Compte rendu SAE S1.02

Ainhoa Gosselin
Jordan Dupuy

Comparaison d'approches algorithmiques

Dans cette SAE le but de notre travail est de trier les éléments d'un tableau par ordre croissant. Pour arriver au meilleur résultat possible nous avons testé différents algorithmes de tri, comme le tri par sélection, le tri à bulles etc.. Grâce à ce travail de comparaison entre tous ces algorithmes nous pourrons déterminer lequel est le plus efficace en fonction de leur vitesse d'exécution et de leur quantité de mémoire nécessaire.

1) Dans un premier temps nous avons commencé en créant un algorithme de tri par sélection. Sa fonction se nomme **triSelection**.
   On l'a initialisé dans le fichier fonctions.h avec cette ligne de code :

```
unsigned int triSelection(std::vector<int>& tab);
```

Ici nous avons notre programme dans notre **ConsoleApplication1.cpp** :

```
int main()
{
    std::vector<int> tab = initTabAleat(150);
    std::cout << triSelection(tab) << "\n";

    verifTri(tab);

    std::vector<int> tab1 = initTabPresqueTri(501);
    std::cout << triSelection(tab1) << "\n";

    verifTri(tab1);


    std::vector<int> tab2 = initTabPresqueTriDeb(501);
    std::cout << triSelection(tab2) << "\n";

    verifTri(tab2);

    std::vector<int> tab3 = initTabPresqueTriFin(501);
    std::cout << triSelection(tab3) << "\n";

    verifTri(tab3);

    std::vector<int> tab4 = initTabPresqueTriDebFin(501);
    std::cout << triSelection(tab4) << "\n";

    verifTri(tab4);
```

Ici nous avons notre programme dans notre fichier **fonctions.cpp** :

```
unsigned int triSelection(std::vector<int>& tab)
{
    size_t n = tab.size();
    unsigned int N = 0;

    for (size_t i = 0; i < n - 1; ++i)
    {
        size_t mini = i;

        for (size_t j = i + 1; j < n; ++j)
        {
            N++;
            if (tab[j] < tab[mini])
            {
                mini = j;
            }
        }

        if (mini != i)
        {
            std::swap(tab[i], tab[mini]);
        }
    }

    return N;
}
```

2) Dans un deuxième temps nous avons étudié les performances de ce tri. Tout d'abord nous avons fait en sorte que la taille du tableau soit aléatoire. De plus nous avons ajouté un un compteur qui nous permet de savoir combien de comparaisons il a été nécessaire pour pouvoir tout trier.
Nous pouvons apercevoir dans le fichier **fonctions.cpp**, à la ligne 4 ; 17 et 19 ce que nous avons rajouté pour avoir notre compteur :

Ensuite nous devions créer un fichier CSV donc pour ça on a initialisé la fonction dans le fichier **fonctions.h** :

```
void writeToCSV(const std::string& filename, const std::vector<std::vector<unsigned int>>& comparaisons, const std::vector<size_t>& tailles);
```

Par la suite nous avons écrit son programme dans le fichier **fonctions.cpp** :

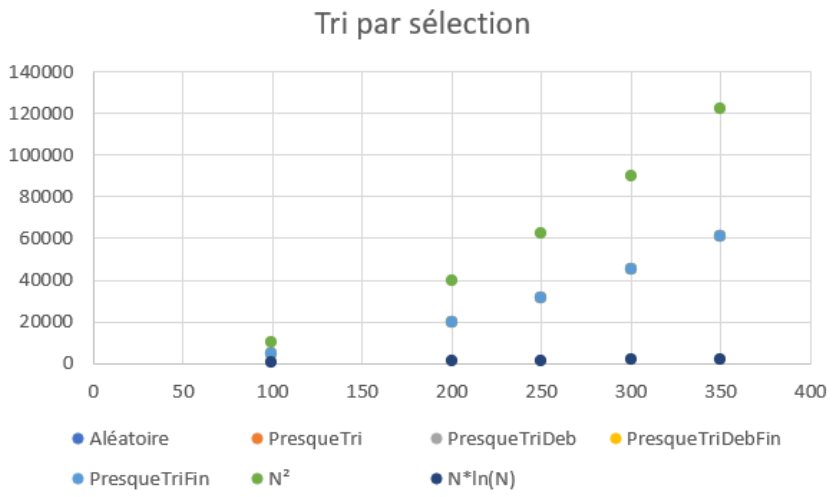
```
void writeToCSV(const std::string& filename, const std::vector<std::vector<unsigned int>>& comparaisons, const std::vector<size_t>& tailles)
{
    std::ofstream csvFile(filename);
    if (csvFile.is_open())
    {
        // Write header
        csvFile << "N;Aleat Select;PresqueTri Select;PresqueTriDeb Select;PresqueTriFin Select;PresqueTriDebFin Select;Aleat Bulle;PresqueTri Bulle;

        // Write data
        for (size_t i = 0; i < comparaisons.size(); ++i)
        {
            csvFile << tailles[i] << ";";
            for (size_t j = 1; j < comparaisons[i].size(); ++j)
            {
                csvFile << comparaisons[i][j] << ";";
            }
            csvFile << "\n";
        }

        csvFile.close();
    }
    else
    {
        std::cerr << "Le fichier ne s'ouvre pas.\n";
    }
}
```

ConsoleApplication1.cpp :

```
const std::string csvFilename = "comparaisons.csv";
```

```
// Écriture des résultats dans le fichier CSV
writeToCSV(csvFilename, comparaisons, N);

std::cout << "Le resultat des comparaisons est ecrit dans le fichier : " << csvFilename << std::endl;

return 0;
```

Enfin nous devons tracer les courbes du nombre de comparaisons en fonction de la taille du tableau :



3) Dans un troisième temps nous devions répéter ces étapes pour différents tris.

- Le tri à bulle

Dans le fichier **ConsoleApplication1.cpp** :

```
std::vector<int> tab5 = initTabAleat(N[i]);
comparaisons[i][6] = triBulles(tab5);
verifTri(tab5);

std::vector<int> tab6 = initTabPresqueTri(N[i]);
comparaisons[i][7] = triBulles(tab6);
verifTri(tab6);

std::vector<int> tab7 = initTabPresqueTriDeb(N[i]);
comparaisons[i][8] = triBulles(tab7);
verifTri(tab7);

std::vector<int> tab8 = initTabPresqueTriFin(N[i]);
comparaisons[i][9] = triBulles(tab8);
verifTri(tab8);

std::vector<int> tab9 = initTabPresqueTriDebFin(N[i]);
comparaisons[i][10] = triBulles(tab9);
verifTri(tab9);
```

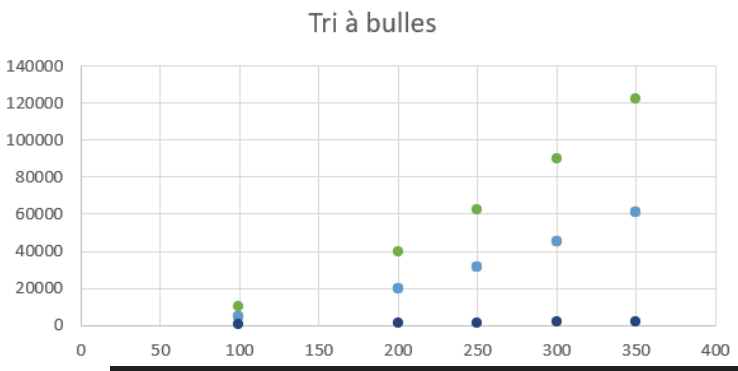
```
unsigned int triBulles(std::vector<int>& tab)
{
    size_t n = tab.size();
    unsigned int N = 0;

    for (size_t i = n - 1; i > 0; --i)
    {
        for (size_t j = 0; j < i; ++j)
        {
            N++;
            if (tab[j + 1] < tab[j])
            {
                std::swap(tab[j], tab[j + 1]);
            }
        }
    }

    return N;
}
```

←---------- Dans le fichier **fonctions.cpp**

Dans le fichier **fonctions.h** :

```
unsigned int triBulles(std::vector<int>& tab);
```

Graphique à partir du fichier CSV :



- Le tri à bulle optimisé :

```
std::vector<int> tab10 = initTabAleat(N[i]);
comparaisons[i][11] = triBullesOptimise(tab10);
verifTri(tab10);

std::vector<int> tab11 = initTabPresqueTri(N[i]);
comparaisons[i][12] = triBullesOptimise(tab11);
verifTri(tab11);

std::vector<int> tab12 = initTabPresqueTriDeb(N[i]);
comparaisons[i][13] = triBullesOptimise(tab12);
verifTri(tab12);
```

Dans le fichier **ConsoleApplication1.cpp** :

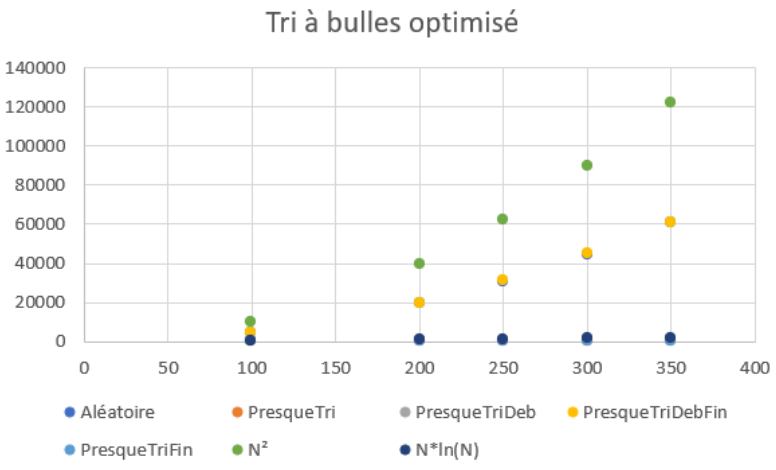
```cpp
unsigned int triBullesOptimise(std::vector<int>& tab)
{
    size_t n = tab.size();
    unsigned int N = 0;

    for (size_t i = n - 1; i > 0; --i)
    {
        bool tableauTriee = true;

        for (size_t j = 0; j < i; ++j)
        {
            N++;
            if (tab[j + 1] < tab[j])
            {
                std::swap(tab[j], tab[j + 1]);
                tableauTriee = false;
            }
        }

        if (tableauTriee)
            break;
    }

    return N;
}
```

←---------- Dans le fichier **fonctions.cpp**

Dans le fichier **fonctions.h** :

```cpp
unsigned int triBullesOptimise(std::vector<int>& tab);
```

Graphique à partir du **fichier CSV** :



- Le tri à peigne

```cpp
std::vector<int> tab15 = initTabAleat(N[i]);
comparaisons[i][16] = triPeigne(tab15);
verifTri(tab15);

std::vector<int> tab16 = initTabPresqueTri(N[i]);
comparaisons[i][17] = triPeigne(tab16);
verifTri(tab16);

std::vector<int> tab17 = initTabPresqueTriDeb(N[i]);
comparaisons[i][18] = triPeigne(tab17);
```

Dans le fichier **ConsoleApplication1.cpp** :

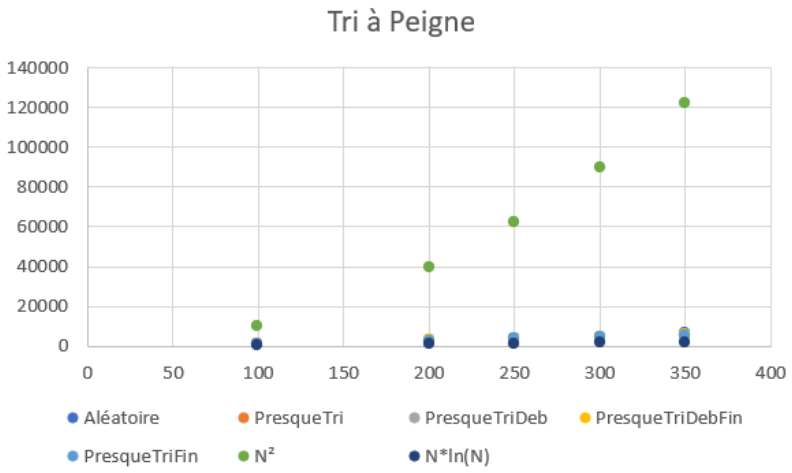
```cpp
unsigned int triPeigne(std::vector<int>& tab)
{
    size_t n = tab.size();
    unsigned int N = 0;

    size_t gap = n;
    bool swapped = true;

    while (gap > 1 || swapped)
    {
        gap = (gap * 10) / 13;

        if (gap < 1)
            gap = 1;

        swapped = false;

        for (size_t i = 0; i < n - gap; ++i)
        {
            size_t j = i + gap;
            N++;

            if (tab[i] > tab[j])
            {
                std::swap(tab[i], tab[j]);
                swapped = true;
            }
        }
    }

    return N;
}
```

←---------- Dans le fichier **fonctions.cpp**

Dans le fichier **fonctions.h** :

```cpp
unsigned int triPeigne(std::vector<int>& tab);
```

Graphique à partir du **fichier CSV** :



- Le tri rapide (short)

```cpp
std::vector<int> tab20 = initTabAleat(N[i]);
comparaisons[i][21] = triRapideShort(tab20);
verifTri(tab20);

std::vector<int> tab21 = initTabPresqueTri(N[i]);
comparaisons[i][22] = triRapideShort(tab21);
verifTri(tab21);

std::vector<int> tab22 = initTabPresqueTriDeb(N[i]);
```

Dans le fichier **ConsoleApplication1.cpp** :

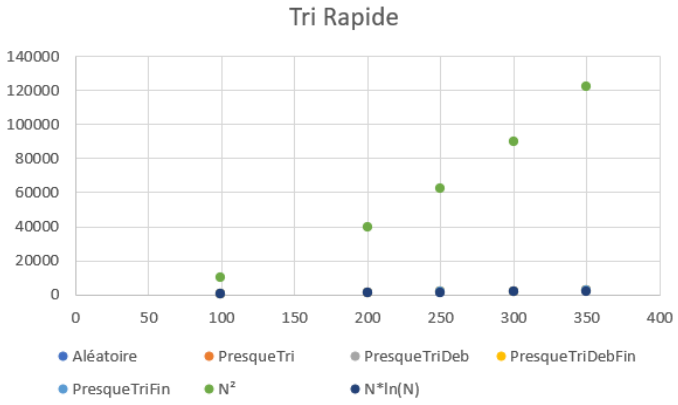
```cpp
unsigned int triRapideShort(std::vector<int>& tab) {
    int N = triRapide(tab, 0, tab.size()-1);
    return N;
}

unsigned int triRapide(std::vector<int>& tab, int debut, int fin) {
    unsigned int N = 0;

    if (debut < fin) {
        int pivotIndex = debut + (fin - debut) / 2;
        int pivot = tab[pivotIndex];
        int i = debut;
        int j = fin;
        while (i <= j) {
            while (tab[i] < pivot) {
                i++;
                N++;
            }
            while (tab[j] > pivot) {
                j--;
                N++;
            }

            if (i <= j) {
                std::swap(tab[i], tab[j]);
                i++;
                j--;
            }
        }
        N += triRapide(tab, debut, j);
        N += triRapide(tab, i, fin);
    }
    return N;
}
```

←---------- Dans le fichier **fonctions.cpp**

Dans le fichier **fonctions.h** :

```cpp
unsigned int triRapideShort(std::vector<int>& tab);

unsigned int triRapide(std::vector<int>& tab, int debut, int fin);
```

Graphique à partir du **fichier CSV** :



Explication triRapideShort et triRapide

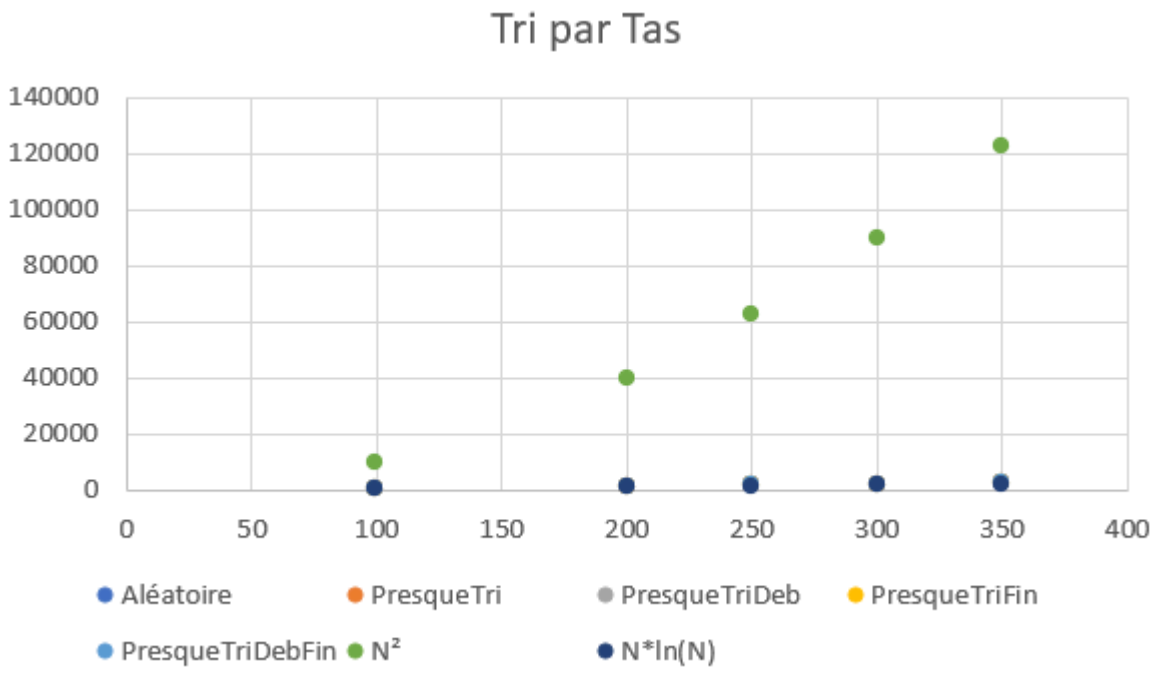
Pour une fonction normale de tri rapide il faut donner 3 valeurs sauf que pour chaque fonction on en donné qu'une par conséquent on à créer une autre fonction tri Rapide short dans laquelle on inscrit qu'une valeur mais qui reprend le code basique de tri Rapide.

4) Dans un quatrième temps, nous avons conclu sur quel type d'algorithme de tri est préférable en fonction des cas de tableaux à trier. Mais aussi si certains algorithmes se démarquent des autres.

Pour le tri par sélection, le tri à bulle et le tri à bulle optimisé on constate que plus N*ln(N) augmente et plus le temps de tri augmente. Néanmoins cela ce produit moins avec le tri à peigne et le tri rapide. Malheureusement pour le tri à peigne, il peine à suivre la cadence, Par conséquent c le tri rapide qui s'adapte le mieux à l'augmentation de N*ln(N).

5) Dans un dernier temps, nous avons rajouté des algorithmes de tri :

- Le tri par tas :



```cpp
unsigned int tamiser(std::vector<int>& tab, size_t noeur, size_t n);

unsigned int triParTas(std::vector<int>& tab);
```

Fonctions.cpp :

```cpp
unsigned int tamiser(std::vector<int>& tab, size_t noeur, size_t n) {
    size_t k = noeur;
    size_t j = 2 * k;

    unsigned int N = 0;  // Variable pour compter le nombre de comparaisons

    while (j <= n) {
        if (j < n && tab[j - 1] < tab[j]) {
            j = j + 1;
        }

        if (tab[k - 1] < tab[j - 1]) {
            std::swap(tab[k - 1], tab[j - 1]);
            k = j;
            j = 2 * k;
            N++;  // Incrémente le compteur de comparaisons
        }
        else {
            j = n + 1;
        }
    }

    return N;
}

unsigned int triParTas(std::vector<int>& tab) {
    size_t longueur = tab.size();

    unsigned int N = 0;  // Variable pour compter le nombre de comparaisons

    // Construction du tas initial
    for (size_t i = longueur / 2; i >= 1; --i) {
        N += tamiser(tab, i, longueur);
    }

    // Extraction successive des éléments du tas
    for (size_t i = longueur; i >= 2; --i) {
        std::swap(tab[0], tab[i - 1]);
        N += tamiser(tab, 1, i - 1);
    }

    return N;
}
```

```
std::vector<int> tab25 = initTabAleat(N[i]);
comparaisons[i][26] = triParTas(tab25);
verifTri(tab25);

std::vector<int> tab26 = initTabAleat(N[i]);
comparaisons[i][27] = triParTas(tab26);
verifTri(tab26);

std::vector<int> tab27 = initTabAleat(N[i]);
comparaisons[i][28] = triParTas(tab27);
verifTri(tab27);

std::vector<int> tab28 = initTabAleat(N[i]);
comparaisons[i][29] = triParTas(tab28);
verifTri(tab28);

std::vector<int> tab29 = initTabAleat(N[i]);
comparaisons[i][30] = triParTas(tab29);
verifTri(tab29);
```

Sources :

https://fr.wikipedia.org/wiki/Tri_par_s%C3%A9lection
https://fr.wikipedia.org/wiki/Tri_%C3%A0_bulles
https://fr.wikipedia.org/wiki/Tri_%C3%A0_peigne
https://fr.wikipedia.org/wiki/Tri_rapide
https://fr.wikipedia.org/wiki/Algorithme_de_tri
https://fr.wikipedia.org/wiki/Tri_fusion
https://fr.wikipedia.org/wiki/Tri_par_tas
https://fr.wikipedia.org/wiki/Tri_de_Shell