|  |  |
| --- | --- |
| Ponctuel 2 | **Les algorithmes de tri** |

|  |  |  |  |
| --- | --- | --- | --- |
| Nom : |  | Groupe : |  |

1. Nous possédons un vecteur de 7 valeurs entières que nous désirons trier. En supposant que nous utilisons l'algorithme de **tri par** **sélection simple** :

* Donnez ci-dessous le contenu du tableau après chacune des étapes de l'algorithme (une étape correspond au placement d'un nouvel élément dans la partie des éléments triés du tableau).
* Dans la colonne de droite, inscrire le nombre de comparaisons effectuées pour réaliser cette étape, depuis la fin de la précédente.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Étape | [ 0 ] | [1] | [2] | [3] | [4] | [5] | [6] | Nb. de comparaisons |
| **21** | **25** | **10** | **3** | **30** | **5** | **18** |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

* 1. Combien de permutations (un échange de deux éléments du tableau) avons-nous dû réaliser au total?

|  |  |
| --- | --- |
| Réponse |  |

* 1. Vrai ou faux, cet algorithme est particulièrement efficace dans le cas où la table est déjà triée en bonne partie.

|  |  |
| --- | --- |
| Réponse |  |

1. Codez un programme Java qui trie un vecteur quelconque de 10 éléments en utilisant la technique de tri par sélection. Le programme doit afficher le contenu du vecteur une fois trié. Utilisez le fichier de départ **triSelectionSimple2.jar** (Léa).
2. Nous possédons un vecteur de 7 valeurs entières que nous désirons trier. En supposant que nous utilisons l'algorithme de **tri par énumération**:

* Donnez, ci-dessous, la valeur du compteur qui sera associé à chacune des valeurs de la table, et ce, à chaque étape. Ici une étape équivaut à un parcours de la table
* Dans la colonne de droite, inscrire le nombre de comparaisons effectuées pour réaliser cette étape, depuis la précédente.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Étape | [ 0 ] | [1] | [2] | [3] | [4] | [5] | [6] | Nb. de comparaisons |
| **10** | **25** | **21** | **3** | **30** | **5** | **18** |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

* 1. Donnez un synonyme de tri par énumération:

|  |  |
| --- | --- |
| Réponse |  |

* 1. Vrai ou faux, l'efficacité du tri par énumération dépend de l'état initial de la table:

|  |  |
| --- | --- |
| Réponse |  |

1. Le fichier *trisComparaisons2.jar* (Léa), une fois complété, contiendra une implantation du tri par sélection, suivi d'une implantation du tri par énumération.
   1. Exécutez le programme et examinez le code. Comment détermine-t-on les valeurs à placer dans le tableau à trier initialement?

|  |
| --- |
|  |

* 1. Nous désirons savoir lequel des deux algorithmes est le plus rapide. La méthode currentTimeMillis()[[1]](#footnote-1) de la classe System retourne l'heure courante, exprimée en millisecondes écoulées depuis la date repère du 1er janvier 1970.

long t1 = System.currentTimeMillis();

À l'aide de cette méthode, on vous demande de modifier le code de façon à chronométrer chaque algorithme de tri. Faites plusieurs tests pour chaque cas et remplissez ensuite le tableau suivant :

|  |  |  |
| --- | --- | --- |
| Nombre d'éléments dans le tableau | Temps requis en ms pour le tri | |
| Sélection | Énumération |
| 100 |  |  |
| 5 000 |  |  |
| 20 000 |  |  |
| 30 000 |  |  |
| 50 000 |  |  |

Quel est l'algorithme le plus performant?

|  |
| --- |
|  |

1. Nous possédons un vecteur de 7 valeurs entières que nous désirons trier. En supposant que nous utilisons l'algorithme de **tri par insertion**

* Donnez, ci-dessous, le contenu de la table après chacune des étapes de l'algorithme. Ici une étape équivaut à l'insertion d'un nouvel élément.
* Dans les colonnes de droite, inscrire le nombre de comparaisons et le nombre de déplacements qu'il a fallu effectuer pour arriver à cette étape, depuis la précédente.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Étape | | [ 0 ] | | [1] | [2] | [3] | [4] | [5] | [6] | Nb. de comparaisons | Nb. de déplacements |
| **10** | | **25** | **21** | **3** | **30** | **5** | **18** |
| 1 | |  |  |  |  |  |  |  |  |  |
| 2 | |  |  |  |  |  |  |  |  |  |
| 3 | |  |  |  |  |  |  |  |  |  |
| 4 | |  |  |  |  |  |  |  |  |  |
| 5 | |  |  |  |  |  |  |  |  |  |
| 6 | |  |  |  |  |  |  |  |  |  |

* 1. Vrai ou faux: l'efficacité du tri par insertion dépend de l'état initial de la table:

|  |  |
| --- | --- |
| Réponse |  |

1. Ajoutez au fichier *trisComparaison.java* l'implantation de la méthode de **tri par insertion** selon l'algorithme vu en classe.  
   N'oubliez pas de remettre les valeurs initiales dans le vecteur à trier.   
   Pour copier un vecteur dans un autre, utilisez la méthode *arraycopy[[2]](#footnote-2)* de la classe *System*.

Chronométrez les temps d'exécution de chaque algorithme et reportez les dans le tableau ci-dessous.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre d'éléments dans le tableau | Temps requis en ms pour le tri | | |
| Sélection | Énumération | Insertion |
| 100 |  |  |  |
| 5 000 |  |  |  |
| 20 000 |  |  |  |
| 30 000 |  |  |  |
| 50 000 |  |  |  |

Comparez l'efficacité du tri par insertion aux deux autres algorithmes de tri ?

|  |
| --- |
|  |

1. Nous possédons un vecteur de 7 valeurs entières que nous désirons trier. En supposant que nous utilisons l'algorithme de **tri à bulle**

* Donner ci-dessous le contenu du tableau après chacune des étapes de l'algorithme. Ici une étape équivaut à une comparaison entre deux valeurs. Ainsi, s'il n'y a pas d'échange, une étape peut être identique à la précédente.
* On vous demande d'encercler les valeurs qui sont définitivement placées au bon endroit par l'algorithme.
* Dans la colonne de droite, cochez la case lorsqu'on a dû faire un échange lors de cette étape.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Étape | Parcours | [ 0 ] | [1] | [2] | [3] | [4] | [5] | [6] | Échange? |
| **2** | **30** | **8** | **25** | **3** | **5** | **28** |
| 1 |  |  |  |  |  |  |  |  |  | |
| 2 |  |  |  |  |  |  |  |  |  | |
| 3 |  |  |  |  |  |  |  |  |  | |
| 4 |  |  |  |  |  |  |  |  |  | |
| 5 |  |  |  |  |  |  |  |  |  | |
| 6 |  |  |  |  |  |  |  |  |  | |
| 7 |  |  |  |  |  |  |  |  |  | |
| 8 |  |  |  |  |  |  |  |  |  | |
| 9 |  |  |  |  |  |  |  |  |  | |
| 10 |  |  |  |  |  |  |  |  |  | |
| 11 |  |  |  |  |  |  |  |  |  | |
| 12 |  |  |  |  |  |  |  |  |  | |
| 13 |  |  |  |  |  |  |  |  |  | |
| 14 |  |  |  |  |  |  |  |  |  | |
| 15 |  |  |  |  |  |  |  |  |  | |
| 16 |  |  |  |  |  |  |  |  |  | |
| 17 |  |  |  |  |  |  |  |  |  | |
| 18 |  |  |  |  |  |  |  |  |  | |
| 19 |  |  |  |  |  |  |  |  |  | |
| 20 |  |  |  |  |  |  |  |  |  | |

* 1. Donnez un synonyme de *tri à bulle* :

|  |  |
| --- | --- |
| Réponse |  |

* 1. Vrai ou faux, cet algorithme est particulièrement efficace dans le cas où la table est déjà triée en bonne partie :

|  |  |
| --- | --- |
| Réponse |  |

1. Ajoutez au fichier *trisComparaison.java* l'implantation de la méthode de **tri à bulle** selon l'algorithme vu en classe. N'oubliez pas de remettre les valeurs initiales dans le vecteur à trier.

Chronométrez les temps d'exécution de chaque algorithme et reportez les dans le tableau ci-dessous.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre d'éléments dans le tableau | Temps requis en ms pour le tri | | | |
| Sélection | Énumération | Insertion | Bulle |
| 100 |  |  |  |  |
| 5 000 |  |  |  |  |
| 20 000 |  |  |  |  |
| 30 000 |  |  |  |  |
| 50 000 |  |  |  |  |

* 1. Quelles sont vos conclusions générales quant aux diverses méthodes de tri, lorsqu'appliquées ainsi sur un tableau de valeurs aléatoires?

|  |
| --- |
|  |

* 1. Changez vos tests pour utiliser dans tous les cas un vecteur déjà trié (ou presque) de 25000 éléments.

Est-ce que la performance relative de chaque algorithme est la même?

|  |
| --- |
|  |

1. Pour plus de détails consultez le site *http://docs.oracle.com/javase/8/docs/api/index.html* [↑](#footnote-ref-1)
2. Pour plus de détails consultez le site *http://docs.oracle.com/javase/8/docs/api/index.html* [↑](#footnote-ref-2)