Übungsblatt 7

1. Im Moodle finden Sie eine Datei `SortingAlgorithms.cpp`, die drei Sortieralgorithmen für Arrays vom Typ `unsigned int` enthält. Die Klasse greift auf die bekannte `Color`-Klasse zu, die Sie in Ihr Projekt einbinden müssen.

   Außerdem finden Sie im Moodle die Template-Klasse `BinaryRelation.h`, die nur die Interface-Methode mit dem Kopf `bool inRelation(S a, T b)` enthält. Die Idee dabei ist, dass `BinaryRelation` von Algorithmen zum Vergleich zweier beliebiger Objekte herangezogen werden kann, ohne dass die Algorithmen wissen müssen, was genau verglichen wird. Unterklassen von `BinaryRelation` definieren dann jeweils eine konkrete Methode zum Vergleichen von Objekten.

   (a) Schreiben Sie modifizierte, flexiblere Template-Varianten von `insertionSort` und `quickSort`, die das Folgende leisten:
   - Sie sind jeweils dazu in der Lage Arrays eines beliebigen, generischen Typen `T` zu sortieren (Sie benötigen also einen Template-Parameter `T`).
   - Als zusätzliches Argument erhalten die Funktionen eine Referenz auf eine durch `T, T` realisierte Instanz von `BinaryRelation`. Eine so definierte Relation kann also zwei Elemente vom Typ `T` zueinander in Relation setzen. Die Algorithmen sollen die Arrays dann nicht mehr gemäß der $\leq$-Relation sortieren, sondern gemäß der durch die `BinaryRelation` gegebene Relation.

   (b) Schreiben Sie eine Klasse `InOrderIntRelation` die von

   ```
   BinaryRelation<unsigned int, unsigned int>
   ```

   abgeleitet ist und die deren `inRelation`-Methode konkret so umsetzt, dass genau dann `true` zurückgegeben wird, wenn das erste Argument kleiner oder gleich dem zweiten ist.

   Können Sie die Klasse so designen, dass es im gesamten Programmverlauf stets nicht mehr als eine einzige Instanz dieser Klasse geben kann?

   Rufen Sie die Sortieralgorithmen, die Sie in Aufgabe 1a geschrieben haben, mit dieser (einer) Instanz von `InOrderIntRelation` auf und lassen Sie einige zufällig erzeugte Arrays sortieren um Ihren Code zu testen!

   Sie finden Funktionen, die zufällig befüllte Arrays erzeugen, in `SortingAlgorithms.cpp`, sowie Funktionen zur Ausgabe der Arrays auf Konsole.

   (c) Schreiben Sie, analog zur Vorgehensweise der letzten Teilaufgabe, eine Klasse `ReverseOrderIntRelation`, mit deren Hilfe Sie Arrays anders herum sortieren können. Schreiben Sie aber keine neuen Sortierfunktionen, sondern benutzen Sie die vorhandenen mit mit der neuen Relation!

   (d) Schreiben Sie, analog zur Vorgehensweise der letzten Teilaufgaben, Klassen zur Sortierung von Farben (benutzen sie die Klasse `Color` der letzten Übungsblätter). Schreiben Sie je eine Klasse zur Sortierung der Farben nach ihrem Farbanteil (rot, grün, blau) und eine zur Sortierung nach ihrer Helligkeit (Summe der drei Farbanteile).

2. Womöglich ist Ihnen bekannt, dass der Algorithmus Insertion-Sort eine Worst-Case-Laufzeit und eine Average-Case-Laufzeit von $O(n^2)$, Quick-Sort aber eine Average-Case-Laufzeit von $O(n \cdot \log n)$ besitzt. Man kann also erwarten, dass Arrays im Durchschnitt mit Quick-Sort deutlich schneller sortiert werden können als mit Insertion-Sort. Bei den obigen Abschätzungen geht es aber um den asymptotischen Fall, d.h. Quick-Sort ist vor allem bei langen Arrays schneller. Bei sehr kurzen Arrays ist aber Insertion-Sort schneller.

   (a) Schreiben Sie eine Template-Funktion `sort` mit Template-Parameter `T`, die ein Array vom Typ `T` sortiert. Wie oben soll diese Funktion drei Argumente bekommen: Das eben genannte Array, einen `int`-Wert, der seine Größe angibt, sowie eine Referenz auf eine `BinaryRelation`, die zwei Elemente vom Typ `T` vergleichen kann. Weisen Sie diesem letzten Funktionsargument als Defaultwert eine Instanz von `InOrderIntRelation` zu. Die Funktion soll die eigentliche Sortierung an die beiden Algorithmen Insertion-Sort und Quick-Sort delegieren und zwar an ersteren, falls die Arraygröße klein ist (z.B. kleiner oder gleich 16) und ansonsten an Quick-Sort.

(b) Für `unsigned int`-Arrays gibt es einen Sortieralgorithmus, der sogar nur die Laufzeit $O(n)$ benötigt, vorausgesetzt, dass im Array keine Zahlen vorkommen, die deren Wert wesentlich größer sind, als die Größe des Arrays. Spezialisieren Sie daher Ihre Template-Funktion `sort` so, dass für den Fall `T = unsigned int` weder Insertion-Sort, noch Quick-Sort, sondern Counting-Sort aufgerufen wird.

Testen Sie alle Ihre Implementationen!