

IMAC

TP Algorithmique 1

Tris

Ce TP a pour but d'implémenter différent type de tri afin d'ordonner un ensemble de nombre dans l'ordre croissant.

1 Quelques tris

1.1 Tri par sélection

L'algorithme par "force brute" pour trier un tableau est le tri par sélection. Le principe est tout simplement de chercher à chaque fois l'élément le plus petit du tableau pour le placer au début.

Algorithm 1 Selection Sort

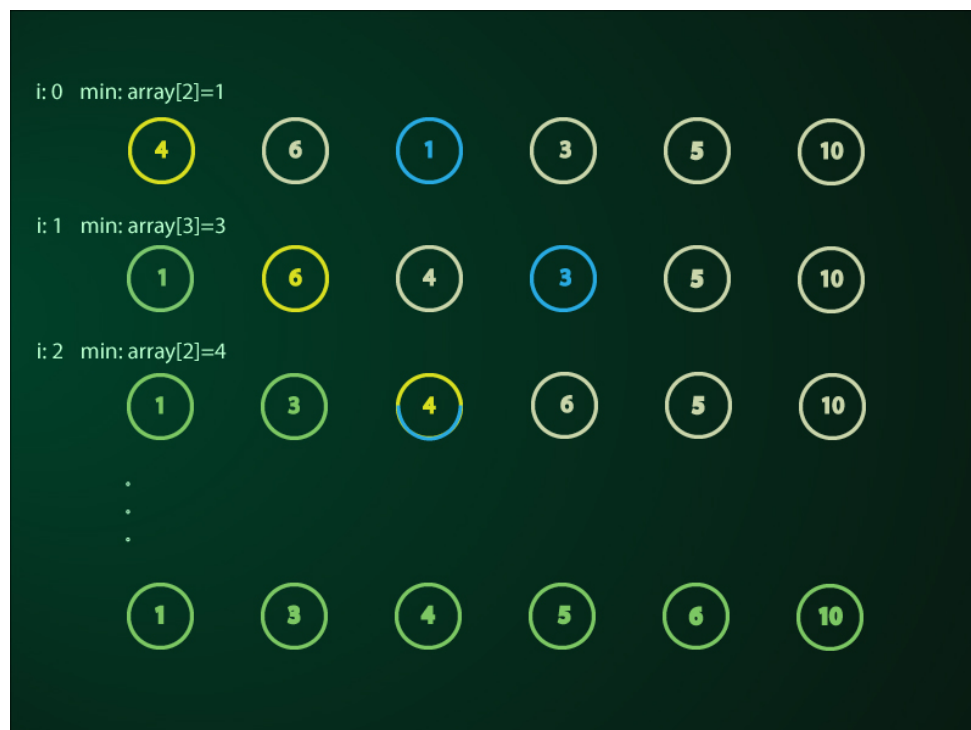
$t \leftarrow$ tableau de nombre aléatoire

Pour chaque indice i de t **faire**

 chercher le minimum à partir de i

 inverser le minimum et la case courante

fin Pour



1.2 Tri par insertion

Une autre version de la "force brute" est le tri par insertion. Le tri consiste à déplacer les valeurs d'un tableau à un autre en déterminant à chaque fois où doit se positionner le nombre par rapport aux nombres déjà présent dans le deuxième tableau.

Algorithm 2 Insertion Sort

$t \leftarrow$ tableau de nombre aléatoire

$result \leftarrow$ tableau vide

$result[0] \leftarrow t[0]$

Pour chaque nombre n de t (excepté le premier) **faire**

Si $\exists m \in result$ tel que $m > n$ **Alors**

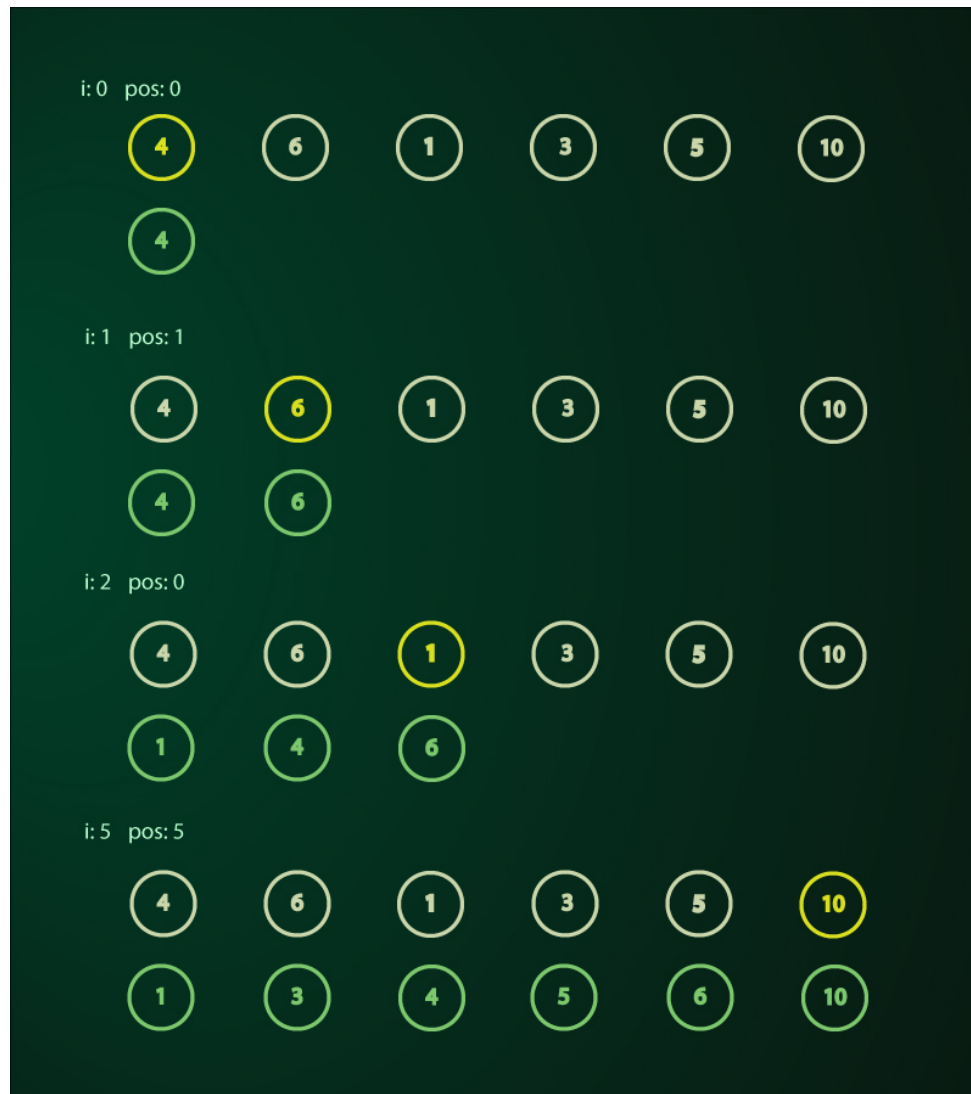
 insérer n à la position de m (en décalant donc le reste du tableau)

Sinon

 insérer n à la fin

fin Si

fin Pour



1.3 Tri à bulles

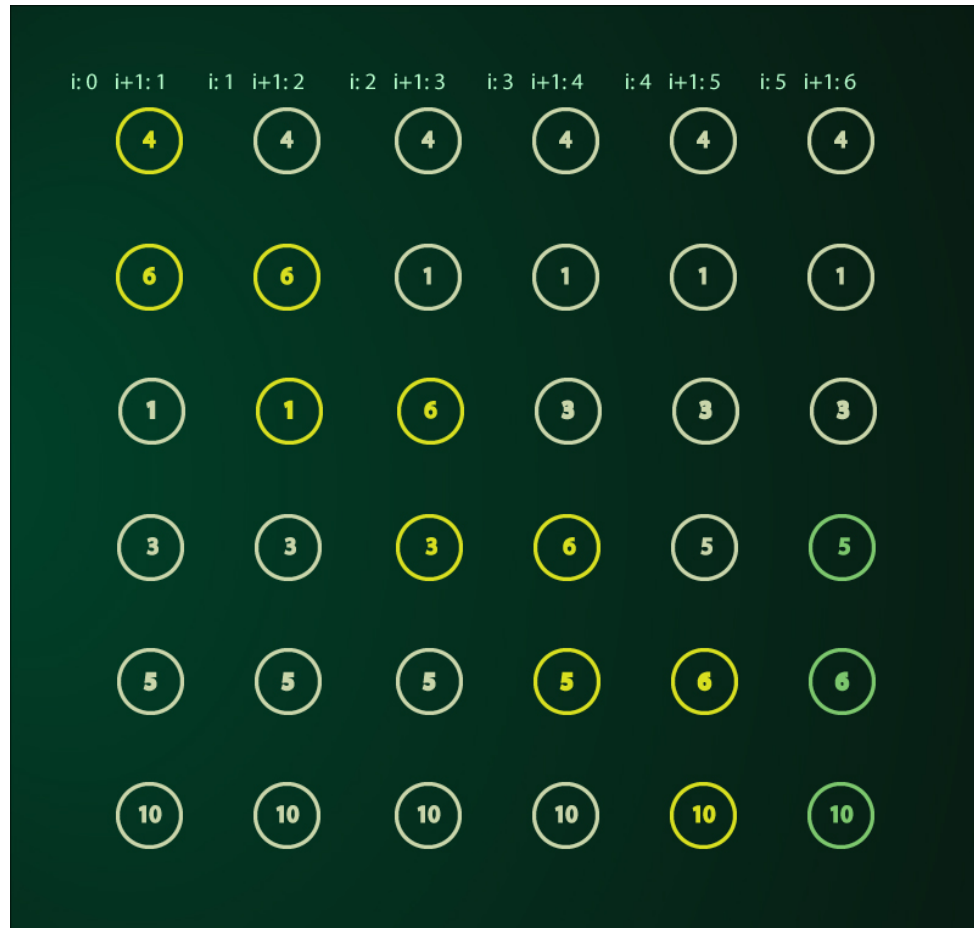
Le tri à bulle consiste à tester chaque paire de cases adjacentes et de faire remonter les valeurs les plus grandes comme des bulles d'air dans de l'eau.

Algorithm 3 Bubble Sort

```

 $t \leftarrow$  tableau de nombre aléatoire
Pour chaque indice  $i$  de  $t$  faire
  Pour chaque cases adjacentes faire
    remonter la valeur la plus haute
  fin Pour
fin Pour

```



1.4 Tri rapide

Le tri rapide se base sur le principe de "diviser pour régner". Le but est de diviser le tableau en deux parties, les petites et les grandes valeurs, puis de réitérer jusqu'à n'avoir qu'un seul élément par tableau. En fusionnant les tableaux on obtient un tableau trié.

Algorithm 4 Quick Sort

$t \leftarrow$ tableau de nombre aléatoire
 $pivot \leftarrow$ un nombre quelconque de t
 $lowers \leftarrow [n \in t \text{ tel que } n < pivot]$
 $greater \leftarrow [n \in t \text{ tel que } n > pivot]$
 trier $lowers$ et $greater$
 fusionner $lowers$, $pivot$ et $greater$



1.5 Tri par fusion

Le tri par fusion est semblable au tri rapide, il divise le tableau en deux parties pour faciliter le rangement. Mais celui-ci range les valeurs non pas à la division mais au moment de la fusion. On divise le tableau en plusieurs petite partie. Lors de la fusion on vérifie laquelle des plus petites valeurs des deux parties va être rangé en premier. On réitère le processus jusqu'à ré obtenir le tableau initial trié. Lorsque vous rangez le minimum de $t1$ et $t2$, n'oubliez de mettre à le jour le minimum des

Algorithm 5 Quick Sort

$t \leftarrow$ tableau de nombre aléatoire

$t1 \leftarrow$ première moitié du t

$t2 \leftarrow$ deuxième moitié t

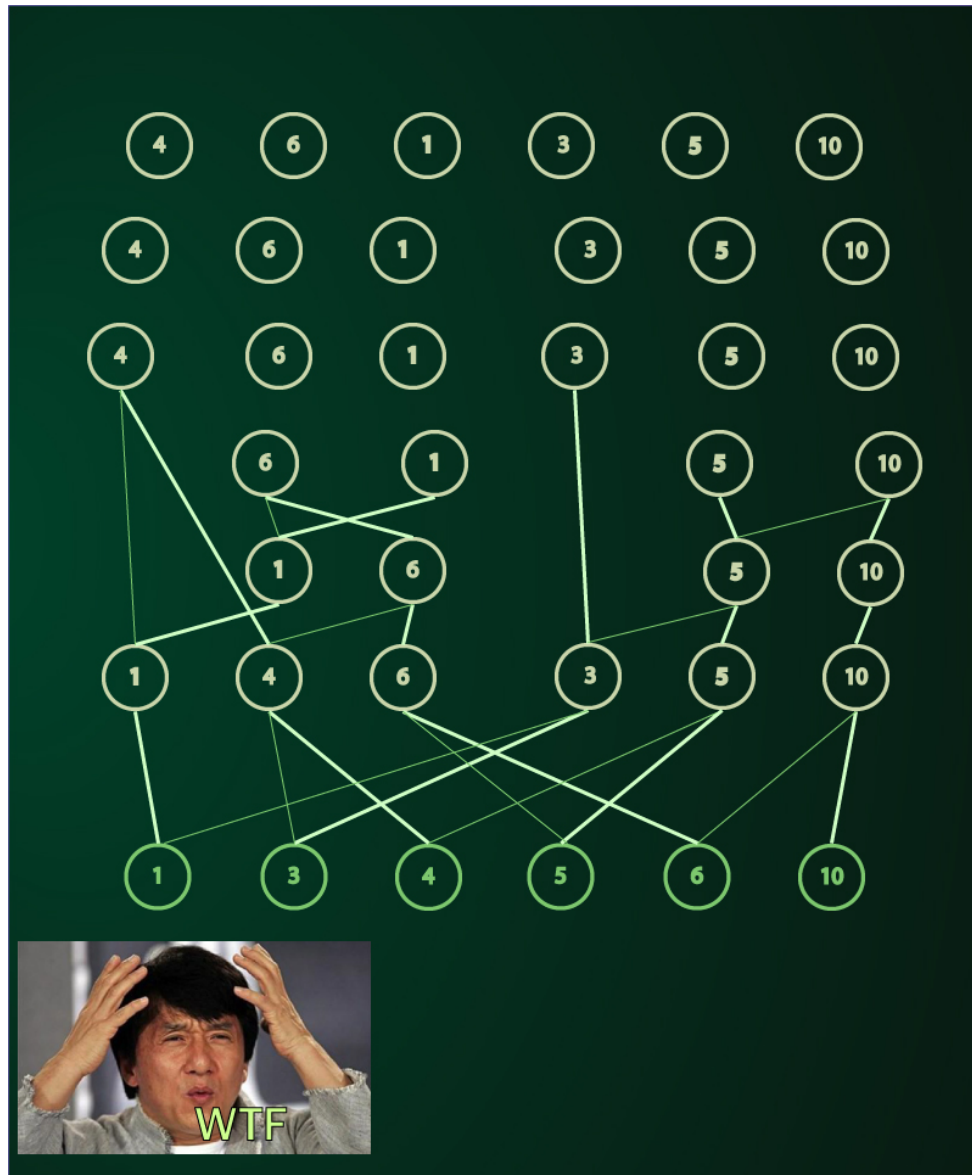
diviser et trier $t1$ et $t2$

Pour chaque indice i de t **faire**

$t[i] \leftarrow \min(t1, t2)$

fin Pour

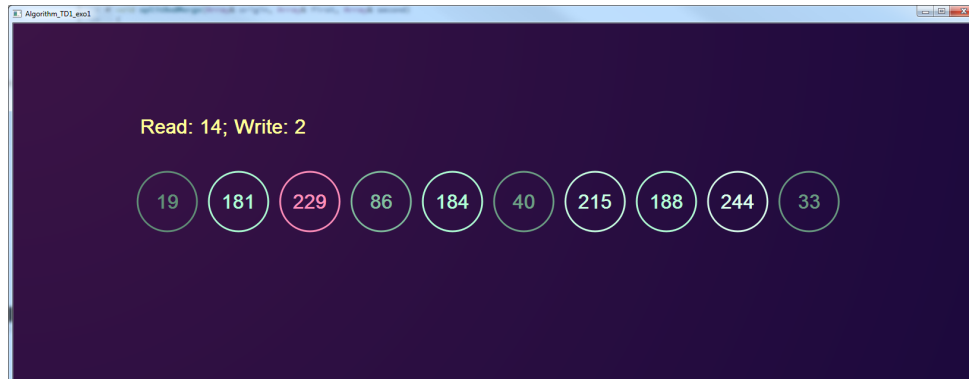
tableaux. Un exemple sera peut-être plus concret !



Bon soit, ça porte à confusion... Vous pouvez constater que chaque case de la deuxième partie pointe vers deux case du niveau inférieur, cela correspond au test effectué entre les deux tableaux que l'on fusionne. La branche claire représente le rangement de la valeur minimum, cette valeur ne rentre donc plus en compte pour les tests suivants.

2 TP

Vous allez compléter un programme qui lance un tri sur tableau d'entier affichant le processus.



Ce programme vous affiche chaque étape de l'algorithme. Les cases en magenta indiquent une lecture, celles en bleu/violet indiquent une écriture. Le dossier `Algorithme1` contient des fichiers PRO, il s'agit d'un fichier décrivant un projet Qt. Grossièrement il s'agit un framework qui permet de faire des interfaces (mais il fait beaucoup plus que ça !). Lancez le avec QtCreator, l'IDE fournit par Qt, il vous permet de compiler et lancer un programme facilement.

Chaque projet contient un fichier `exo<i>.pro` et un fichier `exo<i>.cpp`. Remplissez la fonction de tri de chaque `exo<i>.cpp` pour trier un tableau passé en paramètre. Ce tableau de type *Array* implémente déjà des fonctions d'accès et de modification tel que `get()`, `insert()`, `set()`...

```
void sort(Array& toSort)
{
    int firstNumber = toSort.get(0); // get the first number of the array toSort
    int lastNumber = toSort.get(toSort.size()-1); // get the last number of the
        array toSort
    if (firstNumber > lastNumber)
        toSort.swap(0, toSort.size()-1); // swap between the first index and the last
    printf("I think it's sorted");
}

Array array(int size); // Array array(10) --> make an array of 10 numbers
    initialized to -1;
array.get(int index); // array.get(2) --> get the number at index 2
array[int index]; // equivalent to get()
array.set(int index, int value); // array.set(2, 10) --> set 10 into the 2nd
    case of array
array.swap(int index1, int index2); // array.swap(2,5) --> swap the 2nd and the
    5th case of array
array.insert(int index, int value); // array.set(2, 10) --> insert 10 into the 2
    nd case of array by shifting the all numbers other numbers
```

La fonction `main` est déjà implémenté pour lancer une fenêtre qui tri un ensemble d'entier.

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Array::instruction_duration = 200;
    MainWindow w(insertionSort, 10);
```

```

    w.show();

    return a.exec();
}

```

Le deuxième paramètre de *MainWindow* définit le nombre d'élément dans le tableau. *instruction_duration* est une variable globale affilié à la structure/classe *Array* qui définit la durée de chaque accès mémoire, modifiez le pour voir le résultat plus ou moins rapidement. Une fois votre fonction *sort* est implémenter, cliquer sur exécuter (l'icone lecture) pour lancer le programme. Si votre programme est bon, vous devriez obtenir ce genre de résultat.



Pour certains exercices, il vous faudra créer d'autre tableau intermédiaire. Vous pouvez utiliser l'un des deux moyens suivants :

```

Array a(toSort.size());
Array& a = w->newArray(toSort.size()); // if you want the MainWindow to
    display the new array

```

3 Notes

- *MainWindow* est une classe dérivée de *QMainWindow* une classe de Qt qui représente la fenêtre principale. *MainWindow* a été implémenté pour afficher une scène qui dessine des *GraphicsItems* pour afficher les tableaux. Vous retrouvez notamment la fonction *newArray* qui a été implémenté pour la classe *MainWindow* mais qu'on ne retrouvera donc pas dans une *QMainWindow* classique.
- Il était nécessaire pour le programme que *w* de type *MainWindow* soit un pointer. Pour appeler une méthode sur pointer, il faut faire *w->method("Yolo")*