Sortier- und Suchalgorithmen - Projekt

1 Teil 1 - Sortieren

1.1 Basisimplementierung

Schreiben Sie ein Programm indem sie die folgenden vier Algorithmen die ein Array aufsteigend sortieren sollen implementieren:

a) Mergesort
b) Quicksort
c) Bubblesort
d) Insertionsort

Das Array soll mit Zufallswerten initialisiert werden, wobei die Zahlen zwischen -32.768 und 32.767 liegen sollen. Die Initialisierung soll in einer eigenen Funktion gemacht werden. Verwenden Sie dafür Pointer in einer geeigneten Art und Weise. Führen Sie die Sortierfunktionen mit drei unterschiedlich großen Arrays aus (8, 16, und 64 Elemente). Implementieren Sie eine Funktion, die überprüft ob das Ergebnis tatsächlich sortiert ist. Geben Sie im Fehlerfall eine entsprechende Meldung aus.

Als Ergebnis sind die unsortierten und sortieren Arrays in der Konsole auszugeben. Stellen Sie dabei sicher, dass in jeder Zeile nur maximal 15 Zahlen ausgedruckt werden um die Leserlichkeit zu erhöhen.

1.2 Leistungsvergleich

Schreiben Sie ein Programm, mit dem die Sortieralgorithmen in ihrem Laufzeitverhalten verglichen werden können. Führen Sie die Sortierfunktionen mit unterschiedlich großen Arrays und unterschiedlichen Daten aus. Um die Performanz beurteilen zu können, soll die jeweilige Laufzeit der Sortierung getrackt werden – für jede Größe der Arrays. Um eine Vergleichbarkeit sicher zu stellen, sollen in einem Durchlauf alle Algorithmen mit den gleichen Ausgangsdaten versorgt werden.

- Arraygrößen: 8, 32, 128, 512, 2048, 8192, 32768[1]
- Datensätze:
    - aufsteigend
    - absteigend
    - zufällig

Verwenden Sie dafür die *clock()* Funktion um die Laufzeit der jeweiligen Sortierung zu speichern..

Geben Sie zusätzlich am Ende des Programms alle Laufzeiten von allen Sortierfunktionen für die jeweilige Arraygröße aus. Geben Sie diese Informationen tabellarisch aus um auf den ersten Blick eine Beurteilung hinsichtlich der Performanz treffen zu können. Wählen Sie dazu eine geeignete Maßeinheit.

[1] Sie können auch gerne mit größeren Werten experimentieren – Die Abgabeversion soll aber nur diese Werte benutzen UND größere Werte bitte nicht auf annuminas ausführen.

1.3 Vergleichbarkeit

Schreiben Sie ein Programm, das zeigt, dass einmalige Codedurchläufe nur beschränkt für Zeitvergleiche aussagekräftig[2] sind. Führen Sie den Bubblesort mit dem gleichen zufälligen Set an Daten mit einer Arraygröße von 2000 zwanzigmal aus und vergleichen Sie die Laufzeiten.

Geben Sie am Ende des Programms alle Laufzeiten sowie minimum, maximum und durchschnittliche Laufzeiten aus.

1.4 Verkette Listen

Schreiben Sie ein Programm, dass die Laufzeitunterschiede bei der Verwendung eines Arrays und einer einfach verketten Liste zeigt. Implementieren sie eine zweite Version des Insertionsort mit einer einfach verketten Liste anstelle eines Arrays und einer geeigneten Initialisierung. Der Austausch der Position zweier Werte soll in diesem Fall durch Umverketten durchgeführt werden (im Gegensatz zum Wertetausch zwischen Element bei einem Array).

Vergleichen Sie die beiden Implementierungen des Insertionsort-Algorithmus mit dem gleichen zufälligen Datensatz und einer Arraygröße von 2048. Neben der reinen Sortierperformanz vergleichen Sie auch die Laufzeiten der Initialisierung.

Als Ergebnis sind die unsortierten und sortieren Arrays in der Konsole auszugeben. Stellen Sie dabei sicher, dass in jeder Zeile nur 15 Zahlen ausgedruckt werden um die Leserlichkeit zu erhöhen. Geben Sie auch hier die jeweiligen Laufzeiten für die jeweilige Arraygröße aus. Geben Sie diese Informationen tabellarisch aus um auf den ersten Blick eine Beurteilung hinsichtlich der Performanz treffen zu können. Wählen Sie dazu eine geeignete Maßeinheit.

2 Teil 2 - Suchen

2.1 Sortieren und Suchen – Selbst Implementiert

Schreiben Sie ein Programm, das Sortieren und Suchen verbindet. Erweitern Sie dazu einen[3] der von Ihnen implementierten Suchalgorithmen in einer neuen Version so, dass jedes Element in dem Array um einen String erweitert wird. Lösen Sie das über eine *struct*. Generieren Sie 400 Elemente dieser Struktur und initialisieren Sie die Integer und String Variablen mit Zufallswerten. Die Strings können dabei eine beliebige Länge haben.

Geben Sie dem Benutzer die Möglichkeit nach einem Element der Liste zu suchen (entweder nach Schlüssel, also dem Integer oder nach dem String-Wert). Implementieren Sie die Suche nach dem Element mit einer selbstgeschriebenen binären Suchfunktion. Zeigen Sie dem Benutzer abschließend das Ergebnis der Suche (hit oder miss). Es soll möglich sein wiederholt zu suchen.

2.2 Sortieren und Suchen – StdLib

Schreiben Sie ein Programm, das Sortieren und Suchen verbindet. Verwenden Sie die struct und die Initialisierung von der selbst implementierten Version. Anstelle der selbst implementierten Sortier- und Suchfunktionen verwenden Sie diesmal die Funktionen aus der StdLib:

- **void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))**

[2] Zumindest wenn der Zeitvergleich ohne weitere Maßnahmen auf einem Multitasking-Betriebssystem durchgeführt wird.
[3] Idealer Weise den Quicksort.

- **void *bsearch(const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const void *, const void *))**

Geben Sie dem Benutzer die Möglichkeit nach einem Element der Liste zu suchen (nach dem Schlüssel). Zeigen Sie dem Benutzer abschließend das Ergebnis der Suche (hit oder miss). Es soll möglich sein wiederholt zu suchen.

2.3 Sortieren und Suchen – Leistungsvergleich

Schreiben Sie ein Programm, dass Ihre selbstimplementierten Sortier- und Suchfunktionen mit denen aus der StdLib vergleicht. Erzeugen Sie dazu wieder 30000 Elemente dieser Struktur (Die maximale Länge der Strings auf 10 begrenzen). Erzeugen Sie weiters ein Array mit 500 zufälligen Integer-Werten nach denen gesucht werden soll.

- Suchen Sie mit den selbstimplementierten Funktionen die 500 Werte im ursprünglich zufällig sortierten Array. Messen Sie die Ausführungszeit der Sortierung und der 500 Suchvorgänge mit clock().
- Suchen Sie mit den Library-Funktionen die 500 Werte im ursprünglich zufällig sortierten Array. Messen Sie die Ausführungszeit der Sortierung und der 500 Suchvorgänge mit clock().

Geben Sie zusätzlich am Ende des Programms die beiden Laufzeiten aus um auf den ersten Blick eine Beurteilung hinsichtlich der Performanz treffen zu können. Wählen Sie dazu eine geeignete Maßeinheit.

3 Projekt - Beurteilung

3.1 Punkte

Aufgabe	2er Gruppe	3er Gruppe
1.1	25	25
1.2	6,5	6,5
1.3	-	6,5
1.4	-	12,5
2.1	12,5	12,5
2.2	6	6
2.3	-	6
Max. Gruppenpunkte	**50**	**75**
Einzelergebnis	**Gruppenpunkte/2**	**Gruppenpunkte/3**

3.2 Beurteilung

- Code
  - Korrekt Implementierung aller Funktionalität wie oben beschrieben.
  - Der Code muss dokumentiert sein.
  - Verwenden Sie für die Implementierung die geeigneten Datenstrukturen.
  - Der Speicher soll dynamisch alloziert werden und entsprechend wieder freigegeben werden.
  - Eine falsche Handhabung mit Pointern führt ebenfalls zu Punkteabzügen

    - Warnungen bei einer Kompilierung mit -Wall und -Wextra führen zu automatischen Punkteabzügen
    - Zielplattform ist annuminas – dort muss der Code korrekt Kompilieren und ausführbar sein[4].
- Projektstruktur
    - Sinnvolle Verzeichnisstruktur
    - Sinnvolle Code-Wiederverwendung
    - Einheitliche Dateitemplates
    - Einheitliche Namenskonventionen
- Toolchain
    - Es muss ein bash-script mit abgegeben werden, dass die benötigten Executables erzeugt (alternativ ist natürlich auch ein makefile/cmakefile möglich)
    - Es muss eine Anleitung geben mit welchem Aufruf (Executable + ggf. Parameter) welcher Aufgabe entspricht.
    - Weiters sollen die beteiligten Studenten mit Name, User und Matrikelnummer in der Anleitung stehen.
- Abgabe
    - Es muss der vollständige C-Sourcecode (c & h-Dateien) abgegeben werden.
    - Die Abgabe besteht aus einem tar-archiv das ausschließlich aus dem Shellscript (bzw. makefile), der Anleitung und den c-Sourcen (c- und header-Dateien) besteht
    - Es wird die zuletzt hochgeladene Version beurteilt
- Automatisches *Nicht Genügend*
    - Falls das abgegebene Programm nicht kompiliert, führt das automatisch zu einer Beurteilung mit *Nicht Genügend*
    - Plagiate[5][6] werden automatisch mit *Nicht Genügend* beurteilt[7] (für die ganze Gruppe).
    - Unentschuldigte Nichtdurchführung des Abgabegesprächs bedeutet automatisch ein *Nicht Genügend* (für die Person die unentschuldigt nicht angetreten ist).
    - Das tar-archiv nicht rechtzeitig vor der Abgabe in moodle hochladen bedeutet automatisch ein *Nicht Genügend*
    - Nichtabgeben bedeutet automatisch ein *Nicht Genügend* für die ganze Gruppe.

3.3 GRUPPENARBEIT

Das Projekt ist eine Gruppenarbeit für 3 Personen mit maximal zwei 2-Personengruppen aus numerischen Gründen. Bei den 2-Personengruppen entfallen die drei Aufgaben „1.3 Vergleichbarkeit“, „1.4 Verkette Listen“ und „2.3 Sortieren und Suchen – Leistungsvergleich“. Die Gruppenzusammenstellung ist in der 12. Präsenzeinheit bekannt zu geben. Nachträgliche Änderungen sind nach Absprache mit der LV-Leitung möglich.

[4] Es muss nicht auf annuminas entwickelt werden.

[5] Da es in den meisten Fällen nicht nachträglich feststellbar ist, wer wen plagiiert hat sind alle Gruppen, die in den Plagiatsfall verwickelt sind mit *Nicht Genügend* zu beurteilen.

[6] Es werden die Abgaben aus allen BIC-TI Vorlesungen angeglichen.

[7] Da zu unterschiedlichen Zeitpunkten abgegeben werden kann (die letzten beiden Termine und ggf. der Nachprüfungstermin) kann es zu nachträglichen Feststellungen von Plagiaten und somit nachträglichen Beurteilungskorrekturen kommen.

3.4 ABGABEGESPRÄCH

In den letzten beiden Einheiten findet ein 5 bis 10 minütiges Abgabegespräch statt. Sie müssen dabei in der Lage sein den abgegebenen Code erklären zu können und die Konzepte der verwendeten Algorithmen beherrschen. Gruppen treten gemeinsam zu dem Abgabegespräch an[8].

Die Abgabegespräche finden an den Mittwochen 19.01.2022 sowie 26.01.2022 statt. Die Abgabe des tar-archivs auf moodle muss am vorhergehenden Montag bis spätestens 18:00 erfolgen (Montag 17.01. für Abgaben am 19.01. sowie Montag 24.01. für Abgaben am 26.01.).

[8] Es ist gewünscht, dass eine Gruppe gemeinsam Antritt. Sollte eine Gruppe nicht vollständig anwesend sein, so wird mit den Anwesenden die Abgabe durchgeführt.