size(); k++)            n^2 2^n
                {
                    uint shift = ~(1 << pos[k]);
                    int si = S[i] & shift;
                    int col = 0;
                    while(S[col] != si)                         n^2 2^2n
                    {
                        col++;
                    }
                    if(D[pos[k]][col] != -1)
                    {
                        results.push_back(D[pos[k]][col] +
dist[j][pos[k]]);
                        resultsOrdre.push_back(ordre[pos[k]][col]);
                        resultsOrdre.back().push_front(j + 1);
                    }
                }
                double smallest = -1;
                list<int> ordreMin = {-1};
                for (size_t d = 0; d < results.size(); d++)     n^3 2^n
                {
                    if(smallest == -1 || smallest > results[d])
                    {
                        smallest = results[d];
                        ordreMin = resultsOrdre[d];
                    }
                }

                toAdd = smallest;
                ordreToAdd = ordreMin;
            }

            D[j][i] = toAdd;
            ordre[j][i] = ordreToAdd;
        }
    }
}

list<int> getFinalDistance(vector<XY> & points, vector<vector<double>>
& D, vector<vector<list<int>>> & ordre, vector<int> & S)
{
```

```
    double smallest = -1;
    int idxISmallest = 0;
    int idxJSmallest = 0;
    for (size_t i = 0; i < D.size(); i++)                                   n
    {
        for (size_t j = D[i].size() - D.size() + 1; j < D[i].size();
j++)                                                                        n^2
        {
            if(D[i][j] != -1)
            {
                double val = D[i][j] + D[i][0];
                if(smallest == -1 || smallest > val)
                {
                    smallest = val;
                    idxISmallest = i;
                    idxJSmallest = j;
                }
            }
        }
    }
    ordre[idxISmallest][idxJSmallest].push_front(0);

    return ordre[idxISmallest][idxJSmallest];
}
```