a) Betrachten Sie folgendes zweidimensionale Feld (Programmiersprache C):

```
int x[64][64];
```

Der physische Speicher soll aus vier Seitenrahmen bestehen. Jeder Seitenrahmen umfasst 128 Wörter. Eine Integerzahl belegt ein Wort. Ein Programm, das mit dem x Feld arbeitet passt genau in eine Seite und besetzt immer Seite 0. Die Daten werden von den anderen drei Rahmen ein – und ausgelagert. Das x-Feld wird hintereinander weg in den Speicher gespeichert. Dabei kommt x[0][1] hinter x[0][0] im Speicher.

- Welches der beiden folgenden Codeausschnitte wird die wenigsten Seitenfehler erzeugen? Begründen Sie Ihre Lösung.
- Berechnen Sie die Gesamtzahl an Seitenfehlern.

Ausschnitt A
```
for (int j = 0; j < 64; j++)
   for (int i = 0; i < 64; i++)
      x[i][j] = 0
```

Ausschnitt B
```
for (int i = 0; i < 64; i++)
   for (int j = 0; j < 64; j++)
      x[i][j] = 0
```

b) Eine Maschine hat virtuelle 48bit Adressen und physische 32bit Adressen. Wie viele Seiten sind in der Seitentabelle? Eine Seite soll 8 KiB umfassen. Jedes Byte im physischen Speicher soll eine Adresse haben. Vorsicht: eine Information in dieser Aufgabe wird nicht gebraucht – welche?

c) Ein Computer mit 8 KiB Seiten, 256 MiB Arbeitsspeicher und 64 GiB virtuellem Adressraum benutzt invertierte Seitentabellen für seinen virtuellen Speicher. Wie groß sollte die Hashtabelle mindestens sein, damit die Listen in der Tabelle durchschnittlich **weniger** als einen Eintrag haben. Die Größe der Tabelle muss eine Zweierpotenz sein.

d) Fünf Stapelverarbeitungsjobs von A bis E kommen fast zur gleichen Zeit in einem Rechenzentrum an. Ihre geschätzten Laufzeiten sind 10, 6, 2, 4, 8 Minuten. Ihre (statisch eingestellten) Prioritäten sind 3, 5, 2, 1, 4. Dabei ist 5 die höchste Priorität. Bestimmen Sie für jede der folgenden Schedulingstrategien die durchschnittliche Prozessdurchlaufzeit. Vernachlässigen Sie den Aufwand des Prozesswechsels.

I) Round Robin
II) Prioritätsscheduling
III) First Come First Serve, .d.h. der Job, der zuerst kommt, wird als erster ausgeführt. Die Ankunftsreihenfolge soll sein: Zuerst kommt der Job mit 10 Minuten Laufzeit, dann

der mit 6 Minuten, 2 Minuten, 4 Minuten und 8 Minuten. Die Jobs kommen ohne nennenswerte Zeitverzögerung hintereinander an.

IV) Shortest Job First, d.h. von den jeweils übrigen Jobs wird jeweils der ausgeführt, der die kürzeste Laufzeit hat.

Nehmen Sie für I) an, dass das System multiprogrammierbar ist und dass die CPU unter den Jobs fair aufgeteilt wird. Für II) bis IV) nehmen Sie an, dass nacheinander jeweils ein Job bis zum Ende ausgeführt wird. Jeder Job nutzt die CPU zu 100%, wenn er die Ausführungsberechtigung hat.

e) Wir nehmen an, dass es immer genau eine feste Zeit *T* dauert, bis ein Prozess wegen Ein-Ausgabe blockiert wird. Wenn ein Prozess blockiert wird oder wenn sein Quantum vorbei ist, dann wird auf einen anderen Prozess umgeschaltet. Der Umschaltvorgang soll eine Zeit *S* dauern. Diese Zeit ist verlorene Zeit – sie verringert die CPU Effizienz. Die CPU Effizienz η ist folgendermaßen definiert:

$$\eta = \frac{Rechenzeit}{\text{Gesamtlaufzeit}}$$

Wir betrachten Round-Robin –Scheduling mit dem Quantum *Q* .

I) Warum gilt für $Q = \infty$ und $Q > T$ dieselbe Effizienz?

Geben Sie eine Formel oder einen Wert für die CPU Effizienz für jeden der folgenden Fälle an:

II) $Q = \infty$ bzw. $Q > T$
III) $S < Q < T$ (T sehr viel größer als Q)
IV) $Q = S$
V) *Q* ist ca. null

f) Schreiben Sie ein Programm, das den Einfluss von Seitenfehlern auf die Speicherzugriffszeit demonstriert, indem die Zeit pro Zugriff gemessen wird, die benötigt wird, um ein langes Feld zu durchlaufen.
Die folgende Anleitung gilt für Windows 10. Achten Sie darauf, dass Ihre C:-Festplatten Partition nicht zu stark belegt ist. Es sollten möglichst keine weiteren Anwendungen außer den Anwendungen dieser Aufgabe und unbedingt nötigen Anwendungen auf Ihrem Computer laufen. Lassen Sie das Programm in der Version mit Seitenfehlern nicht zu lange laufen, um die Festplatte zu schonen.

Aufgaben:

I) Öffnen Sie den Task Manager (Ctrl-Alt-Del). Gehen Sie auf den Reiter Leistung. Betätigen Sie den Knopf Ressourcenmonitor. Im Ressourcenmonitor gehen Sie auf den Reiter Arbeitsspeicher. Dort sehen Sie eine Tabelle mit den Prozessen. Was ist der Unterschied zwischen Zugesichertem Speicher, Arbeitsspeicher, Freigabefähig, Privat?

II) Unterhalb der Tabelle befindet sich ein horizontaler Balken. Was ist der Unterschied zwischen Speicher - in Verwendung, geändert, Standby und frei?

III) Unter dem Balken gibt es eine Angabe für verfügbaren Speicher. Wir wollen nun Java Prozesse starten, die zusammen mehr als den verfügbaren physischen Speicher beanspruchen. Dadurch müssen immer wieder Daten aus dem Arbeitsspeicher auf die Festplatte ausgelagert werden. Dieses Auslagern wird durch sogenannte Seitenfehler

hervorgerufen (siehe Vorlesung). Damit die Auslagerung erfolgt, müssen die Prozesse regelmäßig ihren gesamten Speicherbereich lesen bzw. beschreiben.

i) Schreiben Sie ein Java Programm mit dem Namen ZugriffsZeiten.java.
ii) In der main Methode legen Sie ein double Feld mit dem Namen zahlenFeld an.
iii) Wie groß muss das Feld sein, damit aller verfügbarer Speicher durch das Feld verbraucht wird? Ein double benötigt 8 byte.
iv) In einer Endlosschleife fragen Sie zunächst die Startzeit jedes Schleifendurchlaufs mit `System.currentTimeMillis()` ab.
v) Dann durchlaufen Sie in einer inneren Schleife das gesamte Feld, und weisen in einer inneren Schleife jedem Feldplatz mit `Math.random()` einen Zufallswert zu.
vi) Warum reicht es wohl nicht, jedem Feldplatz einfach nur die Zahl 0 zuzuweisen, um das Auslagern von Speicher zu provozieren (der Java Compiler besitzt Optimierungen)?
vii) Direkt hinter der inneren Schleife fragen Sie die Endzeit jedes Schleifendurchlaufs der äußeren Schleife wieder mit `System.currentTimeMillis()` ab. Dadurch können Sie mittlere Zugriffszeit auf einen double Wert (plus Zufallszahlberechnungszeit) bestimmen, und auf der Konsole ausgeben.
viii) Gelingt es Ihnen, den Prozess mit einem für Ihr Zahlenfeld ausreichend großen Heap zu starten?
   (1) Öffnen Sie eine Eingabeaufforderung. (Start Knopf – in Eingabefeld „Programme/Dateien durchsuchen“ cmd eingeben). Hier können Sie das Programm kompilieren und ausführen. Sie müssen beim Start der virtuellen Maschine die Option –Xms<Heapgröße> mit angeben.
   `java -Xms1024m ZugriffsZeiten`
   Die Zahl <Heapgröße> steht dabei für die Anzahl an Byte, die für den Heap reserviert sind. Z.B. sind mit
   -Xms1024m
   1024 Megabyte reserviert.
   (2) Bei meinem Rechner mit 16 GB kann ich das ZugriffsZeiten Programm mit einem double Feld nur mit der Dimension 1 Milliarde starten. Dafür wird ein Speicherplatz von 8 GB benötigt, da ein double 8 Byte beansprucht. Die Speicheroption für die virtuelle Maschine muss bei mir dennoch lauten:
   -Xms12G d.h. es werden für den Heap mindestens 12 GB reserviert. Mit weniger Heapsize läuft meine virtuelle Maschine nicht, obwohl mein Array nur 8 GB benötigt.

ix) Betrachten Sie die Spalte Seitenfehler/s im Resourcenmonitor. Gelingt es Ihnen durch Vergrößern des Felds zahlenFeld, die Anzahl der Seitenfehler/s für Ihren java.exe Prozess auf über 10 Fehler/s zu treiben?
   (1) Wenn sich das Byte Array nicht weiter vergrößern lässt, können Sie ihren Computer auch dadurch fordern, indem Sie das Programm ZugriffsZeiten.java mehrfach starten, nämlich solange, bis die Anzahl der Seitenfehler über 10 Fehler/s geht.
   (2) Sie können java.exe durch Crtl-C beenden, wenn das Eingabeaufforderungsfenster den Fokus hat.
x) Wie groß sind die Zeiten pro inneren Schleifendurchlauf, die Ihr Programm ermittelt für
   - 0 Seitenfehler/s
   - mehr als 0 Seitenfehler/s

xi) Wie groß ist die Datei c:\pagefile.sys, bevor ihr Programm startet und während ihr Programm läuft?

xii) Ist die CPU Auslastung gleich 100%, wenn Ihr Programm läuft?
Wenn nein, warum nicht?