

COMPLEXITÉ

Master 1 IL
Groupe 2
2018

Rapport de TP N°2 COMPLEXITÉ : Recherche d'élément

BOUDOUR Mehdi / 201500008386/ TP: Recherche d'élément



**[ALGORITHMIQUE AVANCÉE
ET COMPLEXITÉ]**

Ce document présente les solutions en 5 étapes : (1) les algorithmes écrits en pseudo-code. (2) le calcul de la complexité au pire des cas. (3) Implémentation de l'algorithme en langage C. (4) capture de l'exécution de l'algorithme. (5) représentation graphique de l'évolution du temps d'exécution en fonction de N. Le programme C complet contenant les détails (affichage, calcul du temps d'exécution,...) d'implémentation est présenté à la fin du document.

I. Algorithme rechElets TabNonTriés :

Soit un tableau de n (n>=2) valeurs entières non triées :

Écrire une fonction rechElets_TabNonTriés permettant de vérifier l'existence d'une valeur x donnée.

Algorithme :

```

FONCTION RECHELETS_TABNONTRIES (E/ T ; TABLEAU[N] D' ENTIER ,
N,X :ENTIER) : ENTIER
    I:ENTIER
DEBUT
    POUR I=0 JUSQU'A N-1 FAIRE
        SI (T[I]=X) ALORS
            RETOURNER I;
        FIN SI;
    FIN POUR;
    RETOURNER -1;
FIN;

```

Diagramme de complexité :

Complexité :

Au pire des cas : La valeur X n'existe pas dans le tableau ainsi la boucle s'itérera jusqu'à $i=N-1$ car le test ne trouvera aucun élément $T[i]=X$.

$$T(N) = \sum_{i=0}^{N-1} 1 + 1 = (N-1 + 1) + 1 = N+1 \sim O(N)$$

Implémentation : En langage C

```

long rechElets_TabNonTries(long *T,long N , long x)
{
    long i;

    for(i=0;i<N;i++)
        if(T[i]==x) return i;

    return -1;
}

```

Exécution :

Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **rechElets_TabNonTriés**).
Les valeur à chercher choisie n'existe pas afin d'obtenir une complexité maximale.

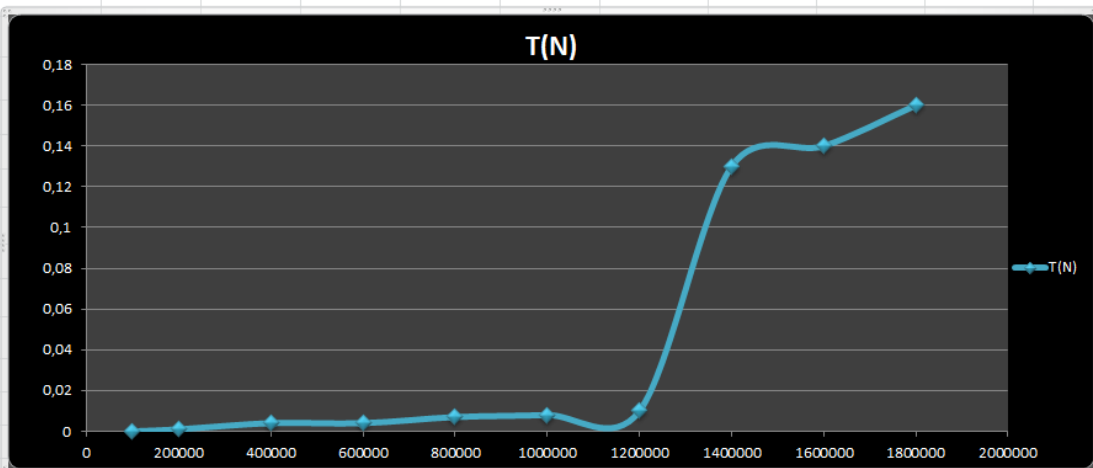
Execution de rechElets_TabNonTriés :

```
N = 100000.000000    T= 0.000000    ,position= -1
N = 200000.000000    T= 0.001000    ,position= -1
N = 400000.000000    T= 0.004000    ,position= -1
N = 600000.000000    T= 0.004000    ,position= -1
N = 800000.000000    T= 0.007000    ,position= -1
N = 1000000.000000   T= 0.008000    ,position= -1
N = 1200000.000000   T= 0.010000    ,position= -1
N = 1400000.000000   T= 0.013000    ,position= -1
N = 1600000.000000   T= 0.014000    ,position= -1
N = 1800000.000000   T= 0.016000    ,position= -1
```

Représentation Graphique :

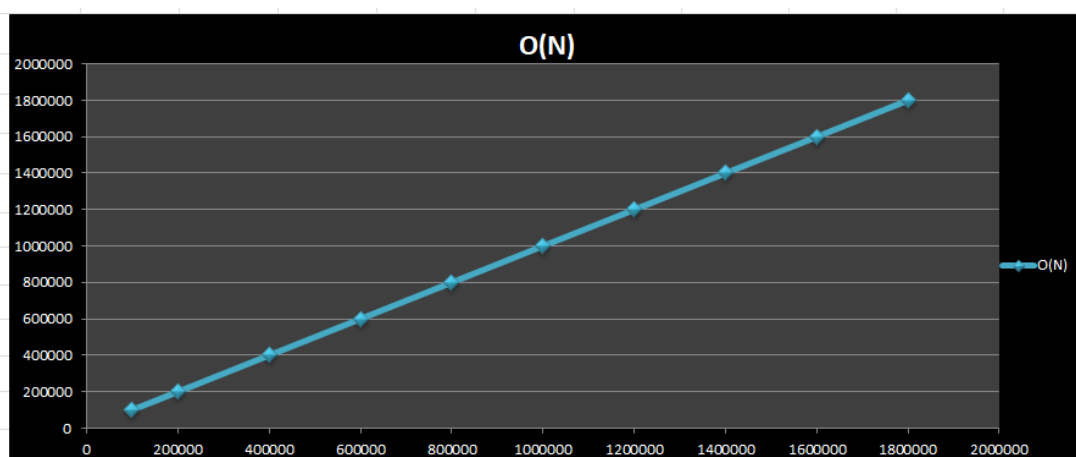
Grappe du temps d'exécution en fonction de N .

rechElets_TabNonTriés	
N	$T(N)$
100000	0
200000	0,001
400000	0,004
600000	0,004
800000	0,007
1000000	0,008
1200000	0,01
1400000	0,13
1600000	0,14
1800000	0,16



Grappe de la complexité théorique en fonction de N .

rechElets_TabNonTriés	
N	$O(N)$
100000	100000
200000	200000
400000	400000
600000	600000
800000	800000
1000000	1E+06
1200000	1E+06
1400000	1E+06
1600000	2E+06
1800000	2E+06



II. Algorithme *rechElets TabTriés*:

Soit un tableau de n ($n \geq 2$) valeurs entières triées :

Recherche séquentielle :

Écrire une fonction *rechElets_TabTriés* permettant de vérifier l'existence d'une valeur x donnée.

Algorithme :

```
FONCTION RECHELETS_TABNTRIES (E/ T:TABLEAU[N] D'ENTIER,  
    E/ N,X:ENTIER ) :ENTIER  
    I:ENTIER  
DEBUT  
    POUR I=0 JUSQU'A N-1 FAIRE  
        SI (T[I]=X) ALORS  
            RETOURNER I;  
        SINON  
            SI (T[I]>X) ALORS  
                RETOURNER -1;  
            FIN SI;  
        FIN SI;  
    FIN POUR;  
    RETOURNER -1;  
FIN;
```

$\sum_{I=0}^{N-1} 1$

1

Complexité :

Au pire des cas : La valeur X n'existe pas dans le tableau ainsi la boucle s'itérera jusqu'à $i=N-1$ car le test ne trouvera aucun élément $T[i]=X$.

$$T(N) = \sum_{i=0}^{N-1} 1 + 1 = (N-1 + 1) + 1 = N+1 \sim O(N)$$

Implémentation : En langage C

```
long rechElets_TabNTries(long *T,long N , long x)  
{  
    long i;  
  
    for(i=0;i<N;i++)  
        if(T[i]==x) return i;  
        else if(T[i]>x) return -1;  
  
    return -1;  
}
```

Exécution :

Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **rechElets_TabTriés**).

Les valeur à chercher choisie n'existe pas afin d'obtenir une complexité maximale.

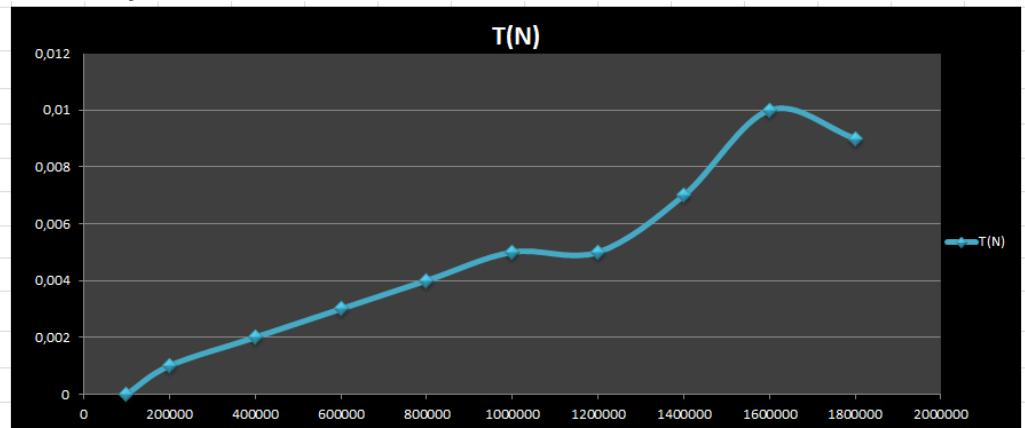
Execution de rechElets_TabNtries :

```
N = 100000.000000    T= 0.000000    ,position= -1
N = 200000.000000    T= 0.001000    ,position= -1
N = 400000.000000    T= 0.002000    ,position= -1
N = 600000.000000    T= 0.003000    ,position= -1
N = 800000.000000    T= 0.004000    ,position= -1
N = 1000000.000000   T= 0.005000    ,position= -1
N = 1200000.000000   T= 0.005000    ,position= -1
N = 1400000.000000   T= 0.007000    ,position= -1
N = 1600000.000000   T= 0.010000    ,position= -1
N = 1800000.000000   T= 0.009000    ,position= -1
```

Représentation Graphique :

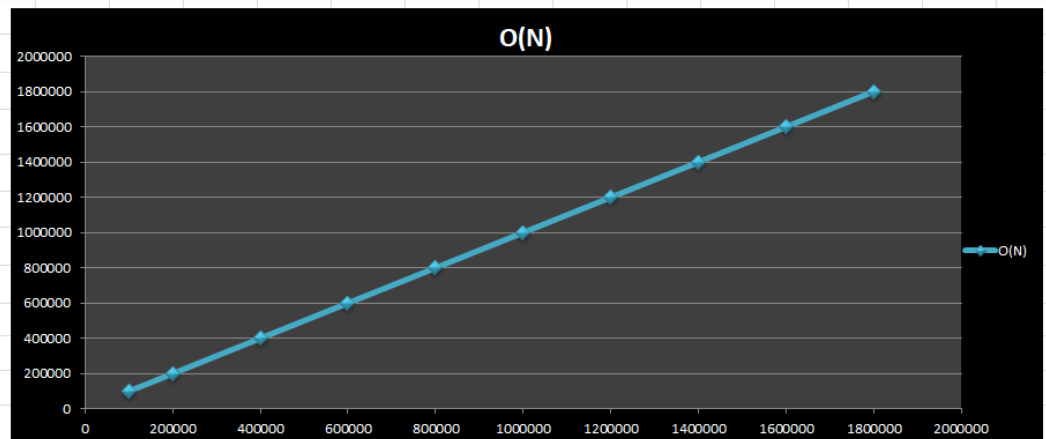
Grappe du temps d'exécution en fonction de N .

rechElets_TabTriés	
N	$T(N)$
100000	0
200000	0,001
400000	0,002
600000	0,003
800000	0,004
1000000	0,005
1200000	0,005
1400000	0,007
1600000	0,01
1800000	0,009



Grappe de la complexité théorique en fonction de N .

rechElets_TabTriés	
N	$O(N)$
100000	100000
200000	200000
400000	400000
600000	600000
800000	800000
1000000	1000000
1200000	1200000
1400000	1400000
1600000	1600000
1800000	1800000



III. Algorithme *rechElets Dicho*:

Recherche Dichotomique :

Écrire une fonction **rechElets_Dicho** permettant de vérifier l'existence d'une valeur x donnée en utilisant la méthode dichotomique.

Algorithme :

FONCTION RECHELETS_DICHO (E/ T:TABLEAU[N] D'ENTIER,

```

E/ N,X:ENTIER) : ENTIER
  D,F,M:ENTIER;
DEBUT
  SI (T[0]=X) ALORS ←
    REOUTNER 0;
  SINON
    SI (T[N-1]=X) ALORS
      RETOURNER N-1;
    SINON
      SI (X<T[0] OU X>T[N-1]) ALORS
        RETOURNER -1;
      FIN SI;
    FIN SI;
  FIN SI; ←
D=1; F=N-1;
TANT QUE (D<F) FAIRE ←
  FM= (D+F) /2
  SI (T[M]=X) ALORS
    RETOURNER M;
  SINON
    SI (X<T[M]) ALORS
      F=M;
    SINON
      D=M;
    FIN SI;
  FIN SI;
FIN TANT QUE; ←
RETOURNER -1; ←
FIN;

```

Diagram annotations:

- A bracket on the right groups the first three conditional blocks (SI (T[0]=X), SI (T[N-1]=X), and the nested SI (X<T[0] OU X>T[N-1])). This bracket is labeled with a large "1".
- A bracket on the right groups the loop body (from FM= (D+F) /2 down to FIN SI;). This bracket is labeled with $2^k = N$, $K * \text{Log}(2) = \log(N)$, and $K = \log(N) / \log(2)$.
- A bracket on the right groups the final RETURN statement (RETOURNER -1;). This bracket is labeled with a large "1".

Complexité :

$$T(N) = \frac{\log(N)}{\log(2)} + 1 + 1 = \frac{\log(N)}{\log(2)} + 2 \sim O(\log(N))$$

Implémentation : En langage C

```

long rechElets_Dicho(long *T, long N , long x)
{
    if(T[0]==x) return 0;
    else if(T[N-1]==x) return N-1;
    else if(x<T[0] || x>T[N-1]) return -1;

    long d=1,f=N-1,m;

    for(;d<f;m=(d+f)/2)
    {

```

```

    if(T[m]==x) return m;
    else
        if(x<T[m]) f=m;
        else d=m;
    }
    return -1;
}

```

Exécution :

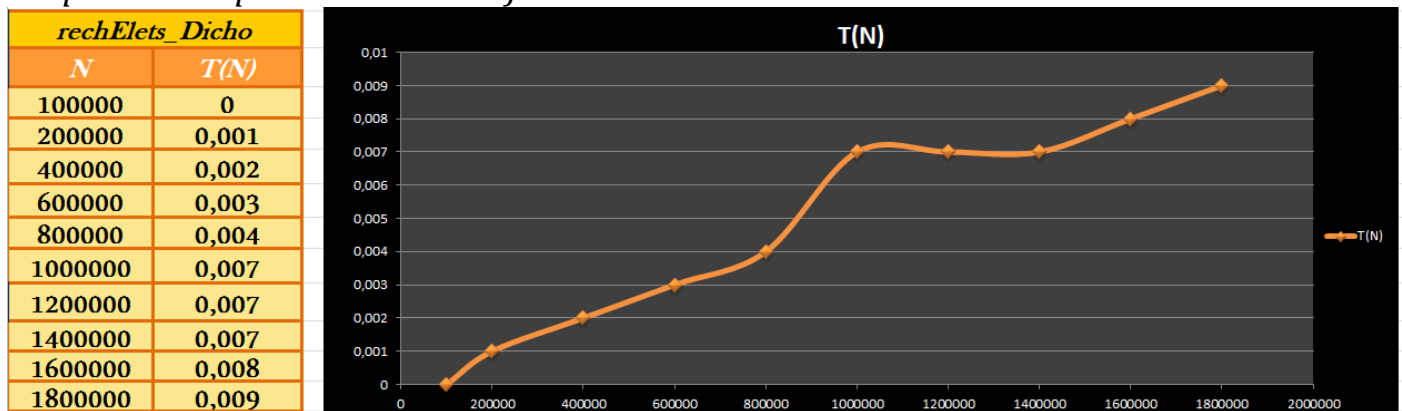
Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **rechElets_Dicho**).
Les valeur à chercher choisie n'existe pas afin d'obtenir une complexité maximale.

Execution de rechElets_Dicho:

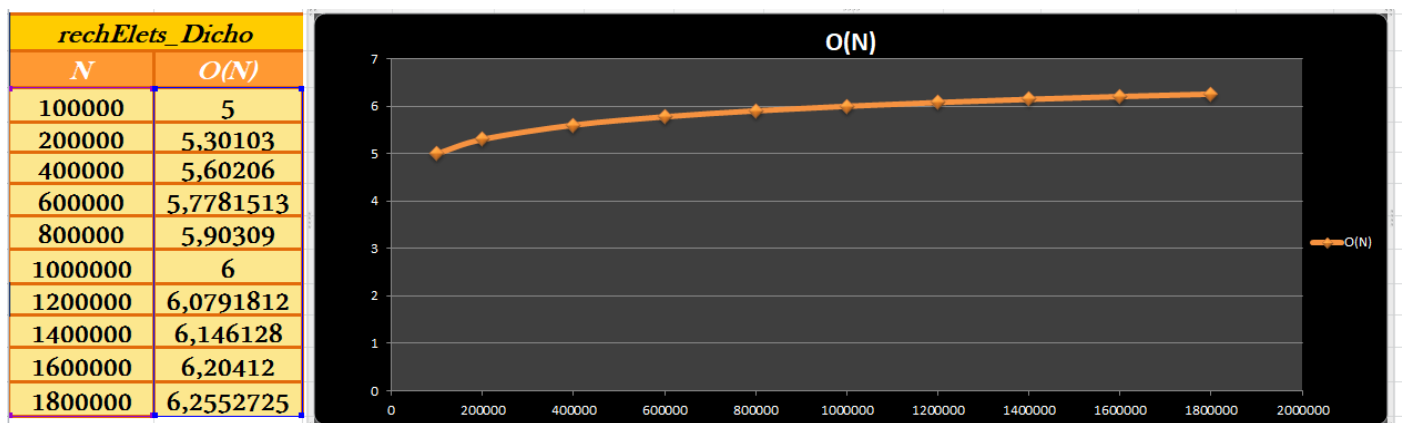
N = 100000.000000	T= 0.000000	,position= -1
N = 200000.000000	T= 0.001000	,position= -1
N = 400000.000000	T= 0.002000	,position= -1
N = 600000.000000	T= 0.003000	,position= -1
N = 800000.000000	T= 0.004000	,position= -1
N = 1000000.000000	T= 0.007000	,position= -1
N = 1200000.000000	T= 0.007000	,position= -1
N = 1400000.000000	T= 0.007000	,position= -1
N = 1600000.000000	T= 0.008000	,position= -1
N = 1800000.000000	T= 0.009000	,position= -1

Représentation Graphique :

Graphe du temps d'exécution en fonction de N .



Graphe de la complexité théorique en fonction de N .



IV. Algorithme *MaxEtMinA*:

Ecrire la fonction *MaxEtMinA* de recherche du maximum et du minimum d'un ensemble non trié de n éléments.

```
PROCEDURE MINMAXA(E/ T:TABLEAU[N] D' ENTIER,  
                  E/ N,MIN,MAX:ENTIER)
```

```
    I:ENTIER;
```

```
DEBUT
```

```
    MIN=T[0]; MAX=T[0];
```

```
    POUR I=1 JUSQU'A N FAIRE
```

```
        SI (T[I]<MIN) ALORS
```

```
            MIN=T[I];
```

```
        SINON
```

```
            SI (T[I]>MAX) ALORS
```

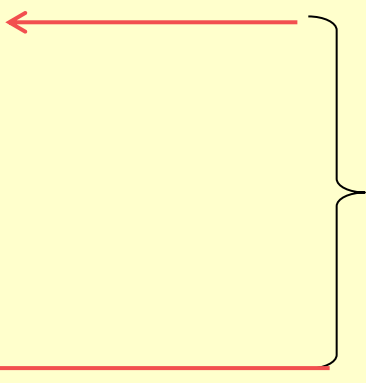
```
                MAX=T[I];
```

```
            FIN SI;
```

```
        FIN SI;
```

```
    FIN POUR;
```

```
FIN;
```


$$\sum_{I=1}^N 1$$

Complexité :

Dans tout les cas la boucle s'exécutera jusqu'au bout ainsi la boucle s'itérera jusqu'à $i=N$.

$$T(N) = \sum_1^N 1 + 1 = (N-1 + 1) + 1 = N+1 \sim O(N)$$

Implémentation : En langage C

```
void MinMaxA(long *T,long n,long *min,long*max)
{
    long i; *min=T[0]; *max=T[0];
    for(i=1;i<n;i++)
    {
        if(T[i]<*min) *min=T[i];
        else
            if(T[i]>*max) *max=T[i];
    }
}
```


Exécution :

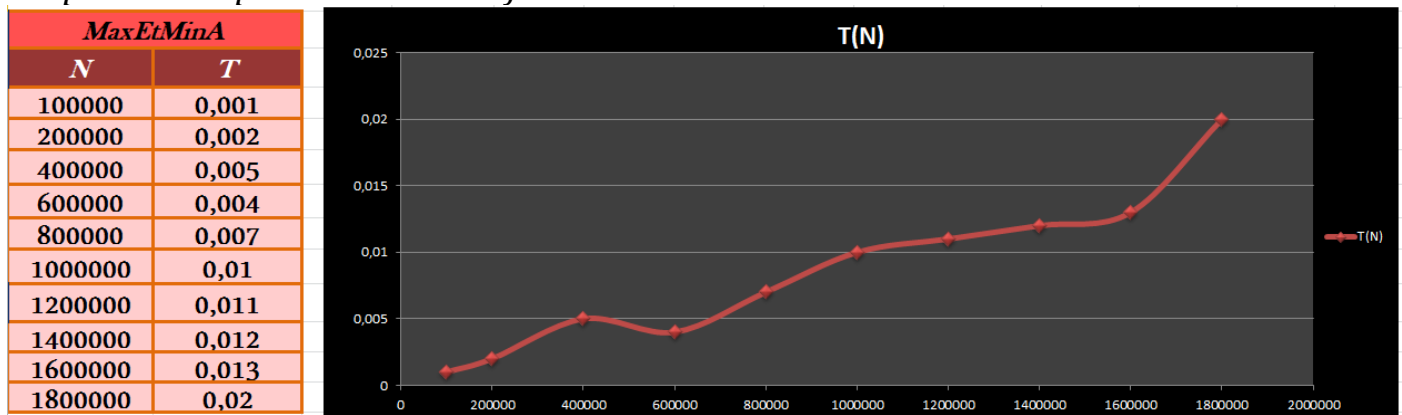
Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **MaxEtMinA**).

Execution de MinMaxA:

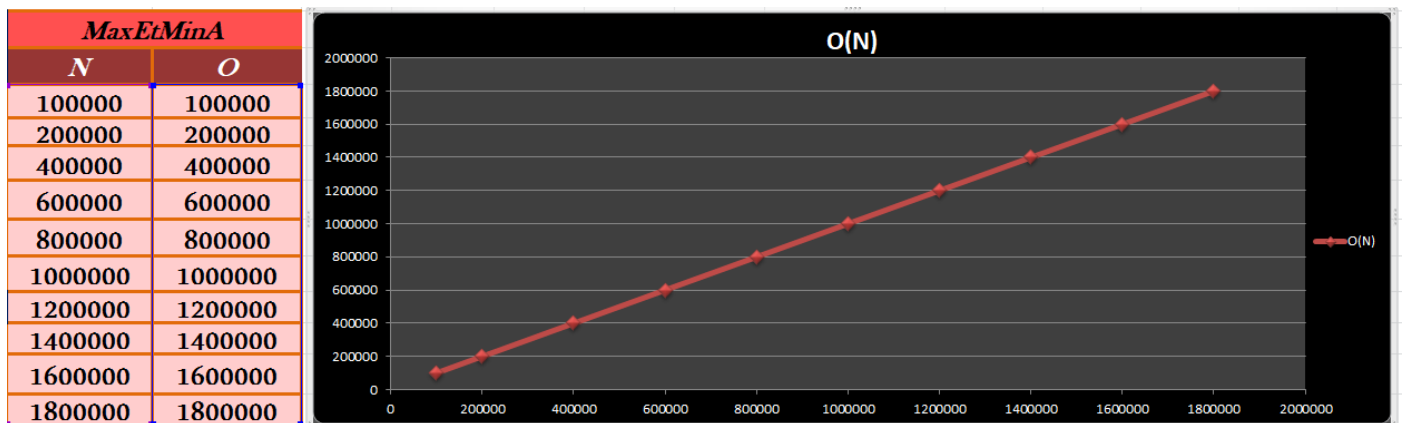
```
N = 100000.000000      T= 0.001000      ,min= 1,max=100000
N = 200000.000000      T= 0.002000      ,min= 1,max=200000
N = 400000.000000      T= 0.005000      ,min= 1,max=400000
N = 600000.000000      T= 0.004000      ,min= 1,max=600000
N = 800000.000000      T= 0.007000      ,min= 1,max=800000
N = 1000000.000000     T= 0.010000      ,min= 1,max=1000000
N = 1200000.000000     T= 0.011000      ,min= 1,max=1200000
N = 1400000.000000     T= 0.012000      ,min= 1,max=1400000
N = 1600000.000000     T= 0.013000      ,min= 1,max=1600000
N = 1800000.000000     T= 0.020000      ,min= 1,max=1800000
```

Représentation Graphique :

Graphe du temps d'exécution en fonction de N .



Graphe de la complexité théorique en fonction de N .



(*)Code Source du Programme complet :

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

void Affiche(long T[],long N)
{
```

```

    long i=0;
    printf("[");
    for(i=0;i<N-1;i++)
        printf("%d,",T[i]);
    printf("%d]\n",T[N-1]);
}

long rechElets_TabNonTries(long *T,long N , long x)
{
    long i;

    for(i=0;i<N;i++)
        if(T[i]==x) return i;

    return -1;
}
// Best : T(N) = 1 , Worst = T(N) = N
long rechElets_TabNTries(long *T,long N , long x)
{
    long i;

    for(i=0;i<N;i++)
        if(T[i]==x) return i;
        else if(T[i]>x) return -1;

    return -1;
}
// Best : T(N) = 1 , Worst = T(N) = N
long rechElets_Dicho(long *T,long N , long x)
{
    if(T[0]==x) return 0;
    else if(T[N-1]==x) return N-1;
    else if(x<T[0] || x>T[N-1]) return -1;

    long d=1,f=N-1,m;

    for(;d<f;m=(d+f)/2)
    {
        if(T[m]==x) return m;
        else
            if(x<T[m]) f=m;
            else d=m;
    }
    return -1;
}

```

```

//T(N) = O(log(N))

void MinMaxA(long *T,long n,long *min,long*max)
{
    long i; *min=T[0]; *max=T[0];
    for(i=1;i<n;i++)
    {
        if(T[i]<*min) *min=T[i];
        else
            if(T[i]>*max) *max=T[i];
    }
}

//Tableau Trié Ordre décroissant
long *PireCas(long n)
{
    long i,*T=(long *)malloc(n*sizeof(long));
    for(i=0;i<n;i++) T[i]=n-i;
    return T;
}

//Tableau Trié Ordre quelconque
long *MoyenCas(long n)
{
    long i,*T=(long *)malloc(n*sizeof(long));
    for(i=0;i<n;i++) {srand ( time(NULL) );
    T[i]= rand() % (n+1);}
    return T;
}

//Tableau Trié Ordre Croissant
long *MeilleurCas(long n)
{
    long i,*T=(long *)malloc(n*sizeof(long));
    for(i=0;i<n;i++) T[i]=i;
    return T;
}

double **Calcul_des_Temps(double **tab , long algorithme)
{
    long j,position,min,max;
    for(j=0 ; j<12 ; j++)
    {
        clock_t begin = clock();
        switch(algorithme)
        {

```

```

        case 1: position =
rechElets_TabNonTries(PireCas(tab[0][j]),tab[0][j],-1); break;
        case 2: position =
rechElets_TabNTries(MeilleurCas(tab[0][j]),tab[0][j],-1); break;
        case 3: position =
rechElets_Dicho(MeilleurCas(tab[0][j]),tab[0][j],-1); break;
        case 4:
MinMaxA(PireCas(tab[0][j]),tab[0][j],&min,&max);break;
    }
    clock_t end = clock();
    tab[1][j] = (double)(end - begin) / CLOCKS_PER_SEC;
    if(algorithme==4)
    {tab[2][j]=min;tab[3][j]=max;}
    else tab[2][j] = position;
    min=0;max=0;
}
return tab;
}

double **Tableau_de_Valeurs(void)
{
    long i ;
    double **tab;
    tab = (double **)malloc(4*sizeof(double *));
    for(i=0 ; i<4 ; i++) tab[i] = (double
*)malloc(10*sizeof(double));
    tab[0][0]=100000;
    tab[0][1]=200000;
    tab[0][2]=400000;
    tab[0][3]=600000;
    tab[0][4]=800000;
    tab[0][5]=1000000;
    tab[0][6]=1200000;
    tab[0][7]=1400000;
    tab[0][8]=1600000;
    tab[0][9]=1800000;
    for(i=0 ; i<10 ; i++)tab[1][i] = 0 ;
    return tab;
}

void Afficher_Tableau_de_Valeurs(double **tab,long minmax)
{
    long j;
    if(!minmax)
        for(j=0 ; j<10 ; j++)

```

```

        {
            printf("N = %f \t T= %f \t ,position= %d
\n",tab[0][j],tab[1][j],(int)tab[2][j]);
        }
        else
        for(j=0 ; j<10 ; j++)
        {
            printf("N = %f \t T= %f \t ,min= %d,max=%d
\n",tab[0][j],tab[1][j],(int)tab[2][j],(int)tab[3][j]);
        }
    }

int main(int argc, char *argv[])
{
    printf("Execution de rechElets_TabNonTries :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs
()),1),0);
    printf("Execution de rechElets_TabNTries :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs
()),2),0);
    printf("Execution de rechElets_Dicho:\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs
()),3),0);
    printf("Execution de MinMaxA:\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs
()),4),1);
    getchar();
    return 0;
}

```