Universidade do Minho	Escola de Engenharia da Universidade do Minho **Mestrado Integrado em Eng. Electrónica Industrial e Computadores** **Complementos de Programação de Computadores**	**2014/2015** **MIEEIC** **(1º Ano)** **2º Sem**
	Luís Paulo Reis	
	Aula Prática 6: Análise de Complexidade	

Objectivos:

Esta Folha de Exercícios destina-se a:

• Compreender o conceito de complexidade temporal.

Os exercícios aqui propostos deverão ser realizados no mais simples ambiente de desenvolvimento possível para a linguagem C: editor de texto de programação ou editor DevC++ e ferramentas da GCC (GNU Compiler Collection) e afins.

Exercício 6

6.1) Suponha o seguinte programa para calcular o tempo decorrido e o número de operações dominantes executadas para diversos algoritmos de resolução do problema da subsequência máxima.

Construa um programa que gere um ficheiro de texto que permita comparar a complexidade temporal teórica e prática (medindo o número de operações) para um vetor de entrada com dimensão N sendo N>=1 e N<=500000

```
// Programa Exemplo de Contagem de Operações e de Tempo Subsequência Máxima
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <ctime>
#include <vector>

using namespace std;

long int op;  //Global Counter for operations

// Time (in seconds) between clock1 and clock2
double diffclock(clock_t clock1,clock_t clock2)
{
      return (double)(clock1-clock2)/CLOCKS_PER_SEC;
}
```

```
// Creates a random vector with n elements between 0 and 1000000
void cria_vetor(int v[], int n)
{
    for(long int i=0; i<n; i++)
        v[i] = rand() % 100 - 50;
}

// Writes the vector on the screen
void escreve_vetor(int v[], int n)
{
    for(long int i=0; i<n; i++)
        cout << v[i] << " ";
    cout << endl;
}

// MaxSubSum1: Calcula a Subsequência máxima utilizando
// três ciclos (O(n^3))
template <class T> long int maxSubSum1(T vec[], long int n)
{
    T maxSum = 0;
    for (int i = 0; i < n; i++)
       for (int j = i; j < n; j++)
       {
                  T thisSum = 0;
            for (int k = i; k <= j; k++) {
                        thisSum += vec[k];
                        op++;
            }
            if (thisSum > maxSum) maxSum = thisSum;
       }
       return maxSum;
}

// MaxSubSum2: Calcula a Subsequência máxima utilizando
// dois ciclos(O(n^2))
template <class T> T maxSubSum2(T vec[], long int n)
{
    T maxSum = 0;
    for (int i = 0 ; i < n; i++)
       {
            T thisSum = 0;
            for (int j = i; j < n; j++) {
          thisSum += vec[j];
          op++;
          if (thisSum > maxSum) maxSum = thisSum;
      }
       }
       return maxSum;
}
```

```
// MaxSubSum3: Calcula a Subsequência máxima utilizando
// um único ciclo (O(n))
template <class T> T maxSubSum3(T vec[], long int n)
{
    T thisSum = 0, maxSum = 0;
    for (int j=0; j < n; j++) {
       thisSum += vec[j];
       op++;
       if (thisSum > maxSum) maxSum = thisSum;
       else if (thisSum < 0) thisSum = 0;
    }
    return maxSum;
}

#define MAX 500000

//Test Times and operations for subsum algorithms
void testa_subsum(ostream &co)
{
    int v[MAX];
    long int i,j;
      clock_t begin, end;

    long int N = 10; // N = 100   N =1000; N =5000; N =50000; N=500000;
    {
      co << N << " ; ";
        cria_vetor(v,N);
        //escreve_vetor(v,N);
 /*
      op=0; begin=clock();
        long int max = maxSubSum1(v, N);
        end=clock();
        co << endl << "Max1: " << max << "  " << op << " ; "
            << diffclock(end,begin) << " ; ";
 */
 /*
      op=0; begin=clock();
        long int max = maxSubSum2(v, N);
        end=clock();
        co << endl << "Max2: " << max << "  " << op << " ; "
            << diffclock(end,begin) << " ; ";
 */
 /*
      op=0; begin=clock();
        long int max = maxSubSum3(v, N);
        end=clock();
        co << endl << "Max3: " << max << "  " << op << " ; "
            << diffclock(end,begin) << " ; ";
 */
  }
}
```

```
//Main - Calls subsum
int main()
{
      srand(time(NULL));
      testa_subsum(cout);
}
```

6.2) Analise a complexidade teórica e comprove na prática (através do número de operações executadas dos seguintes algoritmos):

```
  vector<int> v1;
  int n=100;  //change n
```

Algorithm a)

```
   for(int i=0; i<n; i++)
       cout << i << endl;
```

Algorithm b)

```
   for(int i=0; i<n; i++)
     for(int j=0; j<n; j++) {
         cout << i << " - " << j << endl;
     }
```

Algorithm c)

```
   for(int i=0; i<n; i++)
     for(int j=1; j<n; j*=2) {
         cout << i << " - " << j << endl;
     }
```

Algorithm d)

```
   for(int i=0; i<n; i++)
     for(int j=0; j<n; j*=2) {
         cout << i << " - " << j << endl;
     }
```

Algorithm e)

```
  for(int i=0; i<n; i+=2)
     for(int j=2; j<n-1; j++)
         for(int k=j; k<n; k++)
             v1.push_back(i+j+k);
```

Algorithm f)

```
  for(int i=0; i<n; i+=2)
     for(int j=2; j<n; j++)
         for(int k=2; k<10; k++)
             v1.push_back(i+j+k);
```