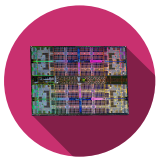


Benchmark

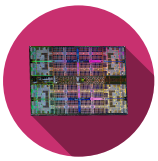
Labo Architecture des ordinateurs

Contenu

1 Objectifs 1
2 Exécuter les benchmarks 2
2.1 Installation de Geekbench 2
2.2 Informations système 2
2.3 Résumé de l'appareil 2
2.4 Test 1 - Geekbench - CPU 3
2.5 Test 2 - Geekbench - GPU 4
2.6 RAM 5
2.7 Test 3 - Compression Zip 6
3 Comparaison des résultats 7
3.1 Comparaison CPU 7
3.2 Comparaison GPU 7
3.3 Comparaison RAM 7
3.4 Comparaison des ordinateurs 7
3.5 Calculs de performance du program Zip 7
4 Optimisation software 9
4.1 Language interprété VS language compilé 9
4.2 Optimisation 12
Glossaire 15

1 Objectifs

L'objectif de ce premier laboratoire est de comparer votre propre ordinateur à d'autres dans le monde ainsi qu'à ceux de vos camarades. L'impact du software sur les performances est ensuite abordé.



2 Exécuter les benchmarks

2.1 Installation de Geekbench

Pour les tests de performance, nous utiliserons le programme gratuit Geekbench 6 de Primatelabs. Il peut être téléchargé sur le site https://www.geekbench.com/download/. [1]

Table 1 - Primatelabs ainsi que le logo Geekbench

2.2 Informations système

Les informations propres au matériel du système sont trouvables grâce à :

2.2.1 Windows

Sur Windows, lancez **CPU-Z** disponible sous **car-labs/bem/cpuzx64.exe**. Il est aussi possible d'utiliser **HWiNFO** sous **car-labs/bem/hwinfo64.exe**.

2.2.2 MAC

Sur MAC, cliquez en haut à gauche sur la pomme → maintenir la touche **Options** → **Informations système**.

2.2.3 Linux

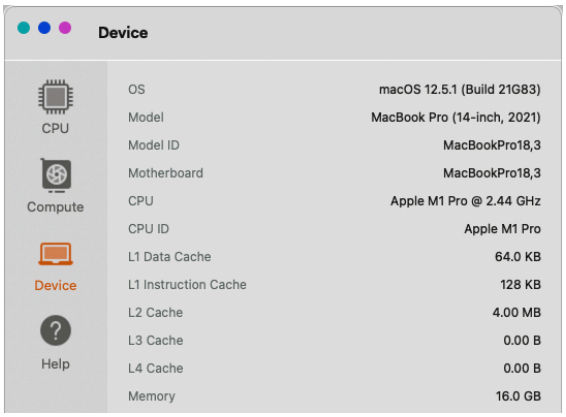
Vous pouvez utiliser le moniteur de ressources intégré à votre OS ou le logiciel HardInfo2.

2.3 Résumé de l'appareil

Tout d'abord, nous allons découvrir les spécifications de l'appareil. Celles-ci sont affichées sur les différentes pages de Geekbench ainsi qu'après les tests. Notez-les et partagez-les avec vos camarades.

Vous pouvez aussi utiliser CPU-Z (Windows) / Informations Système (MAC) ici.

Pour partager les résultats, concertez-vous entre vous et créez un document Excel commun sur le canal Teams.



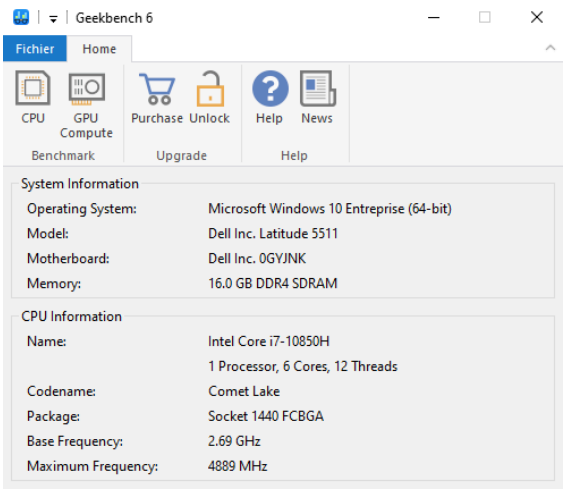


Table 2 - Informations sur la machine (Mac / Windows)

Person	Operating System (OS)	Central Processing Unit (CPU)	CPU -cores	CPU Arch.	Frequency [GHz]
Silvan Zahno	macOS 12.5.1	Apple M1 Pro	10	AARCH64	2.44
Silvan Zahno	macOS 15.5	Apple M4 Max	16(12p/4e)	AARCH64	4.49
Axel Amand	Win 11 10.0.22	I5-12600K	10	AMD64	3.7

L1 Data [kB]	L1 Instruction [kB]	L2 [MB]	L3 [MB]	L4 [MB]	RAM [GB]
64	128	4	n.a.	n.a.	16
64	128	4	n.a.	n.a.	64
48*8	32*8	1.25*2	20	0	32

Table 3 - Résultats Informations de l'appareil

Faites attention aux unités utilisées lorsque vous notez les résultats!

2.4 Test 1 - Geekbench - CPU

Exécutez le test CPU. Relevez les informations importantes de votre CPU ainsi que les scores geekbench.

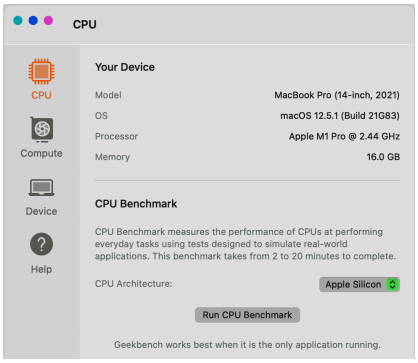


Figure 1 - Test de performance CPU

Person	**Single-Core Score**	**Multi-Core Score**
Silvan Zahno (M1)	2279	11839
Silvan Zahno (M4)	3813	25272
Axel Amand	2372	12566

Table 4 - Résultats du test de performance CPU

2.5 Test 2 - Geekbench - GPU

Exécutez le test Graphical Processing Unit (GPU). Relevez les informations importantes de votre GPU ainsi que les scores geekbench.

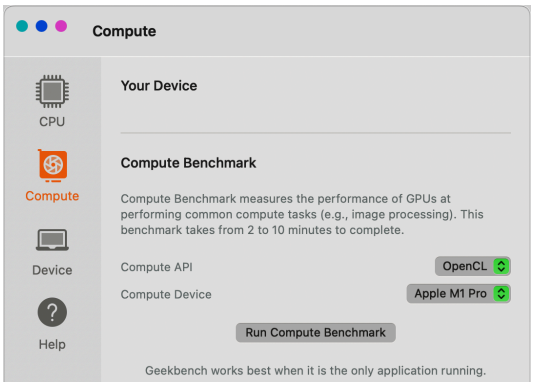
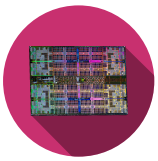


Figure 2 - Test de performance GPU

Person	**OpenCL**	**Metal**	**CUDA**	**Vulkan**
Silvan Zahno (M1)	40823	67476	n.a.	n.a.
Silvan Zahno (M4)	111779	166950	n.a.	n.a.
Axel Amand (UHD Graphics 770)	7516	n.a.	n.a.	8502
Axel Amand (GTX1080)	48083	n.a.	51896	65467

Table 5 - Résultats du test de performance CPU

Découvrez ce qu'est exactement le score Geekbench et documentez-le dans votre rapport.



2.6 RAM

La RAM, pour **R**andom **A**ccess **M**emory, est aussi un élément essentiel pour que le système puisse travailler efficacement.

2.6.1 Windows

Les informations de RAM sont disponibles sous l'onglet **Memory** de CPU-Z :

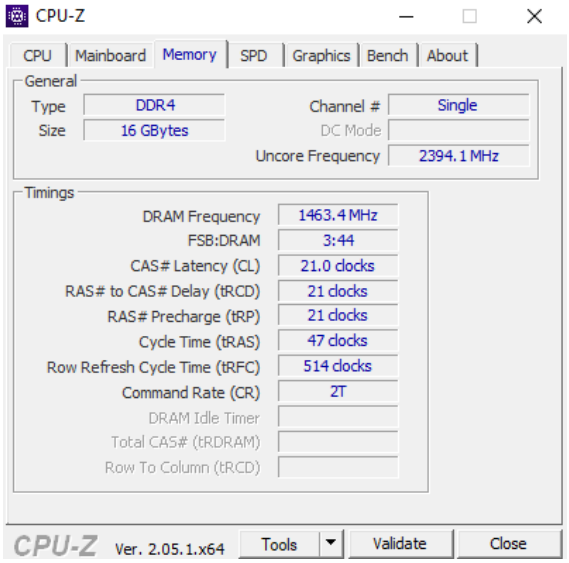


Figure 3 - Informations RAM - CPU-Z

2.6.2 MAC

Les informations de RAM sont disponible sous **Memory** de l'onglet **Informations Système**:

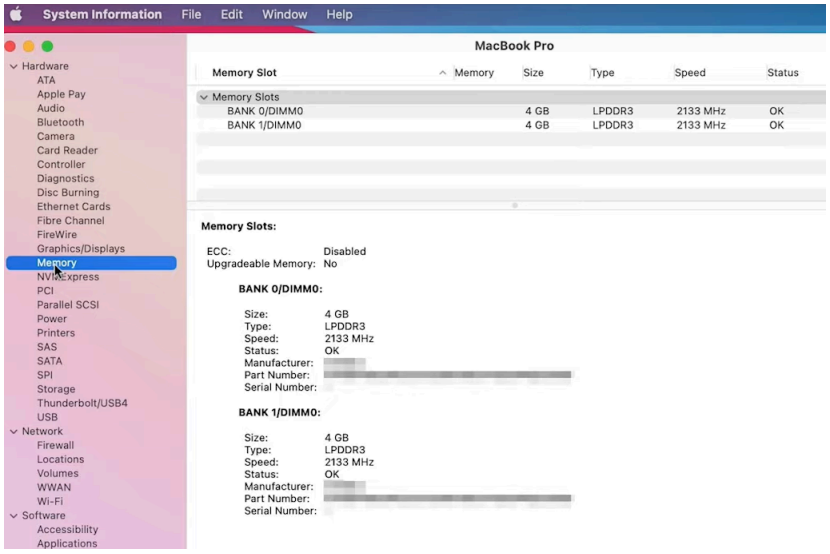
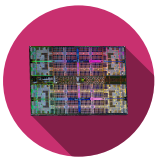


Figure 4 - Informations RAM - MAC

Relevez les informations importantes permettant de caractériser votre mémoire RAM.



2.7 Test 3 - Compression Zip

Pour le dernier test, nous allons décompresser et recompresser le fichier **`42.zip`**. Vous trouverez les fichiers nécessaires sous **car-labs/bem/zip**.

Pour ce test, vous avez besoin d'au moins 1 Go d'espace libre !

Ouvrez un terminal et exécutez les commandes suivantes :

```
# goto the corresponding folder (command to be adapted)
cd car-labs/bem/zip

# Linux and MacOS
chmod +x zip-benchmarking.bash
./zip-benchmarking.bash

# Windows
.\zip-benchmarking.bat
```

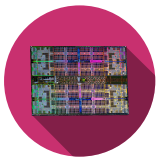
Notez vos résultats.

Comment est-il possible qu'un fichier de 810 Mo soit réduit à 2,7 Mo avec la compression zip ? Faites une hypothèse dans votre rapport.

En raison de la taille du fichier 42.txt, ne l'ouvrez qu'avec précaution dans un éditeur de texte. Votre ordinateur pourrait se bloquer.

Person	**Unzip**	**Zip**
Silvan Zahno (M1)	2.651	3.619
Silvan Zahno (M4)	1.717	2.383
Axel Amand	1.66	2.97

Table 6 - Résultats du test de performance Zip



3 Comparaison des résultats

3.1 Comparaison CPU

Comparez vos résultats Geekbench avec la liste officielle de Geekbench. [2]. Décrivez vos résultats :
- A quoi correspond le score ?
- Quels sont les points testés par Geekbench 6 ? Donner les 5 grandes catégories.

Au niveau du CPU lui-même :
- A quoi correspondent les architectures x86, AMD64 (x86_64) et AARCH64 ?
- Une fréquence supérieure d'horloge est-elle gage de performances supérieures d'un CPU à l'autre ?

3.2 Comparaison GPU

Comparez vos résultats Geekbench avec la liste officielle de Geekbench. [2]. Pour cela, vous devez prendre la comparaison de votre carte graphique, Cuda [3], OpenCL [4] ou Metal [5]. Décrivez vos résultats :
- Qu'est-ce que CUDA, OpenCL et Metal ?
- Quels sont les points testés par Geekbench 6 ? Donner les 4 grandes catégories.

Au niveau du GPU lui-même :
- Comment un GPU diffère-t'il d'un CPU ?

3.3 Comparaison RAM

Décrivez les caractéristiques et taux de transferts de votre RAM.

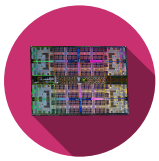
D'un point de vue plus général :
- Quelle(s) différence(s) existe(nt) entre des RAMs de type DDR4, LPDDR4 et DDR5 ?
- A quoi correspond le CAS, aussi nommé CL pour CAS Latency ?
- Je travaille sur un programme accédant à des milliers de données en cache. D'un point de vue purement performance d'accès à une donnée, devrais-je préférer utiliser de la RAM DDR4 4000 MT/s CL18 ou de la RAM DDR5 4000 MHz CL38 ?

3.4 Comparaison des ordinateurs

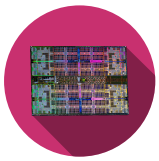
A l'aide du fichier Excel, créez des déclarations quantitatives sur les performances de votre ordinateur par rapport à celles de vos camarades, à l'aide de graphiques et d'une analyse de données.

3.5 Calculs de performance du program Zip

Dans le programme Zip, vous avez reçu un temps pour la compression et la décompression de votre système et de celui de vos camarades. Rassemblez et **normalisez** les différents temps :
- Quel système est le plus rapide ?
- Quel est le facteur entre le système le plus lent et le plus rapide ?
- Y'a-t'il une corrélation avec le score CPU ? La RAM ?



Voir à ce sujet les diapositives et les exercices du cours sur les calculs de performance dans le chapitre Per. En particulier les formules de « *Normalized Execution Time* ».



4 Optimisation software

En dehors des performances pures du système, le software joue un grand rôle sur lequel le développeur à un impact direct, notamment sur deux points:

1. L'optimisation de la mémoire et des algorithmes utilisés.
   - Utiliser plus d'instructions CPU que nécessaire réduit d'autant la performance de son programme.
   - La complexité temporelle de l'algorithme lui-même, noté O (grand O).
2. Le language utilisé ainsi que son type d'évaluation :
   - Les languages comme l'Assembleur, C, C++, Rust, Go … sont dits **compilés** - ils sont traduits directement en code machine compréhensible par le CPU. Ils sont plus rapides, mais doivent être compilés pour chaque architecture CPU et plateforme visées.
   - Les languages comme Javascript, PHP, Ruby, Python … sont dits **interprétés** - le code est transformé en instructions CPU au fur et à mesure. Ils sont plus lents, mais permettent une plus grande flexibilité et portabilité sur différents systèmes.
   - Les languages comme Java et C# sont un mélange, faisant usage de la notion de « compilation à la volée » (just-in-time compilation) :
     - Java compile le code en bytecode, language compréhensible par une machine virtuelle Java (JVM). La JVM s'occupe ensuite de traduire le bytecode en instructions machine.
     - C# compile le code en language intermédiaire IL. Sur la machine cible, le code IL est traduit en instructions CPU.

4.1 Language interprété VS language compilé

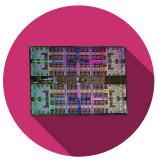
Trois implémentations de l'algorithme BubbleSort sont disponibles sous **car-labs/bem/sorting/algorithms**. Des données de test sont disponibles sous **car-labs/bem/sorting/data**, le nom du fichier se référant au nombre de valeurs présentent.

Le principe de l'algorithme et de réaliser n passes en croisant les données deux à deux pour ramener les valeurs les plus hautes au fond du tableau de données, le pseudo-code donnant :

```
procedure bubbleSort(A : list of sortable items)
  n := length(A)
  repeat
    swapped := false
    for i := 1 to n-1 inclusive do
      { if this pair is out of order }
      if A[i-1] > A[i] then
        { swap them and remember something changed }
        swap(A[i-1], A[i])
        swapped := true
      end if
    end for
  until not swapped
end procedure
```

Sa complexité est fixe et de $O(n^2)$ (il faut passer jusqu'à n^2 éléments pour trier la liste au complet).

Avec une petite optimisation, il est possible de faire varier sa complexité entre $O(n^2)$ pour le pire des cas, jusqu'à $O(n)$ pour une liste déjà triée.



4.1.1 Javascript

Sous **javascript**, lancer **index.html**.

Vous pouvez utiliser la partie **Bubble Sort Viewer** pour générer des données et animer le tri.

Sous **Performance Test**, il est possible de charger un fichier de données en appuyant sur **Parcourir ...** puis de lancer le tri en Javascript. Le navigateur bloque jusqu'à ce que les données soient triées, puis le temps est donné sous forme de texte.

Evitez de charger plus de 10'000 valeurs !

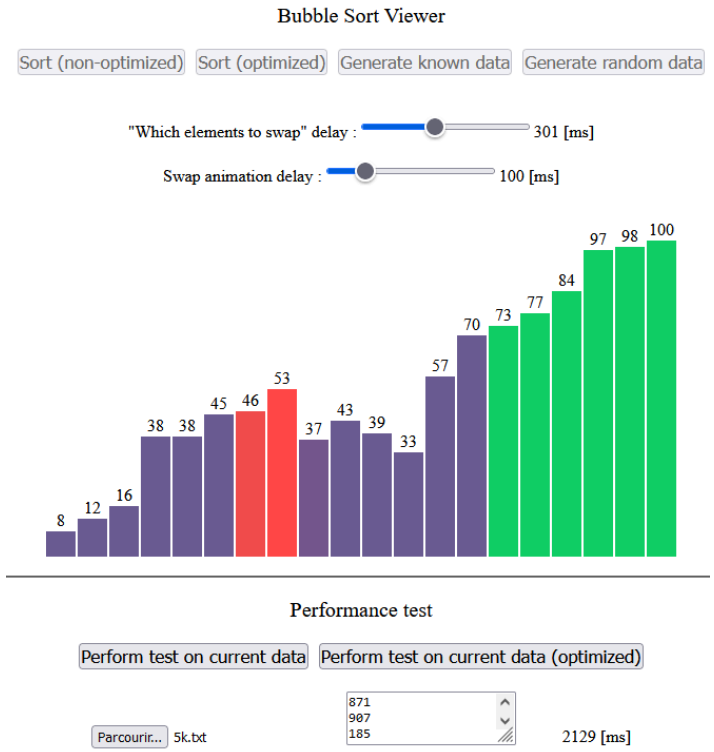


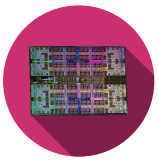
Figure 5 - Bubblesort Javascript

4.1.2 Rust

Sous **rust/release**, lancer:

- Windows: **rust_sorting_xxx_Windows_x64.exe**
- MAC: lancez une console et `chmod +x rust_sorting_xxx_MacOS_x64 && ./rust_sorting_xxx_Mac_AARCH64` *(MAC ARM64 uniquement)*
- Linux: lancez une console et `chmod +x rust_sorting_xxx_Linux_x64 && ./rust_sorting_xxx_Linux_x64`

Une fenêtre propose de charger le fichier de données puis donne les résultats pour l'algorithme non-optimisé et optimisé. Seuls les résultats de l'algorithme BubbleSort sont importants ici.



```
Select data file on the newly opened GUI
Reading data into memory
Sorting
 * Bubble sort (non-opti) done in 189 [ms]
Sorting
 * Bubble sort (optimized) done in 156 [ms]
```

Figure 6 - Bubblesort Rust

4.1.3 C

Sous **C/release**, lancer:

- **`Windows`**: `run.cmd`
- **`MAC`**: lancez une console et `chmod +x run.sh && ./run.sh`
- ***`Linux`***: *le programme n'est pas compilé pour Linux*

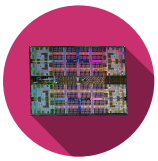
Une fenêtre demande le fichier à charger, puis donne les résultats pour l'algorithme non-optimisé et optimisé.

```
Bubble Sort - C version
    Data Selection

1. 20
2. 1k
3. 5k
4. 10k
5. 50k
6. 100k
0. Exit

Enter your choice: 6
Checking args
Reading data into memory
 * Found 100000 data
Copying data for opti. algo
Sorting
 * Bubble sort (non-opti) done in 24883 [ms]
 * Bubble sort (optimized) done in 23289 [ms]
```

Figure 7 - Bubblesort C

4.1.4 Test

1. Testez et classez les résultats pour l'algorithme non-optimisé :
   - Javascript : 1k, 5k et 10k, 50k, 10ksorted éléments
   - C / Rust : 1k, 5k, 10k, 50k, 100k, 10ksorted, 50ksorted éléments
2. Testez et commentez les résultats pour l'algorithme optimisé :
   - C / Rust : 10ksorted, 50ksorted
3. Donnez le facteur de différence entre les temps pour les différents languages.
4. Dans quelles conditions le code optimisé est-il plus rapide que le code non optimisé et quelle pourrait en être la raison ?



Regardez avec une liste déjà triée, dans le code Javascript « Bubble Sort Viewer » lors de l'exécution du code non optimisé par rapport au code optimisé.

1. « Generate Random Data »
2. « Sort (optimized) » avec délais 0ms
3. Définir le délai de permutation et l'animation de permutation à 100ms
4. Exécuter « Sort (non-optimized) »
5. Exécuter « Sort (optimized) »
6. Comparer les deux dernières exécutions

4.2 Optimisation

Proposez un algorithme de tri plus efficace et comparez-le au bubble sort. Vous pouvez également utiliser des librairies ou fonctions existantes.

Il est possible d'écrire cet algorithme dans le language de votre choix. Sinon, un template Scala est disponible sous **car-labs/bem/sorting/algorithms/scala/main.sc**.

Pour l'utiliser rapidement et simplement, rendez-vous sous https://www.jdoodle.com/compile-scala-online/ et copiez-collez son contenu dans l'éditeur.

En bas de page, cliquez sur le bouton upload et sélectionnez le fichier de données à téléverser. Une fois fait, le fichier est disponible en lecture au chemin « **/uploads/myfilename.txt** ». Modifiez la ligne `val dataFile = "/uploads/10k.txt";` afin de réfléter le fichier à tester.

Le code peut être exécuté en appuyant sur le bouton **Execute**. La fenêtre de sortie donne le temps d'exécution, ainsi que si le tableau a été correctement trié ou non.

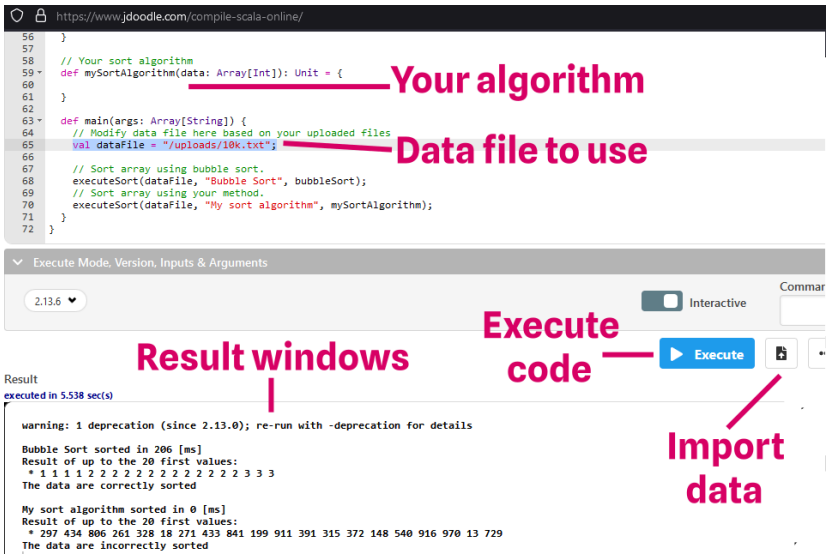
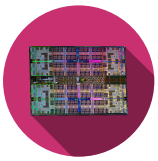


Figure 8 - Scala sur JDoodle

4.2.1 Todo

1. Codez l'algorithme de tri de votre choix.
2. Testez le pour les cas standards ainsi que les meilleurs des cas (pour ça, utiliser les fichiers nommés **xxxsorted**).
3. Comparez-le avec vos résultats BubbleSort précédents.

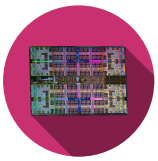


Si vous utilisez un service comme JDoodle qui fait tourner votre code sur un ordinateur différent du votre, il est nécessaire de réitérer les tests Bubble Sort afin d'écarter le facteur de différence de performance entre les deux machines.

1. Quels sont les avantages du language choisi ? Et ses inconvénients ?
2. Donnez le nom de l'algorithme utilisé, sa manière de fonctionner, et :
   - Scala : incluez votre fonction de tri dans le rapport. Le reste du template donné n'est pas nécessaire.
   - Autre : annexez le code complet.

4.2.2 Template scala

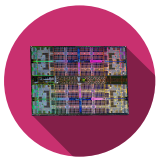
```
object JDoodle {

  // Reads file and create memory array
  def readFile(filename: String): Array[Int] = {
    val bufferedSource = io.Source.fromFile(filename)
    val lines = (for (line <- bufferedSource.getLines()) yield line.toInt).toArray
    bufferedSource.close
    lines
  }

  // Execute the given sort function with timings check
  def executeSort(filename: String, sortName: String, func: (Array[Int]) => Unit): Unit
= {
    var data = readFile(filename);
    var start = System.currentTimeMillis();
    func(data);
    var end = System.currentTimeMillis();

    printf("\n%s sorted in %d [ms]\nResult of up to the 20 first values:\n * ",
sortName, end-start);
    var i = 0;
    while(i < 20 && i < data.length){
      printf("%d ", data(i));
      i += 1;
    }
    printf("\nThe data are %s sorted\n", if(checkValues(data)) "correctly" else
"incorrectly");
  }

  // Test if the array is correctly sorted
  def checkValues(data: Array[Int]): Boolean = {
    var last = data(0);
    var dta = 0;
    for(dta <- data){
        if (dta < last){return false;}
        last = dta;
    }
    return true;
  }

  // A bubble sort example
  def bubbleSort(data: Array[Int]): Unit = {
```



```
    var i: Int = 0
    var j: Int = 0
    var t: Int = 0

    while (i < data.length) {
      j = data.length - 1;
      while (j > i) {
        if (data(j) < data(j - 1)) {
          t = data(j);
          data(j) = data(j - 1);
          data(j - 1) = t;
        }
        j = j - 1
      }
      i = i + 1
    }
  }

  // Your sort algorithm
  def mySortAlgorithm(data: Array[Int]): Unit = {

  }

  def main(args: Array[String]) {
    // Modify data file here based on your uploaded files
    val dataFile = "/uploads/10k.txt";

    // Sort array using bubble sort.
    executeSort(dataFile, "Bubble Sort", bubbleSort);
    // Sort array using your method.
    executeSort(dataFile, "My sort algorithm", mySortAlgorithm);
  }
}
```

4.2.3 *Tâche optionnelle*

Pour les plus ambitieux d'entre vous, vous pouvez essayer de battre le programme Rust **`rust-glidesort`**.
Uniquement dans ce cas, vous devriez utiliser le fichier **`10000k.txt`**.



Glossaire

CPU **– Central Processing Unit** 3, 4, 7, 9

GPU **– Graphical Processing Unit** 4, 7

OS **– Operating System** 3