Aufgabe 2: naives vs komplexes Sortieren

In dieser Aufgabe werden zwei Algorithmen zum Sortieren implementiert. Bei dem komplexen Algorithmus werden unterschiedliche Strategien implementiert und verglichen. **Dabei sind die Verfahren zu verwenden, die in der Vorlesung vorgestellt wurden!**

Aufgabenstellung

Sei `zahlen` eine endliche Folge von ganzen positiven Zahlen (in Form einer Liste/Reihung, Trennungssymbol ist die Leerstelle/Blank, als Datei `zahlen.dat` vorgegeben bzw. zu erzeugen) gegeben. Diese ist nach der Grösse zu sortieren (Kleinstes Element vorne bzw. links).
Verwenden Sie dazu die im ersten Praktikum implementierte ADT Array (Achtung: ggf. wird der ADT von anderen Teams eingesetzt!).
Implementieren Sie nun folgende drei Algorithmen:

1. **sortNum** als Zahlengenerator: Dieser Algorithmus erzeugt eine vorgegebene Anzahl an positiven ganzen Zufallszahlen oder eine "worst case" Situation: von links nach rechts oder rechts nach links sortiert, und speichert sie in einer Datei `zahlen.dat`.
2. **insertionsort**: Damit der Algorithmus von Quicksort verwendet werden kann, ist die Schnittstelle (wird noch bekannt geben) einzuhalten: semantische Vorgabe: insertionsort: array × pos × pos → array, technische Vorgabe: `public static void insertionsort(AdtArray array, int start, int end)`, wobei die beiden Positionen den Bereich im Array vorgeben, auf den Insertionsort angewendet werden soll, d.h. Insertionsort sortiert nicht immer das ganze Array! Achtung: auch hier wird der Algorithmus mit anderen Teams getauscht werden.
3. **quicksort**: Dieser Algorithmus ist mit dem Pivotelement an linkester Stelle zu implementieren. Sind weniger als 75 Elemente (innerhalb der Rekursion) zu sortieren, ist insertionsort zu verwenden, d.h. ab einer Anzahl von 75 Elementen arbeitet das Programm rekursiv und verzweigt bei weniger als 75 Elementen (für genau diese Elemente!) auf insertionsort. Dieser Wert soll bei der Eingabe zu Testzwecken als Parameter qiswitch übergeben werden. Der Algorithmus ist so zu implementieren, dass die Pivoauswahl einfach verändert werden kann (z.B. durch Rechts, Links, Mitte, Median of 3 und Random): semantische Vorgabe: quicksort: array × method_pivot → array, technische Vorgabe: `public static void quicksort(AdtArray array, PivotMethod pivotMethod, int qiswitch)`, wobei method_pivot die Möglichkeiten left/middle/right/medianof3/random zulässt.
4. Zudem sind für insertionsort und quicksort **JUnit-Tests** zu implementieren, die einfach ausgetauscht werden können (als insertionJUt.jar bzw. quickJUt.jar speichern). Beachten Sie: die Tests sollten auch Belastungen (große Datenmengen) und Grenzfälle beinhalten. Die Tests werden ausgetauscht. **Deshalb dürfen die Tests nicht von der internen Darstellung der ADT Array abhängen**, sondern nur rein die Funktion betreffen! Die durchgeführten Tests sind zu dokumentieren (als *.pdf) und gehören zur Abgabe: Testkonzept, Testspezifikation und Testbericht zur eigenen Software und zu mindestens einer Software eines anderen Teams.
5. Implementieren Sie eine Version der Algorithmen, die die Laufzeit misst (insertionsort und quicksort getrennt, d.h. also quicksort abzüglich der Zeit, die dort insertionsort benötigt).
6. Implementieren Sie eine Version der Algorithmen, die die Anzahl der lesenden und schreibenden Zugriffe auf das Array zählt (insertionsort und quicksort getrennt, d.h. also quicksort abzüglich der Zugriffe, die insertionsort durchführt).
7. Führen Sie (**auf den Rechnern des Labors wegen der Vergleichbarkeit**) mit den unter

5. und 6. implementierten Versionen aussagenkräftige Messungen durch, mit der Sie die Laufzeitkomplexität der einzelnen Algorithmen nachweisen können (best/worst/average case). Zudem ist die "Wechselgrösse" von 75 Elementen durch Tests zu bestätigen oder eine andere Größe zu ermitteln. **Erstellen Sie ein pdf,** indem die Messungen dokumetiert werden (Versuchsaufbau, Resultate, Interpretation, geforderte Nachweise etc.).

Abnahme

Da die Aufgaben des Praktikums sehr frühzeitig im WWW zur Verfügung stehen, wird die Abnahme stark auf eine **vorbereitende Arbeit** aufgebaut.

Bis Mittwoch Abend 20:00 Uhr vor Ihrem Praktikumstermin ist eine erste Skizze der Aufgabe als *.pdf Dokument (Dokumentationskopf nicht vergessen!) mir per E-Mail zuzusenden. Ggf. können offene Fragen mit gesendet werden. Die Skizze **muss** beschreiben, wie Sie sich die Realisierung denken. Als erfolgreich wird eine Skizze bewertet, wenn Ihre Kenntnisse bzgl. der gestellten Aufgabe eine erfolgreiche Teilnahme an dem Praktikumstermin in Aussicht stellen.

Am Tag vor dem Praktikumstermin bis **20:00 Uhr** : aktuellen Stand (als *.zip) zusenden.

Am Tag des Praktikums findet eine Befragung von Teams statt. Die **Befragung muss erfolgreich absolviert werden**, um weiter am Praktikum teilnehmen zu können. Ist die Befragung nicht erfolgreich, gilt die Aufgabe als nicht erfolgreich bearbeitet. Als erfolgreich wird die Befragung bewertet, wenn Ihre Kenntnisse bzgl. der gestellten Aufgabeausreichend oder besser sind. Dazu gehört insbesondere eine mindetsens ausreichende Kenntnis über Ihren gesamten Code. Bei der Befragung handelt es sich nicht um die Abnahme.

Abgabe: Unmittelbar am Ende des Praktikums, spätestens bis 19:00 Uhr am selben Tag, ist von allen Teams der Code abzugeben. Zu dem Code gehören die Sourcedateien, die ggf. erzeugten *.log etc. Dateien, die während der Tests erzeugt wurden, und eine Readme.txt Datei, in der ausführlich beschrieben wird, wie die Software zu starten ist! Zudem ist der aktuelle Dokumentationskopf abzugeben. Die Dateien sind als *.zip Ordner (mit cc an den/die Teampratner_in) per E-Mail abzugeben. Die Abgabe gehört zu den PVL-Bedingungen und ist einzuhalten, terminlich wie auch inhaltlich!

Wird eine Aufgabe nicht erfolgreich bearbeitet, gilt die **PVL** als **nicht bestanden**. Damit eine Aufgabe als erfolgreich gewertet wird, muß die Befragung als erfolgreich gewertet werden sowie die Abgabe abgenommen worden sein. **Alle gesetzten Termine sind einzuhalten**. Dies ist notwendig, da sonst erhebliche zeitliche Verzögerungen stattfinden würden.