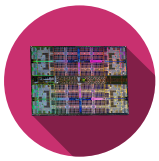


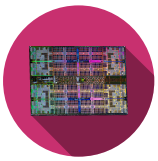
Benchmark

Labor Computerarchitektur

Inhalt

1 Ziel

Ziel dieses ersten Labors ist es, Ihren eigenen Computer mit anderen aus der Welt, sowie denen Ihrer Kommilitonen zu vergleichen. Anschliessend wird der Einfluss der Software auf die Leistung besprochen.



2 Ausführen der Benchmarks

2.1 Installation Geekbench

Für die Leistungstests werden wir das Gratis Program Geekbench 6 von Primatelabs benutzten. Dies kann auf der Webseite https://www.geekbench.com/download/ [1] heruntergeladen und installiert werden.

2.2 Systeminformationen

Informationen, die für die Hardware des Systems spezifisch sind, finden Sie über :

2.2.1 Windows

Unter Windows starten Sie **CPU-Z**, das unter **car-labs/bem/cpuzx64.exe** verfügbar ist.

2.2.2 MAC

Unter Mac klicken Sie oben links auf den Apfel → halten Sie die Taste **Optionen** → **Systeminformationen**.

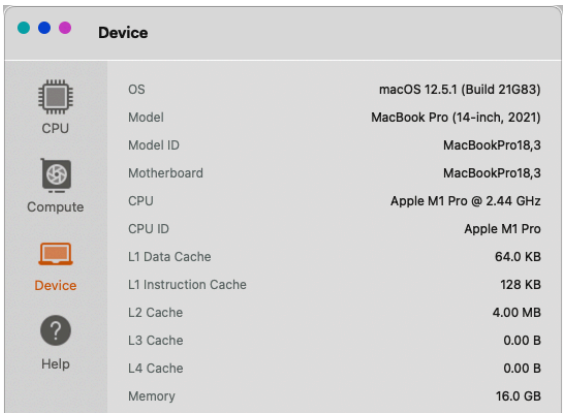
Tabelle 1 - Primatelabs sowie Geekbench Logo

2.3 Zusammenfassung des Geräts

Als erstes werden wir die Spezifikationen der Gerätes herausfinden. Diese werden bei Geekbench auf den einzelnen Seiten sowie nach dem durchführen der einzelenen Tests angezeigt. Schreiben Sie diese auf und teilen Sie diese mit Ihren Komilitonen.

Sie können hier auch CPU-Z (Windows) / Systeminformationen (MAC) verwenden.

Um die Resultate auszutauschen, sprechen Sie sich untereinander ab und erstellen Sie ein gemeinsames Excel Dokument auf dem Teams Kanal.



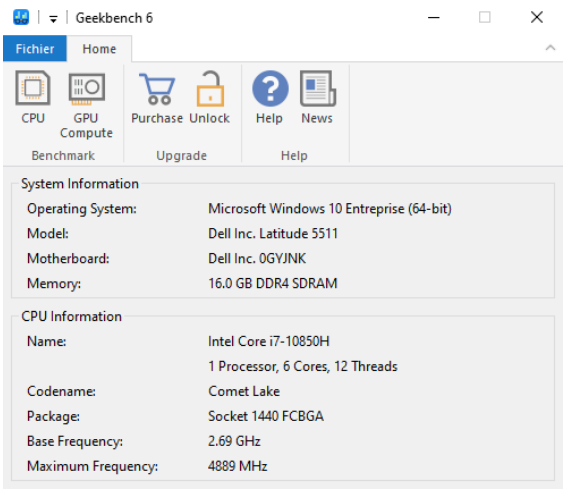


Tabelle 2 - Geräteinformationen (Mac / Windows)

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
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	0.128	4	0	0	16
48*8	32*8	1.25*2	20	0	32

Tabelle 3 - Resultate Geräteinformationen

Achten Sie beim Aufschreiben der Resultate auf die benutzten Einheiten!

2.4 Test 1 - Geekbench - CPU

Führen Sie den CPU Leitungstest durch. Notieren Sie wichtige Informationen eures CPU's sowie der Geekbench-Werte.

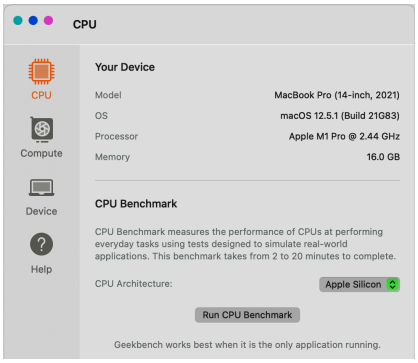


Abbildung 1 - CPU Leistungstest

Person	**Single-Core Score**	**Multi-Core Score**
Silvan Zahno	2279	11839
Axel Amand	2372	12566

Tabelle 4 - Resulate CPU Leistungstest

2.5 Test 2 - Geekbench - GPU

Führen Sie den Graphical Processing Unit (GPU) Leitungstest durch. Notieren Sie wichtige Informationen auf eures GPU's und dessen Geekbench-Werte.

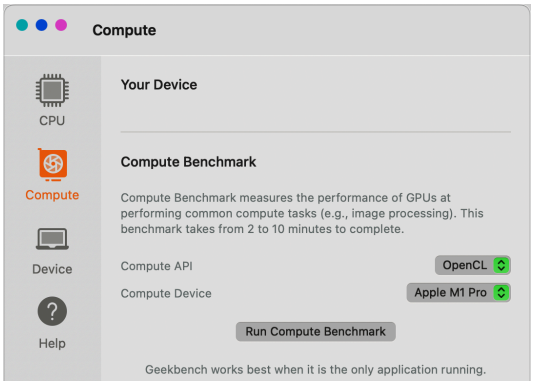
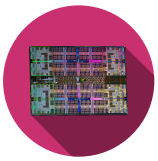


Abbildung 2 - GPU Leistungstest

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

Tabelle 5 - Resulate CPU Leistungstest

Finden Sie heraus, was genau der Geekbench-Score ist, und dokumentieren Sie ihn in Ihrem Bericht.



2.6 RAM

RAM, für **R**andom **A**ccess **M**emory, ist ebenfalls ein wesentlicher Bestandteil, damit das System effizient arbeiten kann.

2.6.1 Windows

RAM-Informationen finden Sie unter **Memory** von CPU-Z:

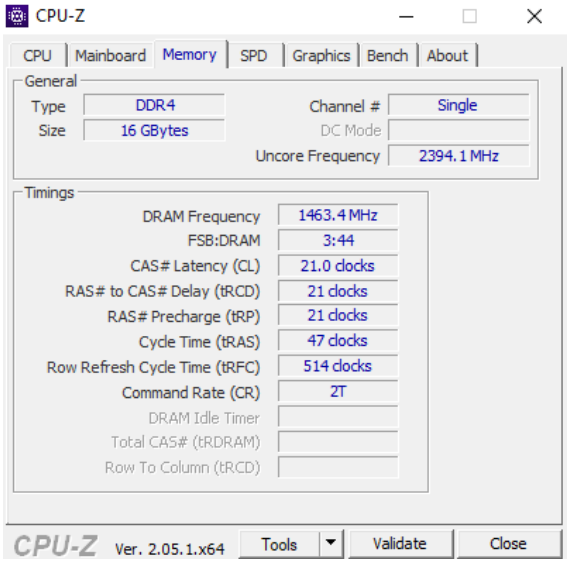


Abbildung 3 - RAM-Datei - CPU-Z

2.6.2 MAC

Informationen über den Arbeitsspeicher finden Sie unter **Memory** von **Systeminformationen** :

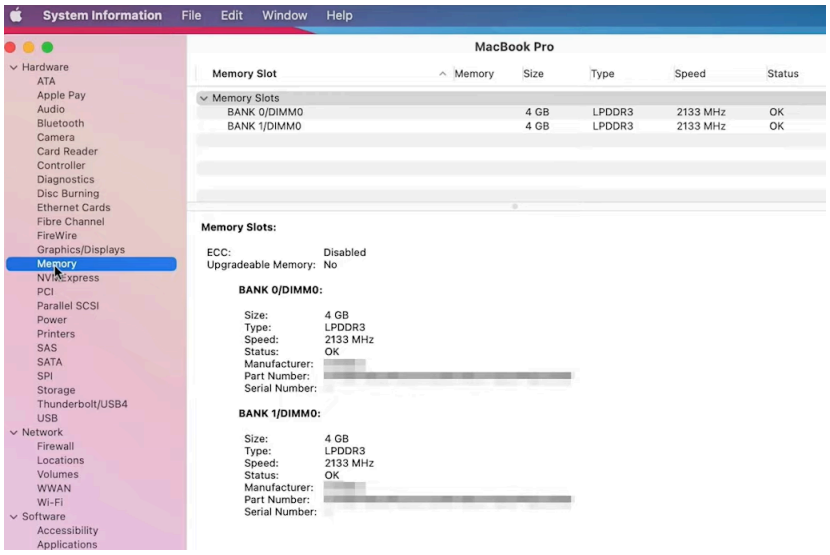
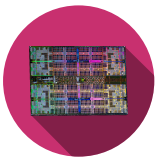


Abbildung 4 - RAM-Datei - MAC

Notieren Sie wichtige Informationen, um ihren RAM-Speicher zu charakterisieren.



2.7 Test 3 - Zip Komprimierung

Für den letzten Test werden wir eine die Datei **`42.zip`** dekomprimieren und wieder komprimieren. Die benötigten Dateien finden sie in ihrem Repository **car-labs/bem/zip**.

Für diesen Test benötigen Sie mindestens 1GB freier Speicherplatz!

Öffnen Sie ein Terminal und führen Sie die folgende Befehle aus:

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

# Linux and MacOS
./zip-benchmarking.bash

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

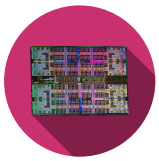
Notieren Sie Ihre Resultate.

Wie ist es möglich das eine 810MB Datei mithilfe von Zip-komprimierung auf 2.7MB verkleinert werden kann? Machen Sie eine Annahme in Ihrem Bericht.

Dadruch dass die Datei 42.txt sehr gross ist öffnen Sie diese nur mit Bedacht in einem Texteditor. Ihr Computer könnte sich aufhängen.

Person	**Unzip**	**Zip**
Silvan Zahno	2.651	3.619
Axel Amand	1.66	2.97

Tabelle 6 - Resulate des Zip Leistungstest



3 Vergleich der Resultate

3.1 CPU Vergleich

Vergleichen Sie Ihre Geekbench Resultate mit der offiziellen Geekbench liste [2]. Beschreiben Sie Ihre Resultate:

- Worauf bezieht sich die Punktzahl?
- Welche Punkte wurden von Geekbench 6 getestet? Nennen Sie die fünf Hauptkategorien.

Auf der Ebene des CPU s selbst:

- Was bedeuten die Architekturen x86, AMD64 (x86_64) und AARCH64?
- Bedeutet eine höhere Taktfrequenz eine bessere Leistung von einem CPU zum anderen?

3.2 GPU Vergleich

Vergleichen Sie Ihre Geekbench Resultate mit der offiziellen Geekbench liste [2]. Hierzu müssen Sie den Vergleich von Ihrer Grafikkarte nehmen, Cuda [3], OpenCL [4] oder Metal [5]. Beschreiben Sie Ihre Resultate:

- Was sind CUDA, OpenCL und Metal ?
- Welche Punkte wurden von Geekbench 6 getestet? Nennen Sie die vier Hauptkategorien.

Auf der Ebene des GPU s selbst:

- Wie unterscheidet sich ein GPU von einem CPU?

3.3 RAM Vergleich

Beschreiben Sie die Merkmale und Transferraten Ihres RAM.

Allgemeiner formuliert:

- Welche(n) Unterschied(e) gibt es zwischen RAMs der Typen DDR4, LPDDR4 und DDR5?
- Was ist CAS, auch CL für CAS Latency genannt?
- Ich arbeite an einem Programm, das auf Tausende von Daten im Cache zugreift. Aus rein leistungsorientierter Sicht beim Zugriff auf Daten: Welchen RAM sollte ich bevorzugen, DDR4 4800 MT/s CL18 oder DDR5 5600 MHz CL40?

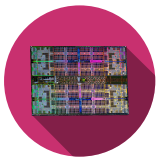
3.4 Computervergleich

Erstellen Sie mithilfe der Excel-Datei quantitative Aussagen über die Leistung Ihres Computers gegenüber der Ihrer Komilitonen mithilfe von Diagrammen und einer Datenanalysen.

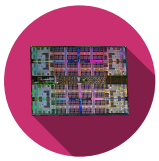
3.5 Leistungsberechnung Zip Program

Beim Zip Program haben Sie für die Komprimierung sowie dekomprimierung eine Zeit für Ihr sowie den Systemen Ihrer Komilitonen erhalten. Normalisieren Sie die Zeiten der Komilitonen auf Ihr System:

- Welches System ist am schnellsten?
- Wie gross ist der Faktor zwischen dem langsamsten und dem schnellsten System?
- Gibt es einen Zusammenhang mit der CPU -Punktzahl? RAM?



Sehen Sie sich hierzu die Slides und Übungen im Kurs über Leistungsberechnungen im Kapitel Per an. Im speziellen die Formeln zur „*Normalized Execution Time*“.



4 Software-Optimierung

Neben der reinen Systemleistung spielt auch die Software eine grosse Rolle, auf die der Entwickler einen direkten Einfluss hat, insbesondere in zwei Punkten:

1. Die Optimierung des Arbeitsspeichers und der verwendeten Algorithmen.
   - Wenn Sie mehr CPU Anweisungen verwenden als nötig, verringert sich die Leistung ihres Programms entsprechend.
   - Die zeitliche Komplexität des Algorithmus selbst, die mit O (grosses O) bezeichnet wird.
2. Die verwendete Sprache und ihre Art der Bewertung:
   - Sprachen wie Assembler, C, C++, Rust, Go ... werden als **compiled** bezeichnet - sie werden direkt in Maschinencode übersetzt, der von der CPU verstanden werden kann. Sie sind schneller, müssen aber für jede angestrebte CPU -Architektur und Plattform kompiliert werden.
   - Sprachen wie Javascript, PHP, Ruby, Python usw. sind sogenannte **interpretierte** Sprachen - der Code wird nach und nach in CPU -Anweisungen umgewandelt. Sie sind langsamer, ermöglichen aber eine grössere Flexibilität und Übertragbarkeit auf verschiedene Systeme.
   - Sprachen wie Java und C# sind eine Mischung, die von dem Konzept der „Just-in-time-Kompilierung“ Gebrauch machen:
     - Java kompiliert den Code in Bytecode, eine Sprache, die von einer Java Virtual Machine (JVM) verstanden werden kann. Die JVM kümmert sich dann um die Übersetzung des Bytecodes in Maschinenbefehle.
     - C# kompiliert den Code in die Zwischensprache IL. Auf dem Zielrechner wird der IL-Code in Anweisungen CPU übersetzt.

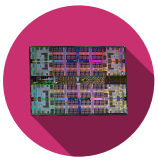
4.1 Interpretierte Sprache VS kompilierte Sprache

Drei Implementierungen des Algorithmus BubbleSort sind unter **car-labs/bem/sorting/algorithms** verfügbar. Testdaten sind unter **car-labs/bem/sorting/data** verfügbar, wobei sich der Dateiname auf die Anzahl der dargestellten Werte bezieht.

Das Prinzip des Algorithmus besteht darin, n Durchläufe durchzuführen, indem die Daten paarweise gekreuzt werden, um die höchsten Werte auf den Boden der Datentabelle zu bringen:

```
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
```

Ihre Komplexität ist fest und beträgt $O(n^2)$ (es müssen bis zu n^2 Elemente durchlaufen werden, um die gesamte Liste zu sortieren).



Mit einer kleinen Optimierung ist es möglich, die Komplexität zwischen $O(n^2)$ für den schlimmsten Fall und $O(n)$ für eine bereits sortierte Liste zu variieren.

4.1.1 Javascript

Unten **javascript**, starten Sie **index.html**.

Sie können den Teil **Bubble Sort Viewer** verwenden, um Daten zu generieren und die Sortierung zu animieren.

Unter **Performance Test** ist es möglich, eine Datei mit Daten zu laden, indem man **Durchsuchen ...** drückt und dann die Sortierung in Javascript startet. Der Browser blockiert, bis die Daten sortiert sind, dann wird die Zeit in Textform angegeben.

Vermeiden Sie es, mehr als 10.000 Werte zu laden!

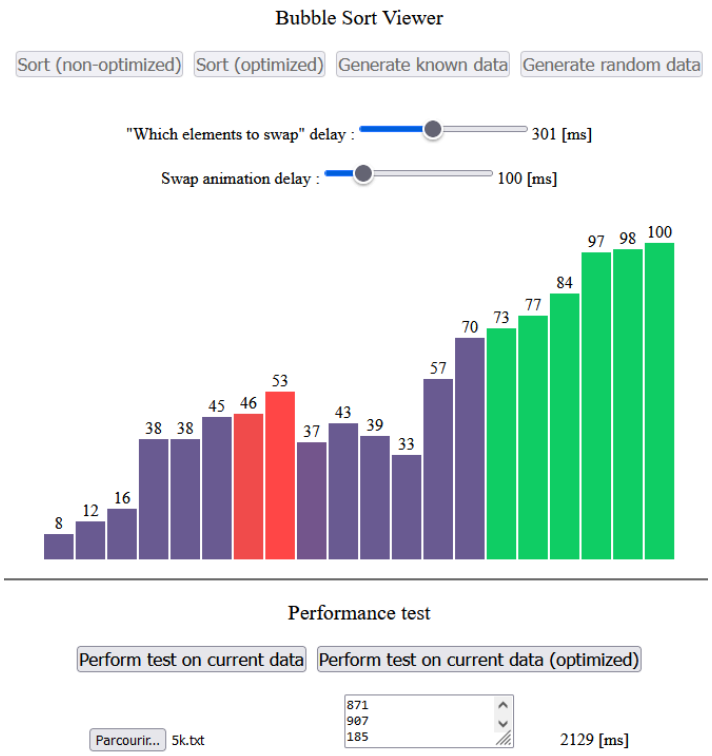
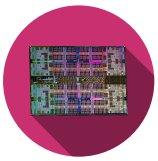


Abbildung 5 - Bubblesort Javascript

4.1.2 Rust

Unten **rust/release**, starten Sie **rust_sorting_xxx** je nach eure Plattform.

Ein Fenster bietet an, die Datendatei zu laden, und zeigt dann die Ergebnisse für den nicht optimierten und den optimierten Algorithmus an. Nur die Ergebnisse des BubbleSort Algorithmus sind für hier wichtig.



```
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]
```

Abbildung 6 - Bubblesort Rust

4.1.3 C

Unten **C/release**, starten Sie **run.cmd**.

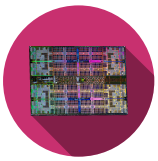
Ein Fenster fragt nach der zu ladenden Datei und gibt dann die Ergebnisse für den nicht-optimierten und den optimierten Algorithmus aus.

```
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]
```

Abbildung 7 - Bubblesort C

4.1.4 Test

1. Testen und ordnen Sie die Ergebnisse für den nicht-optimierten Algorithmus :
   - Javascript : 1k, 5k et 10k, 50k, 10ksorted Elemente
   - C / Rust : 1k, 5k, 10k, 50k, 100k, 10ksorted, 50ksorted Elemente
2. Testen und kommentieren Sie die Ergebnisse für den optimierten Algorithmus :
   - C / Rust : 10ksorted, 50ksorted
3. Geben Sie den Faktor an, um den sich die Zeiten für die verschiedenen Sprachen unterscheiden.
4. Unter welchen Bedingungen ist der optimisierte Code im Vegleich zum nicht-optimierten Code schneller und was könnte der Grund sein?



Schauen Sie sich mit der bereits sortierten Liste, im Javascript Code den visuellen „Bubble Sort Viewer“ bei der Ausführung der Nicht-Optimierten Code im Vergleich des Optimierten Codes.

1. „Generate Random Data“
2. „Sort (optimizied)“ with delays 0ms
3. Set **`Swap Delay`** and **`Swap Animation`** to 100ms
4. Run „Sort (non-optimized)“
5. Run „Sort (optimized)“
6. Compare the two last runs

4.2 Optimisierung

Schlagen Sie einen effizienteren Sortieralgorithmus vor und vergleichen Sie ihn mit dem Bubble Sort. Sie können auch existierende Libraries oder Funktionen verwenden.

Es ist möglich, diesen Algorithmus in einer Sprache Ihrer Wahl zu schreiben. Alternativ ist ein Scala-Template verfügbar unter **car-labs/bem/sorting/algorithms/scala/main.sc**.

Um es schnell und einfach zu nutzen, gehen Sie unter https://www.jdoodle.com/compile-scala-online/ und kopieren Sie seinen Inhalt in den Editor.

Klicken Sie unten auf der Seite auf die Schaltfläche upload und wählen Sie die Datendatei aus, die Sie hochladen möchten. Sobald Sie dies getan haben, ist die Datei zum Lesen verfügbar unter dem Pfad **„/uploads/myfilename.txt“**. Ändern Sie die Zeile `val dataFile = "/uploads/10k.txt";` so, dass sie die zu testende Datei widerspiegelt

Der Code kann durch Drücken der Schaltfläche **Execute** ausgeführt werden. Das Ausgabefenster gibt die Ausführungszeit an, sowie ob die Tabelle richtig sortiert wurde oder nicht.

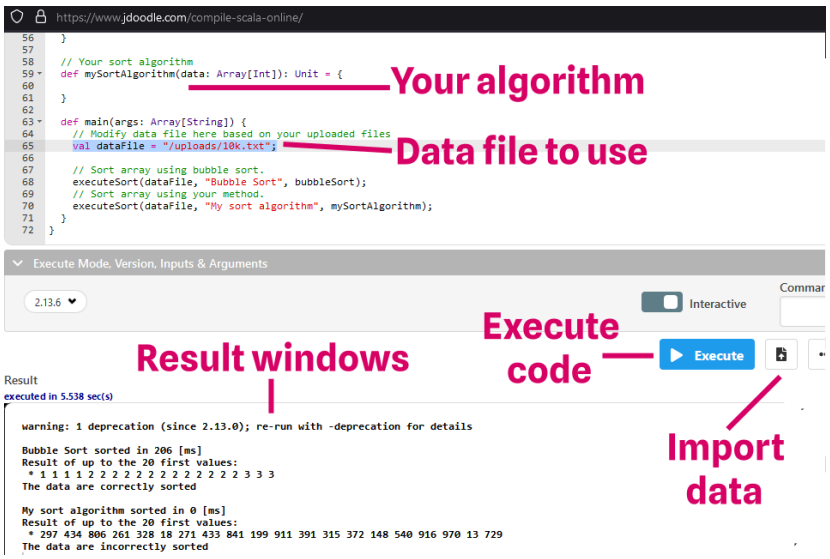
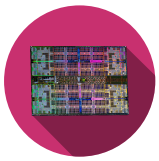


Abbildung 8 - Scala auf JDoodle

4.2.1 Todo

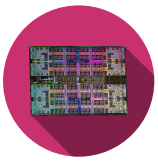
1. Kodieren Sie den Sortieralgorithmus Ihrer Wahl.



2. Testen Sie Ihn für die Standardfälle sowie die besten der Fälle (verwenden Sie dafür die Dateien mit dem Namen **xxxsorted**).
3. Vergleichen Sie sie mit Ihren bisherigen BubbleSort-Ergebnissen.
4. Was sind die Vorteile der gewählten Sprache? Was sind die Nachteile?
5. Nennen Sie den Namen des verwendeten Algorithmus, seine Funktionsweise und:
   - Scala : schliessen Sie Ihre Sortierfunktion in den Bericht ein. Der Rest des vorgegebenen Templates ist nicht erforderlich.
   - Andere Programmiersprachen : fügen Sie den vollständigen Code bei.

4.2.2 Template scala

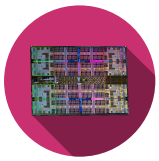
```
object SortOptimization {

  // 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);
  }
}
```

4.2.2.1 *Optionale Aufgabe*

Für die Ambitioniertesten unter ihnen können Sie versuchen das Rust Program **`rust-glidesort`** zu unterbieten. **Nur** in diesem Fall sollten Sie die Datei **`10000k.txt`** verwenden.



Glossar

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

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

OS **– Operating System** 3