IMPLEMENTATION ET ANALYSE DE QUELQUES ALGORITHMES DE TRI

RAPPORT DE TRAVAIL

2019/2020

MODULE : ALGORITHMIQUE

PREPARÉ PAR

A.ZIANI - I.KHELIFA

DEPARTEMENT D'INFORMATIQUE-UNIVERSITE USTHB

IMPLEMTATION ET ANALYSE D'ALGORITHMES DE TRI

AbdelHakim Ziani, Ihab Elhakim Khelifa

Résumé: Le travail présenté dans ce rapport s'est porte sur les algorithmes de tri pour le projet du semestre 3 L2 ACAD informatique, qui consiste à implémenter 4 algorithmes : Bubble Sort, Selection Sort, Insertion Sort et Shaker Sort sur différentes structures de données : Vecteurs, Matrices, Listes chainées. L'utilisateur doit choisir une seule structure par exécution, celles-ci sont déclarées de manière dynamique. Le programme affiche après chaque itération la situation de la structure, le nombres de comparaisons, le nombres de permutations et enfin la situation finale.

Introduction

Le programme commence par demander à l'utilisateur de choisir une structure de donnée à travers un menu a choix, en entrant son numéro : 1- pour le vecteur, 2- pour la matrice et 3- pour la liste (1,2,3 sont considérés en tant que caractères). Ensuite l'utilisateur est demandé de faire entrer l'algorithme de son choix : 1- pour Bubble, 2- pour Selection, 3- pour Insertion et 4- pour le Shaker Sort, le menu continue de s'actualiser en lui demandant l'ordre du tri : 1- pour Ascendant et 2- pour Descendant. Cette partie ce finit par l'insertion de la structure « Inputting Data part » et le programme passe à la prochaine étape, L'étape du tri « Sorting Part ». Et finit par donner le résultat final du tri ainsi que les étapes intermédiaires et le nombre de comparaisons et permutations effectuées.

Copyright et autorisations

Plan global du projet

Pour des raisons de simplicité, le projet est constitué de 7 fichiers, 1 du programme principal « main.c » et 6 fichiers de fonctions intitulés array_functions.h, array_sorting.h, matrix_functions.h, matrix_sorting.h, list_functions.h, list_sorting.h.

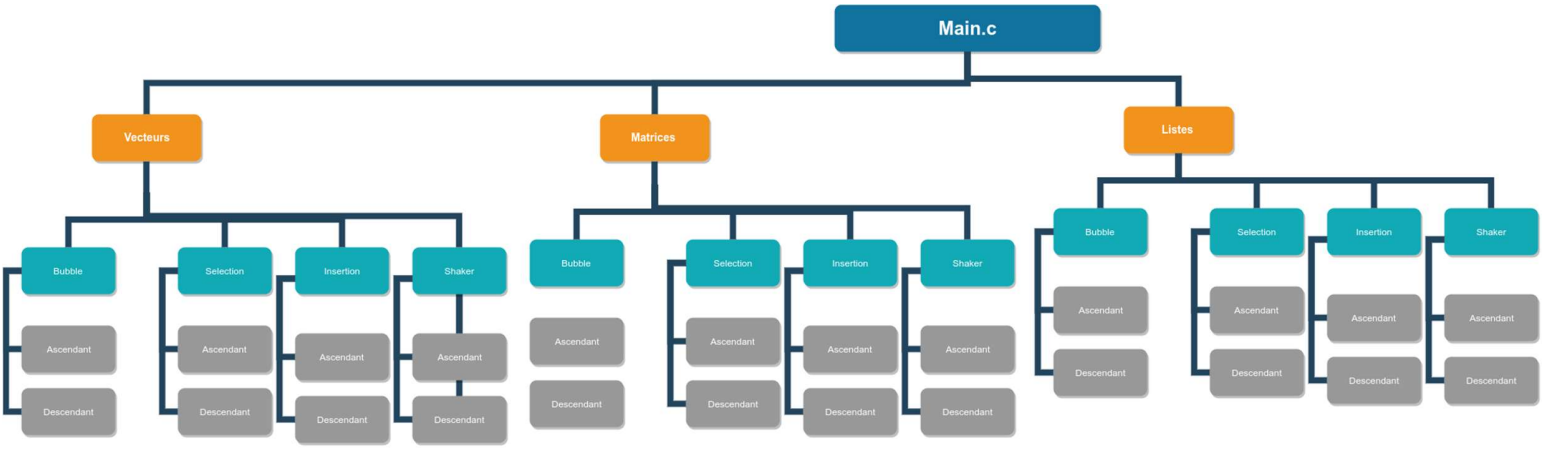
Les fichiers '***_fonctions.h' contiennent les fonctions de manipulations des structures tels que input_array(), print_array()…, et les fichiers '***_sorting.h' contiennent les fonctions de tri des structures.

Bibliothèques utilisées

Le fichiers main.c fait appel à 4 bibliothèques : stdio.h, stdlib.h, string.h, ctype.h.

Organigramme général du projet

La figure ci-dessous montre l'algorigram général du projet



Le tri des vecteurs

Tri par Bulles

Complexité

La complexité du Bubble Sort applique aux vecteurs par la notation de Landau est : O(n^2).

```c
//Bubble sort algorithm for both orders
void Bubble_sort_vector(char order,int vector[],int n)
{
  int nbcmp=0;
  int nbperm=0;
  for(int i=0;i<n-1;i++)
  {
    for(int j=0;j<n-i-1;j++)
    {
      //Depending on the order(Ascending or Descending
)     // 1 for Ascending
      // 2 for Descending
      if(order == '1' )
      {
        if(vector[j]>vector[j+1])
        {
        nbperm+=1;
        swap(&vector[j],&vector[j+1]);
          nbcmp+=1;

        }
      }
      else{
        if(vector[j]<vector[j+1])
        {
          swap(&vector[j],&vector[j+1]);
          nbperm+=1;
          nbcmp+=1;
        }
      }
    }
    if (nbperm==0)
      break;
    printf(" The #%d iteration : \n",i+1);
    print_array(vector,n);
  }
  printf("Number of comp : %d :\n",nbcmp);
  printf("Number of permutations : %d \n",nbperm);
  printf("-----------------------------\n" );
}
```

Tri par Insertion

Complexité

La complexité de l'Insertion Sort applique aux vecteurs par la notation de Landau est : O(n^2)

```c
//Insertion sort functino :
//-------------------------
void insertion_sort_array(char order,int array[],int
n)
  int alpha,j;
  int nbcmp=0;
  int nbperm=0;
  for(int i=1;i<=n-1;i++)
  {
    alpha = array[i];
    j=i-1;
    if(order=='1')
    {
      while(j>=0 && array[j]>alpha)
      {
        array[j+1]=array[j];
        j--;
      }
      array[j+1]=alpha;
    }
    else
    {
      while(j>=0 && array[j]<alpha)
      {
        array[j+1]=array[j];
        j--;
      }
      array[j+1]=alpha;
    }
    printf(" The #%d iteration : \n",i);
    print_array(array,n);
    }
    printf("Number of comp : %d :\n",nbcmp);
    printf("Number of permutations : %d \n",nbperm );
    printf("-----------------------------\n" );

}
```

Tri par Selection

Complexité

La complexité du Selection Sort applique aux vecteurs par la notation de Landau est : O(n^2).

```c
//selection sort function
//-----------------------
void selection_sort_array(char order,int array[],int
{)
  int j,alpha;
  int nbcmp=0;
  int nbperm=0;
  for(int i=0;i<n-1;i++)
  {
    alpha = i;
    if(order == '1')
    {
      for(int j=i+1;j<=n-1;j++)
        {
        if(array[j]<array[alpha])
          {alpha=j;}
          nbcmp+=1;

        }
      if(alpha!=i){
        swap(&array[i],&array[alpha]);
        nbperm++;}
    }
    else
    {
      for(int j=i+1;j<=n-1;j++)
      {
        if(array[j]>array[alpha])
          {alpha=j;}
          nbcmp+=1;


      }
     if(alpha!=i){
        swap(&array[i],&array[alpha]);
        nbperm++;}

    }
    if (nbperm==0)
    break;
    printf(" The #%d iteration : \n",i+1);
    print_array(array,n);
  }
printf("Number of comp : %d :\n",nbcmp);
printf("Number of permutations : %d \n",nbperm );
printf("-----------------------------\n" );
}
```

Tri par Shaker

Complexité

La complexité du Shaker Sort applique aux vecteurs par la notation de Landau est : O(n^2).

```c
// Additional sorting algorithm
// Shaker sort
void shaker_sort_array(char order,int array[],int
{)
  int nbperm=0,nbcomp=0;
  for(int i=0 ;i<n-1;i++)
  {
    for(int j=i;j<n-1-i;j++)
    {
      if(order=='1')
      {
       if(array[j]>array[j+1])
        {
        swap(&array[j],&array[j+1]);
        nbcomp++;
        nbperm++;
        }
      }else{
        if(array[j]<array[j+1])
         {
           nbcomp++;
           nbperm++;
         swap(&array[j],&array[j+1]);
      }}
    }
    for(int l=n-1-i;l>i;l--)
    {
      if(order =='1' )
      {
      if(array[l]<array[l-1])
      {nbcomp++;
      nbperm++;
        swap(&array[l],&array[l-1]);
      }}
      else{
        if(array[l]>array[l-1])
        {
          nbcomp++;
          nbperm++;
          swap(&array[l],&array[l-1]);
        }
      }
    }
  }
  printf("Number of permutations : %d \n",nbperm );
  printf("Number of comparisons  : %d\n",nbcomp );
}
```

Analyse des résultats

Après avoir calculé la complexité du Bubble, Sélection, Insertion et Shaker Sort le résultat observé et le même pour les quatre algorithmes donne avec la notation de Landau(on). Vous remarquez que la complexité est quadratique, ce qui signifie que la vitesse devient très rapidement lente en ajoutant des structures de plus en plus grandes, La figure 1 explique l'augmentation de cette vitesse. De la figure 1, on peut affirmer qu'après 20 éléments entre, ces algorithmes deviennent très lents ce qui nécessite de trouver d'autres algorithmes capables à répondre à nos besoins.

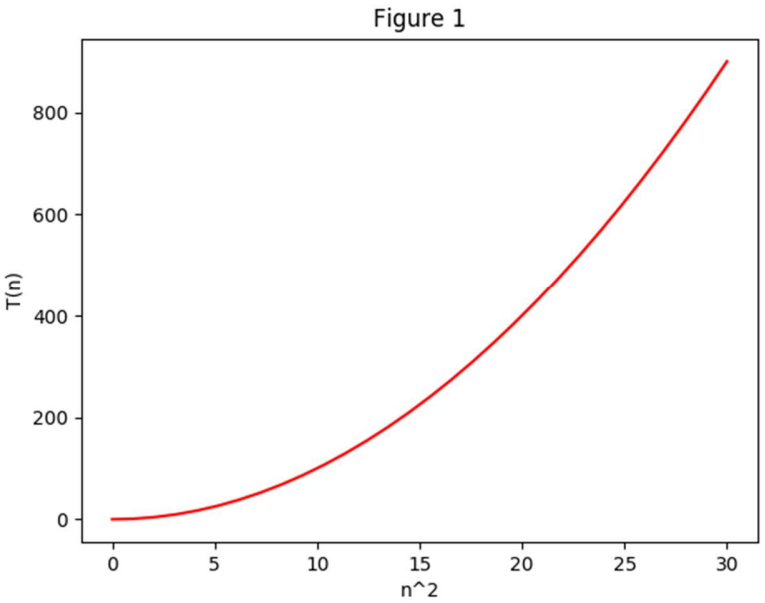


Figure 1 : Une courbe représentant la complexité des algorithmes utilise en facteur de temps d'exécution en fonction de n.

Conclusion

En conclusion de ce projet qui visait à implémenter et estimer la complexité de 4 algorithmes de tri Bubble, Selection, Insertion et Shaker Sort on peut Affirmer l'hypothèse de l'inefficacité de ces algorithmes implémenter a de grandes structures de données. Ceci a été clairement prouve et démontre par l'étude de complexité par rapport à d'autres algorithmes de tri. Pour tout dire il faut choisir un algorithme en fonctions des besoins à implémenter.