Universität des Saarlandes
FR 6.2 Informatik
Prof. Dr. Hans-Peter Lenhof
Dipl. Inform. Andreas Hildebrandt

2. Übung zur Vorlesung Programmierung II, SS 2003

Aufgabe 1: So ein Zufall

Mit der Funktion `rand()` steht Ihnen ein *Pseudozufallszahlengenerator* zur Verfügung. Diesen können Sie verwenden, wenn sie zufällige Zahlenfolgen erzeugen müssen. Um zufällige Integerzahlen zu generieren, müssen Sie diesen Generator zuerst mit einer beliebigen ganzen Zahl initialisieren, indem Sie die Funktion `srand(int seed)` aufrufen. Danach liefert jeder Aufruf von `rand()` eine weitere Zufallszahl zurück (da es sich um Pseudozufallszahlen handelt, erhalten Sie bei Verwendung des selben Startwerts auch jedesmal die selben Zahlen!).
Erstellen Sie ein Programm, welches einen `vector<int>` mit zufälligen Zahlen füllt, wobei Sie die Länge des Vektors und den *seed*, mit dem Sie den Generator initialisieren, als Kommandozeilenargumente übergeben lassen.
Finden Sie zusätzlich mit Hilfe des Unix-Komandos `man` heraus, wie Sie mit `srand48(long int seed)` und `drand48()` zufällige Gleitkommazahlen erzeugen können.

Aufgabe 2: InsertionSort

Auf dem letzten Übungsblatt haben Sie den STL - Algorithmus `sort` zum Sortieren eines `vector` verwandt. In dieser Übung sollen Sie nun selbst einen einfachen Sortieralgorithmus implementieren, nämlich das sogenannte *InsertionSort-Verfahren.*
InsertionSort sortiert ein Array $V = (v_1, \ldots, v_n)$, indem es das jeweils erste noch nicht betrachtete Element v_i aus dem Teilarray $(v_i, \ldots, v_n)$ entfernt und an der richtigen Stelle in das *schon sortierte Teilarray* $(v_1, \ldots, v_{i-1})$ einsortiert.
Implementieren Sie InsertionSort für den Datentyp `vector<int>`.

Aufgabe 3: MergeSort

Ein effizienteres Sortierverfahren liefert der sogenannte *MergeSort-Algorithmus.* MergeSort ist ein typischer *Divide, Conquer, and Combine* – Algorithmus, bei dem der divide-Schritt daraus besteht, das zu sortierende Array V in zwei etwa gleich große Teilarrays V_1 und V_2 zu zerteilen und diese rekursiv zu sortieren. Im combine - Schritt werden V_1 und V_2 schließlich zum sortierten Array V_3 zusammengefügt, in dem so lange das jeweils kleinste Element aus V_1 bzw V_2 entnommen und hinten an V_3 angefügt wird, bis V_1 und V_2 beide leer sind.
Implementieren Sie Mergesort für den Datentyp `vector<int>`.

Aufgabe 4: Laufzeitvergleich, Zusatzaufgabe

Im Laufe dieser Vorlesung werden Sie lernen, wie man mit den Mitteln der Laufzeitanalyse die erwartete Anzahl von Rechenschritten berechnet, die ein Algorithmus für eine bestimmte Eingabe benötigt. Sie werden dann leicht zeigen können, dass InsertionSort im schlimmsten Fall $\Theta(n^2)$ Schritte benötigt, Mergesort $\Theta(n \log(n))$. Um ein Gefühl dafür zu bekommen, was diese Aussage bedeutet, sollen Sie in dieser Aufgabe die Laufzeit Ihrer Implementierungen aus den Aufgaben 2 und 3 visualisieren.

Zählen Sie dazu in beiden Implementierungen die Anzahl von Vergleichen der Art $a < b$, $b <= c$ etc., die zum Sortieren einer bestimmten Eingabe benötigt werden, in einer Variable mit. Erzeugen Sie in einer Schleife Vektoren der Länge N von 10 bis 4010, wobei Sie die Länge in Schritten von 500 erhöhen.

Führen Sie für jede Länge N 20 Experimente durch, bei denen Sie den Vektor mit Zufallszahlen füllen und anschliessend sortieren. Mitteln Sie die Anzahl der dabei benötigten Vergleichsoperationen. Erzeugen Sie eine Datei, die für jede Länge N eine Zeile der Form `N Mittelwert` enthält und plotten Sie den Inhalt dieser Datei mit dem tool *gnuplot*.
