Rapport de TP 4

Algorithmes de Complexité temporelle quadratique

Khezour Mohamed Oussama

Meceffeuk Adda

Khiat Mokhtar

1- Développer un algorithme itératif pour le tri avec la méthode par sélection de n nombres entiers naturels enregistrés dans un tableau.

```
Algorithme Tri_Selection
 local:   m, i , j , n, temp :  Entiers naturels
 Entrée : Tab  : Tableau d'Entiers naturels de 1 à n éléments
 Sortie : Tab  : Tableau d'Entiers naturels de 1 à n éléments
début
 pour i de 1 jusquà n-1 faire
  m = i ; // i est l'indice de l'élément frontière Tab[ i ]
  pour j de i+1 jusquà n faire // liste non-triée : (ai+1, a2, ... , an)
   si Tab[ j ] <  Tab[ m ] alors // aj est le nouveau minimum partiel
     m = j ;
     temp = Tab[ m ] ;
     Tab[ m ] = Tab[ i ] ;
     Tab[ i ]  = temp //on échange les positions de ai et de aj
     m = i ;
    Fsi
  fpour
 fpour
Fin Tri_Selection
```

2- Calcule de complexité :

2-1- Calculer la complexité temporelle, notée CT(n) en notation asymptotique de Landau O (Grand O) de cet algorithme. Préciser le meilleur cas, le pire cas et/ou le cas exact.

Le nombre de comparaisons "si Tab[j] < Tab[m] alors" est une valeur qui ne dépend que de la longueur n de la liste (n est le nombre d'éléments du tableau), ce nombre est égal au nombre de fois que les itérations s'exécutent, le comptage montre que la boucle "pour i de 1 jusquà n-1 faire" s'exécute n-1 fois (donc une somme de n-1 termes) et qu'à chaque fois la boucle "pour j de i+1 jusquà n faire" exécute (n-(i+1)+1 fois la comparaison "si Tab[j] < Tab[m] alors".

La complexité en nombre de comparaison est égale à la somme des n-1 termes suivants (i = 1, ...i = n-1)

C = (n-2)+1 + (n-3)+1 +.....+1+0 = (n-1)+(n-2)+...+1 = n.(n-1)/2 (c'est la somme des n-1 premiers entiers).

La complexité en nombre de comparaison est de de l'ordre de n^2, que l'on écrit $O(n^2)$.

Au pire chaque cellule doit être échangée, dans cette éventualité il y a donc autant d'échanges que de tests.

La complexité au pire en nombre d'échanges de la version 1 est de l'ordre de n^2, que l'on écrit $O(n^2)$.

2.2- Calculer la complexité spatiale en notation asymptotique de Landau O (Grand O) de cet algorithme notée CS(n).

Il s'agit du nombre de case mémoire ou octets utilisés par le programme. Pour calculer la complexité spatiale de cet algorithme nous allons considérer les tailles mémoires des types en langage C ; tel que nous aurons : int/entier : 2 octets .

1. En notation exacte : Nous avons dans cet algorithme cinq (5) variables de type "entier" en plus d'un tableau de n entiers. Ce qui nous fait : (n + 5)(2 octets) = 2n + 10 octets

2. En notation asymptotique : Le nombre de case mémoire dépend uniquement de la taille du tableau, donc on obtient : s(n) = O(n) ;

3- Ecrire le programme avec le langage C :

```c
#include <stdio.h>

void permuter(int *a, int *b) {
      int temp = *a;
      *a = *b;
      *b = temp;
}

void Tri_Selection(int arr[], int n) {
      int i, j, min = 0;

      for (i = 0; i < n-1; i++ , min = i  ){
    for (j = i+1; j < n; j++)
      if (arr[j] < arr[min])
        min = j;
    permuter(&arr[min], &arr[i]);
  }

}

void affiche(int arr[], int size) {
      int i;
      for (i=0; i < size; i++)
            printf("%d ", arr[i]);
      printf("\n");
}

int main() {
      int arr[] = {64, 25, 12, 22, 11};
      int n = sizeof(arr)/sizeof(arr[0]);
      Tri_Selection(arr, n);
      affiche(arr, n);
      return 0;
}
```

4- Mesurer les temps d'exécution T pour l'échantillon suivant de la taille n et compléter le tableau ci-dessous. On suppose que les données peuvent se présenter en entrée selon 3 configurations

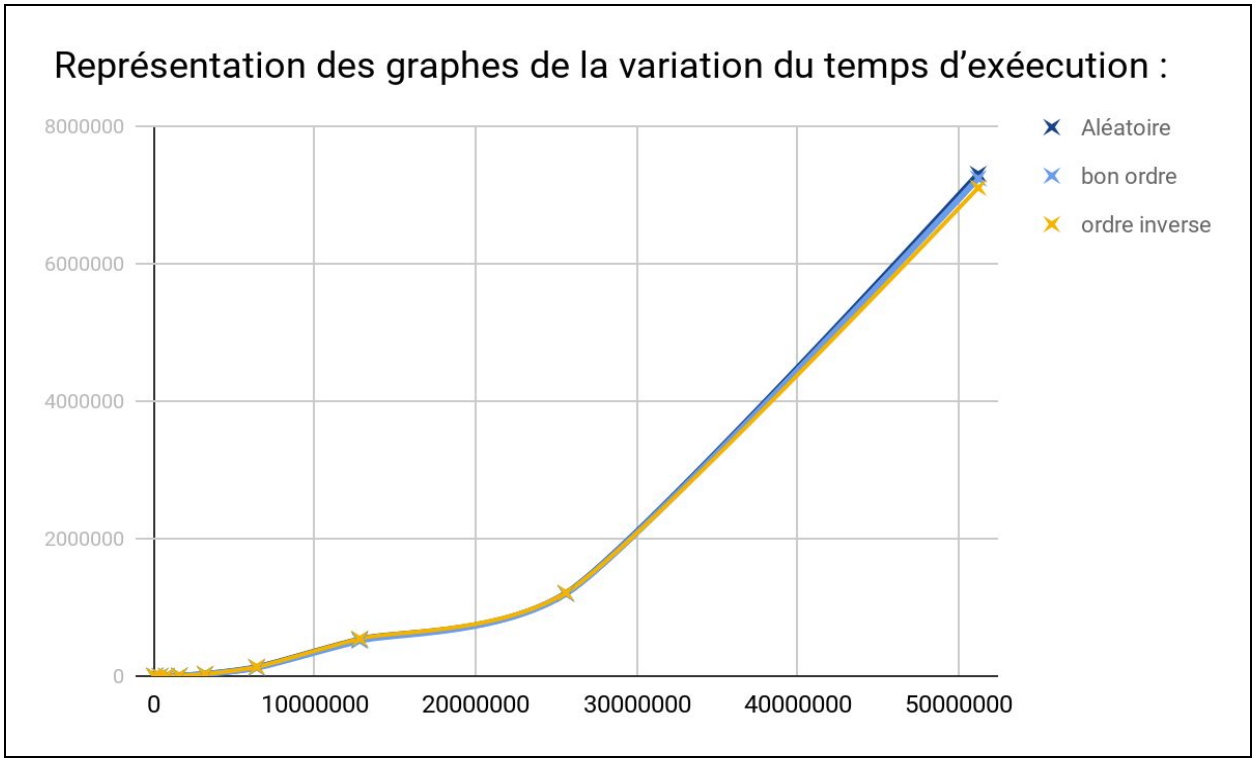
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_SIZE 204800000

typedef  long long unsigned int big_int ;

big_int N[] ={ 50000 , 100000 , 400000 , 800000 , 1600000 , 3200000 ,
6400000 , 12800000 , 25600000 , 51200000 , 102400000 , 204800000 };

void permuter(big_int *a, big_int *b) {
      big_int temp = *a;
      *a = *b;
      *b = temp;
}

void selectionSort(big_int arr[], big_int n) {
  big_int i, j, min = 0;
  for (i = 0; i < n-1; i++ , min = i  ){
    for (j = i+1; j < n; j++)
      if (arr[j] < arr[min])
        min = j;
    permuter(&arr[min], &arr[i]);
  }
}

int main() {
  clock_t t1,t2;
  big_int *arr_sorted , *arr_reversed , *arr_rand;

  printf("\n1 -> Les données du tableau sont triées en bon ordre ;");
  arr_sorted = (big_int*)malloc(sizeof(big_int) * MAX_SIZE);

  big_int i , j;
```

```
  for (i = 0 ; i < 12  ; i++ ){
    for ( j = 0 ; j < N[i] ; j++ )
      arr_sorted[j] = j ;

    t1=clock();
    selectionSort(arr_sorted , N[i]);
    t2=clock();
    printf("\n  N=%llu\t -- Execution time: %fs", N[i],
(double)(t2-t1)/CLOCKS_PER_SEC );

  }

  printf("\n2 -> Les données du tableau sont triées en ordre inverse ;");
  arr_reversed = (big_int*)malloc(sizeof(big_int) * MAX_SIZE);
  for (i = 0 ; i < 12  ; i++ ){
    for ( j = 0  ; j < N[i] ; j++)
      arr_reversed[j] = N[i] -j ;

    t1=clock();
    selectionSort(arr_reversed , N[i]);
    t2=clock();
    printf("\n  N=%llu\t -- Execution time: %fs", N[i],
(double)(t2-t1)/CLOCKS_PER_SEC );

  }

  printf("\n3 -> Les données du tableau ne sont pas triées (çàd.,
aléatoires). ");
  arr_rand = (big_int*)malloc(sizeof(big_int) * MAX_SIZE);
  for (i = 0 ; i < 12  ; i++ ){
    for ( j = 0 ; j < N[i] ; j++ )
      arr_rand[j] = rand() ;

    t1=clock();
    selectionSort(arr_rand , N[i]);
    t2=clock();
    printf("\n  N=%llu\t -- Execution time: %fs", N[i],
(double)(t2-t1)/CLOCKS_PER_SEC );
  }
      return 0;
}
```

N	50000	100000	400000	800000	1600000	3200000
Temps (bon ordre)	4.561	16.354	265.3626	1088.0345	5859.864	29451.21
Temps (ordre inverse)	4.902	16.710	273.5219	1102.551	5921.214	29842.95
Temps (Aléatoire)	4.7201	16.515	269.218	1072.9348	5940.643	30107.24

N	6400000	12800000	25600000	51200000	102400000	204800000
Temps (bon ordre)	121419.69	514937.01	1198672.10	7241269.84		
Temps (ordre inverse)	134282.45	544153.59	1212146.92	7103245.9		
Temps (Aléatoire)	129193.22	533490.21	1203553.12	7303460.1		

5- Représenter avec des graphes les variations du temps d'exécution mesuré T(n) et les variations des complexités théoriques.



6- Interprétation des résultats :

6.1 Comparaison des mesures de temps d'exécution avec le pire et meilleur cas :

Vu que la complexité au pire et meilleur cas sont identiques, il est claire que les temps d'exécutions suivent la même évolution, en O(n 2).

6.2 Remarque et déduction d'une fonction T(n) reliant n au temps d'exécution.

On remarque que les temps d'exécution sont approximativement multipliés par 4 lorsque N est doublé n'importe l'ordre des éléments du tableau.

On en déduit que le temps d'exécution est proportionnel à N, ce que l'on peut représenter par la formule suivante :

$$T(x * N) = x\,2 * T(N) \text{ pour tous } x * N \in [50000 - 2048000000]$$

(x étant la tangente d'un point sur le graphe).

6.3 Comparaison de la complexité théorique et expérimentale.

Dans le cas des données de l'échantillon la complexité théorique et expérimentale sont du même ordre de grandeur que la complexité théorique du pire et meilleur cas, Donc Le mod`ele théorique est conforme aux mesures expérimentales.

Même si en générale la complexité expérimentale d'un échantillon quelconque est compris entre la complexité au pire cas et au meilleur cas.

Complexité Meilleur cas = complexité expérimentale = complexité Pire cas

7- Le tri par sélection avec le langage Java

```java
class TriSelection{
    static void Tri_Selection(int arr[]){
        int n = arr.length;
        for (int i = 0; i < n-1; i++){
            int min = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min = j;

            int temp = arr[min];
            arr[min] = arr[i];
            arr[i] = temp;
        }
    }

    static void affiche(int arr[]){
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    public static void main(String args[]){
        int arr[] = {64,25,12,22,11};
        Tri_Selection(arr);
        affiche(arr);
    }
}
```