Centre Universitaire de Mila

Institut des sciences et de la technologie

2eme année LMD informatique

Module A.S.D

**Série de TP N 01**

**Complexité algorithmique**

Par le calcul de la complexité algorithmique nous voulons principalment :

1. Comparer l’efficacité d’algorithmes résolvant le même problème.
2. Verfier si un algorithme resolvant un problème peut donner une réponse dans un temps raisonable.

Le premier objectif de ce TP et de comparer pratiquement des algorithmes selon le nombre d’itérations effectués (exercice 1 et 2) et selon leurs temps d’execution (exercice 3). Le deuxième objectif c’est de voir si un programme peut répendre dans un temps raisonable ou non (exercice 3).

**Exercice 1 :**

Soit les deux programmes suivants calculant la somme des n premiers entiers naturels.

**Le programme P1**

#include <stdio.h>

#include <conio.h>

int main ()

{ int i, n; long S=0, Nbriteration=0;

 printf("Entre un entier n :");

 scanf("%d", &n);

 for ( i=1; i<=n; i++)

 {

 S = S+ i;

 Nbriteration++;

 }

 printf("\n la somme est : %ld ", S) ;

 printf("\n le nombre d'iterations est : %ld ", Nbriteration);

}

**Le programme P2**

#include <stdio.h>

#include <conio.h>

int main ()

{ int i, j, n; long S=0, Nbriteration=0;

printf("Entre un entier n :");

scanf("%d", &n);

 for ( i=1; i<=n; i++)

 {

 for ( j=1; j<=i; j++)

 { S = S+ 1;

 Nbriteration++;

 }

 }

 printf("\n la somme est : %ld ", S) ;

 printf("\n le nombre d'iterations est : %ld ", Nbriteration);

}

1. Tester les deux programmes pour déférentes valeurs de n en comparant le nombre d’itérations effectuées par ces deux programmes. Que remarquez vous lorsque n=100000 pour le programme P2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N  | 5 | 20 | 50 | 10000 | 100000 |
| Nbr Itérations | P1 |  |  |  |  |  |
| P2 |  |  |  |  |  |

1. Théoriquement le nombre d’itération du programme P1 est N tandis que le nombre d’itération du programme P2 est N\*(N+1)/2. Comparer les résultats du tableau avec les résultat théorique.
2. Proposer une solution plus rapide que les deux solution precedentes.

**Exercice 2:** Il existe deux techniques principales pour la recherche d’un élément dans un tableau, séquentielle et dichotomique. Dans cet exercice nous allons comparer le nombre d’itérations effectués par ces deux techniques lors de la recherche d’un élément dans un tableau rempli de 10000 entiers (0,1,2,3,…,9999).

1. Definir les deux fonctions rechseq et rechdich. Les fonctions retournent le nombre d’itérations effectuées lors de la recherche d’un élément.
2. Tester le programme pour les valeurs suivant de x :5, 25, 300, 6000,10000, -1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N  | 20 | 30 | 40 | 50 | 10000 | 1000000000 |
| Temps d’exécution | f1 |  |  |  |  |  |  |
| f2 |  |  |  |  | / | / |

#include <iostream>

using namespace std;

int rechseq (int T[], int N, int x); // recherche sequentielle

int rechdich (int T[], int N, int x); // recherche dichotomique

int main ()

{

 int T[10000]; int N=10000; int N1=0, N2, x;

 for (int i=0; i<N; i++)

 T[i] = i;

 cout << "entrez un nombre entier x:";

 cin>>x;

 N1=rechseq (T, N, x);

 N2=rechdich (T, N, x);

 cout<< "\nle nombre d'itérations en utilisant une recherche sequentielle est"<<N1<<endl;

 cout<< "le nombre d'itérations en utilisant une recherche dichotomique est"<<N2<<endl;

}

**Exercice 3:** La suite de Fibonacci est la suite telle que :

U0 = 1, U1 = 1 et pour n >= 2, Un= Un-1 + Un-2.

Soit les deux fonctions **f1** et **f2** suivantes calculant le nième terme de la suite de Fibonacci définie ci-dessus.

**Version récursive** complexité (***O*(2n**) )

int f2 (int n)

{

if (n<=1)

return 1;

else

return f2(n-1) +f2(n-2);

}

**Version itérative** Complexité ***O*(N)**

int f1 (int n)

{int u0=1,u1=1,Un;

 if (n<=1)

return 1;

 else

 {

for (int i=2; i<=n; i++)

{

 Un = u1+u0;

 u0=u1;

 u1=Un;

 }

return Un;

 }

1. A l’aide des fonctions de **time.h**, comparer les temps d’exécution de ces deux fonctions pour différentes valeurs de n en remplissant le tableau suivant.
2. Selon les temps d’éxecution optenus quelle fonction choisissez vous pour calculer le nième terme de la suite de Fibonacci. Est-ce que on peut utiliser la fonction f2 pour calculer le nième terme de la suite de Fibonacci.

\*\*\*\*\*\* Exemple de culcule du temps d’éxecution\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include <time.h>

#include<stdlib.h>

.

.

.

int main()

{

 clock\_t T1, T2; double temps;

 T1 = clock(); // prend l’horloge avant l’appel de la fonction

 f(. . .); // l’appel de la fonction f

 T2 = clock(); // prend l’horlog apèrs l’appel de la fonction

 // calcule du temps d’éxecution en second

 temps = (double)(T1 – T2) / (double)CLOCKS\_PER\_SEC;

 printf("\n le temps d'execution est **%.3f** seconde(s)!\n", temps);

 .

 .

}