
 <p>جامعة عبد المالك السعدي ⵜⴰⵎⴻⵔⴰⵏ ⵜⴰⵎⴰⵏⴰⵢⵜ ⵜⴰⵖⴻⵔⴰⵏⵜ Université Abdelmalek Essaadi</p>	<p>Université Abdelmalek Essaadi Ecole Nationale des Sciences Appliquées Al Hoceima</p>	 <p>ENSAH الجامعة الوطنية للعلوم التطبيقية</p>
--	---	---

Compte rendu

Des

Travaux Pratiques 2024/2025

Module :

Algorithmique avancée et complexité

Réalisé par :

Aicha chrika

Ce compte rendu sera dédié à répondre aux différentes questions et objectifs du travail pratique (TP) portant sur l'algorithme avancé. Il s'agira d'explorer les

concepts, les étapes de mise en œuvre et les résultats obtenus lors de ce TP, en fournissant des explications détaillées et en analysant les performances de l'algorithme étudié.

EXERCICE 1 :

1. Donnons la fonction récursive qui nous permet de calculer et de retourner le nieme valeur de la suite de Fibonacci:

❖ En langage C++

```
nain.cpp x
1  #include <iostream>
2  #include <ctime>
3  using namespace std;
4  int fibonacci (int n){
5  if (n<2){
6      return n;
7  }else{
8      return fibonacci(n-1)+fibonacci(n-2);
9  }
10 }
11 int main()
12 {
13     double temps_initial, temps_final, temps_cpu;
14     temps_initial= clock();
15     int op=fibonacci(20);
16     temps_final = clock();
17     temps_cpu = (temps_final - temps_initial) / CLOCKS_PER_SEC;
18     cout<<"temps d'execution est ; "<<temps_cpu<<endl;
19     cout<<"Fibo vaut : "<<op<<endl;
20     return 0;
21 }
```

2. Exécution du programme :

- Pour n=20

```
"C:\Users\Hp\Desktop\projet x + v
temps d'execution est á0
Fibo vaut :á6765

Process returned 0 (0x0)  execution time : 0.126 s
Press any key to continue.
```

- pour n=48

```
"C:\Users\Hp\Desktop\projet x + v
temps d'execution est á57.268
Fibo vaut :á512559680

Process returned 0 (0x0)  execution time : 57.373 s
Press any key to continue.
```

- Pour n=50

```
"C:\Users\Hp\Desktop\projet" × + v
temps d'execution est á165.723
Fibo vaut :á-298632863

Process returned 0 (0x0)   execution time : 165.838 s
Press any key to continue.
|
```

- Pour n=55

```
"C:\Users\Hp\Desktop\projet" × + v
temps d'execution est á1887.13
Fibo vaut :á2144908973

Process returned 0 (0x0)   execution time : 1887.259 s
Press any key to continue.
|
```

Remarque:

- ❖ Premier problème:
 - Problème :

Le temps d'exécution prend du temps ; plus on augmente la valeur de n plus le temps d'exécution pour calculer le **n-ième terme** de la suite augmente . Également après (n=55) il ne donne pas de réponse.

- Explication :

calculer la suite de Fibonacci de manière récursive naïve devient **très lent** quand n augmente, car chaque appel génère deux nouveaux appels, créant une explosion exponentielle du temps d'exécution

- Solution :

La solution la plus facile pour calculer la suite de Fibonacci efficacement, sans avoir à gérer des appels récursifs coûteux, est d'utiliser une **boucle itérative** (programmation dynamique).

- ❖ Deuxième problème :
 - Problème :

La négativité du résultat, après une certaine n donnée dans notre cas c'est 50 ; le résultat de la suite de fibonacci devient négative.

- Explication :

Le résultat de Fibonacci devient négatif à cause du **dépassement de capacité** des types entiers.

- Solution :

La création d'un nouveau type qui supporte un nombre de chiffres sans limite (Big int)

EXERCICE2 :

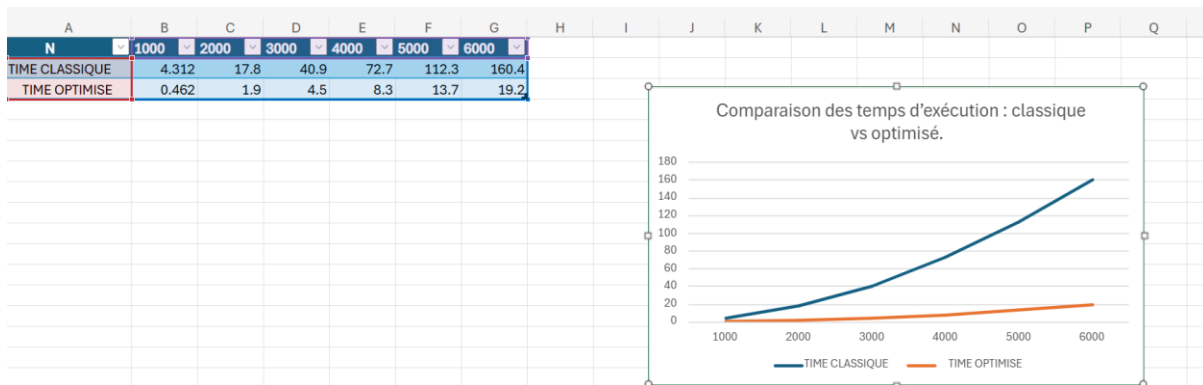
1/ la fonction de multiplication :

```
void multiplierMatrices(int** A, int** B, int** C, int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            C[i][j] = 0;
            for (int k = 0; k < n; k++)
                C[i][j] += A[i][k] * B[k][j];
        }
}
```

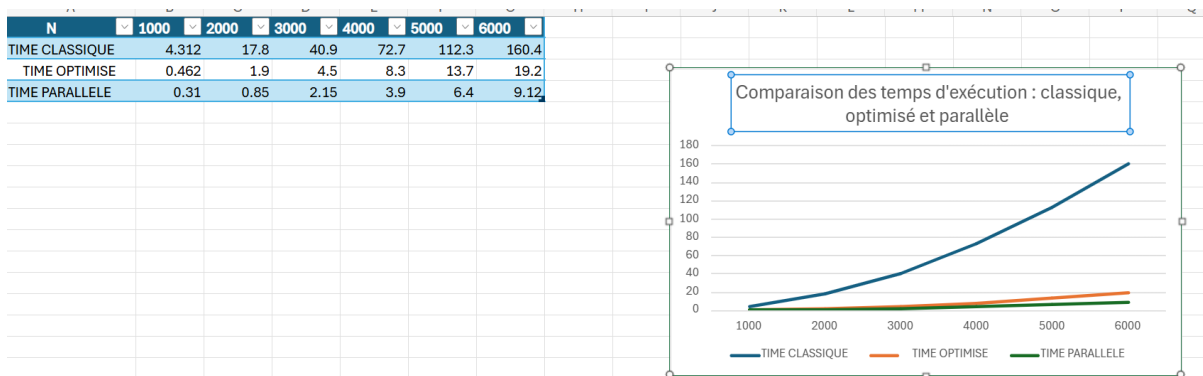
2/ fonction permettant l'optimisation du temps d'exécution

```
void multiplierOptimisee(int** A, int** B, int** C, int n) {
    for (int i = 0; i < n; ++i) {
        int* Ci = C[i];
        for (int k = 0; k < n; ++k) {
            int a = A[i][k];
            if (a != 0) {
                int* Bk = B[k];
                for (int j = 0; j < n; ++j) {
                    Ci[j] += a * Bk[j];
                }
            }
        }
    }
}
```

3/Test de comparaison des matrices de grande taille :



4/Utilisation de la programmation parallèle :



EXERCICE 3 :

Voici les algorithmes utilisés:

- Tri par sélection :

```
void triSelection(int tab[], int n) {
    int i, j, min, temp;
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++) {
            if (tab[j] < tab[min]) {
                min = j;
            }
        }
        if (min != i) {
            temp = tab[i];
            tab[i] = tab[min];
            tab[min] = temp;
        }
    }
}
```

- Tri à bulles :

```

void triBulle(int tab[], int n) {
    int i, j, temp;
    int exchange;
    for (i = 0; i < n - 1; i++) {
        exchange = 0;
        for (j = 0; j < n - i - 1; j++) {
            if (tab[j] > tab[j + 1]) {
                temp = tab[j];
                tab[j] = tab[j + 1];
                tab[j + 1] = temp;
                exchange = 1;
            }
        }
        if (!exchange) break;
    }
}

```

- Tri par insertion :

```

void triInsertion(int tab[], int n) {
    int i, cle, j;
    for (i = 1; i < n; i++) {
        cle = tab[i];
        j = i - 1;
        while (j >= 0 && tab[j] > cle) {
            tab[j + 1] = tab[j];
            j--;
        }
        tab[j + 1] = cle;
    }
}

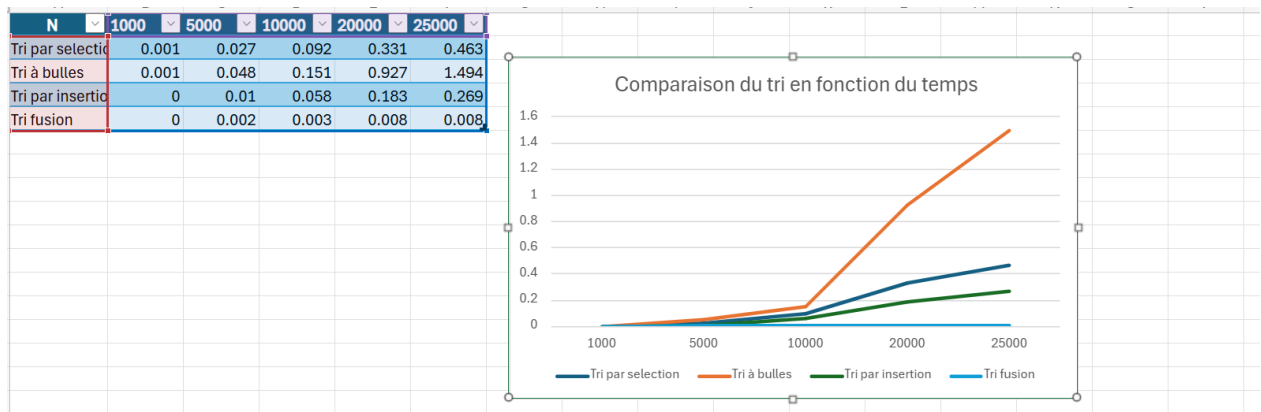
```

- Tri par fusion :

```

void triFusion(int tab[], int debut, int fin) {
    if (debut < fin) {
        int milieu = debut + (fin - debut) / 2;
        triFusion(tab, debut, milieu);
        triFusion(tab, milieu + 1, fin);
        fusion(tab, debut, milieu, fin);
    }
}

```

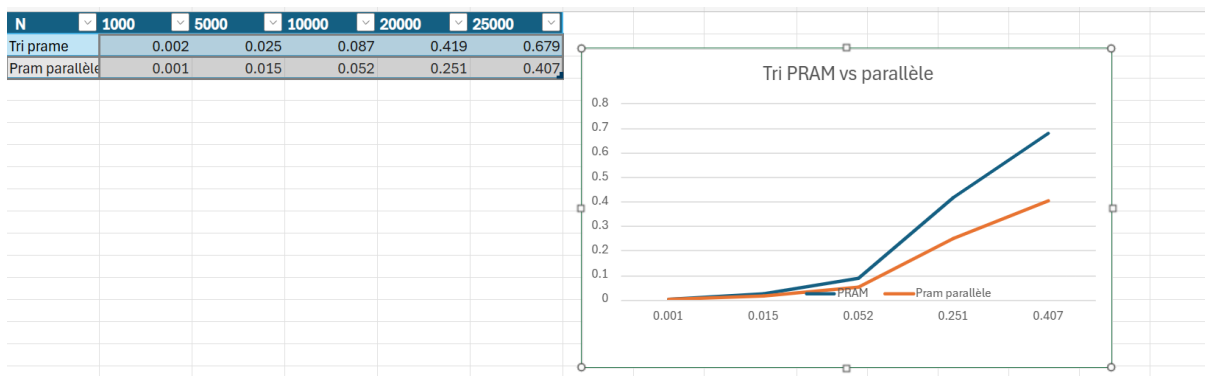


EXERCICE 4 :

Temps d'exécution de PRAM :

On observe que le **temps d'exécution augmente fortement** avec la taille du tableau, ce qui est cohérent avec la complexité **quadratique** de l'algorithme pair-impair : $O(n^2)$

Cette exécution **sans parallélisme** montre clairement les limites de performance du modèle PRAM lorsqu'il est simulé de manière séquentielle, notamment pour de grandes tailles N.



On peut conclure que ->L'utilisation d'OpenMP permet d'accélérer l'exécution du tri PRAM de manière notable. Le gain en performance devient de plus en plus significatif à mesure que la taille du tableau augmente. Cela illustre concrètement l'intérêt du calcul parallèle dans les environnements multi-cœurs.