Praktikum **Algorithmik SS 2007**

Aufgabe 4: MERGESORT im Pseudocode

Aufgabe 5: Analyse von MERGESORT mit dem Master-Theorem

Aufgabe 6: Nicht-deterministischer Algorithmus RANDOMIZEDQUICKSELECT

Aufgabe 7: Programmierung und Analyse von RANDOMIZEDQUICKSELECT

Aufgabe 8: Die Ackermann-Funktion

Name: ..... Matr-Nr: .....

Datum: …… Unterschrift des Dozenten (wenn bestanden): ……..……

Lernziele der Aufgaben 4 u. 5: Divide & Conquer – Prinzip, Anwendung des Master-Theorems zur Analyse eines D&C-Algorithmus

Aufgabe 4 (**MERGESORT**)

Formulieren Sie MERGESORT(A, p, r) rekursiv als Divide&Conquer-Algorithmus. Unterscheiden Sie in Ihrem Algorithmus deutlich die drei Schritte *Divide*, *Conquer* und *Combine*.

Hinweise: A = (A[1], A[2], ..., A[n]) ist die zu sortierende Folge, p ist der Index der ersten Elements und r der Index des letzten Elements der Folge A.

Gegeben ist eine Prozedur MERGE(A, p, q, r), die zwei Folgen (A[p], ..., A[q]) und (A[q+1], ..., A[r]) zu einer sortierten Folge (A[p], ..., A[r]) mischt.

Schreiben Sie MERGESORT im Pseudocode. Erklären Sie die Prozedur MERGE anhand des Pseudocodes.

Aufgabe 5 (**Analyse von MERGESORT**)

a) Identifizieren Sie im Pseudocode die Schritte *Divide*, *Conquer* und *Combine* und ermitteln Sie für jeden dieser Schritte die Zeitkomplexität.

b) Stellen Sie mit den Überlegungen aus a) eine Differenzgleichung für MERGESORT auf.

c) Lösen Sie die Differenzgleichung mit Hilfe des in der Vorlesung besprochenen Mastertheorems.

Lernziele der Aufgaben 5 u. 6: Am Beispiel des Algorithmus RANDOMIZEDQUICKSELECT soll eine Decrease-and-Conquer-Strategie (auch Prune-and-Conquer genannt) angewendet werden (→ Kapitel 1.3 Algorithmische Entwurfsmuster). Simulation eines schnellen nicht-deterministischen Algorithmus.

Aufgabe 6 (Verstehen von RANDOMIZEDQUICKSELECT)

Gegeben ist eine unsortierte Folge von n Elementen, auf der eine Ordnungsrelation definiert ist. Die Aufgabe besteht darin, das k.-kleinste Element in der Folge zu finden. Ein zufallgesteuerter Algorithmus RANDOMIZEDQUICKSELECT wird eine mittlere Laufzeit von O(n) haben, wobei der Algorithmus mögliche Entscheidungen zufallgesteuert trifft. Über die Verteilung der Eingabedaten muss keine statistische Annahme gemacht werden. Im Worst Case wird die Laufzeit $O(n^2)$ betragen. Mit dem zufallgesteuerten Algorithmus RANDOMIZEDQUICKSELECT wird hier ein schneller „nicht-deterministischer" Algorithmus simuliert.

Idee des Algorithmus

Gegeben sei eine unsortierte Folge S von n vergleichbaren Elementen und eine Integerzahl $k \in [1, n]$. Wähle zufällig ein Element x aus S und benutze dieses als Pivot-Element, um S in die drei Subsequenzen L, E und G zu teilen. L speichert alle Elemente von S, die kleiner als x, E alle die gleich x und G alle die größer als x sind. Dieser Schritt heißt **Decrease**- oder auch **Prune**-Schritt (prune = abschneiden). Anhand des Vergleichs von k mit der Kardinalität |L|, |E| und |G| der drei Mengen wird entschieden, mit welchen Parameterwerten für S und k der Algorithmus RANDOMIZEDQUICKSELECT rekursiv aufgerufen wird. Der Pseudocode der Algorithmus-Idee sieht wie folgt aus:

Algorithmus RANDOMIZEDQUICKSELECT(S, k):
Input:Sequenz S von n vergleichbaren Elementen und Integer $k \in [1, n]$
Output: Das k-kleinste Element von S
if n = 1 **then**
return das (erste) Element von S
wähle ein Element x zufällig aus S aus
entferne alle Elemente aus S und speichere sie in den Folgen:
- L, speichert alle Elemente aus S kleiner als x
- E, speichert alle Elemente aus S gleich x
- G, speichert alle Elemente aus S größer als x

if $k \leq |L|$ **then** ... ?
else if $k \leq |L| + |E|$ **then** ... ?
else RANDOMIZEDQUICKSELECT(G, k - |L| - |E|)

Bearbeiten Sie folgende Teilaufgaben:

- Erklären Sie, warum die Brute-Force-Methode zur Lösung des Select-Problems $O(n^2)$ Zeit benötigt!

- Erklären Sie genau die Idee des beschriebenen Algorithmus RANDOMIZEDQUICKSELECT, und vervollständigen Sie den Pseudocode an den mit Fragezeichen versehenen Stellen. Für jede der drei Fallunterscheidungen findet ein rekursiver Aufruf statt. Wie lauten die Parameter dieser Aufrufe?

- Worin besteht der nicht-deterministische Kern des Algorithmus RANDOMIZEDQUICKSELECT? An welchen Stellen trifft der Algorithmus „selbst" Entscheidungen, die zu einer sehr schnellen Lösung des Select-Problems führen? Bitte genau erklären!!

- Führen Sie den Algorithmus von Hand auf Papier für die folgende Menge S und k = 9 aus. Untersuchen Sie jeweils für die Wahl von x die Fallunterscheidungen, und schreiben Sie jeden Rekursionsaufruf mit den geeigneten Parametern auf.

  S = { 3, 9, 15, 8, 5, 17, 10, 1, 13, 6, 20, 11, 15, 25, 29, 2, 3, 14, 4, 3}

  Zur Konstruktion der beiden rekursiven Aufrufe versuchen Sie, Antworten auf folgende zwei Fragen zu finden:

  - Welcher Zusammenhang besteht zwischen der Größe der Menge L und dem k.-kleinsten Element von S?
  - Was kann man über das k.-kleinste Element aussagen, wenn k = |L| + |E|?

- Diskutieren Sie die Laufzeit-Performance des Algorithmus RANDOMIZEDQUICKSELECT.

Aufgabe 7 (Programmieren, Testen und Analysieren von RANDOMIZEDQUICKSELECT)

Programmieren Sie den Algorithmus RANDOMIZEDQUICKSELECT und geben Sie in jedem Rekursionsschritt die Mengen L, E und G sowie deren Kardinalität auf dem Bildschirm aus.

- Erzeugen Sie die Startmenge S mit Hilfe eines Zufallszahlengenerators. Folgende Parameter zur Erzeugung von S geben Sie interaktiv ein
  - die Größe von S
  - den Wertebereich von S, z.B. [10, 8888]
  - den Wert k
- Geben Sie das Pivotelement $x \in S$ vor jeder Aufteilung S in die Mengen L, E und G, d.h. in jedem Rekursionsschritt interaktiv ein

- Führen Sie den Blackbox-Power-Test für RANDOMIZEDQUICKSELECT durch und zeichnen Sie die Laufzeiten mit einem Plot-Programm. Erklären und dokumentieren Sie die Ergebnisse.

Aufgabe 8 (Die Ackermann-Funktion)

Die Ackermann-Funktion ist quasi eine Erweiterung der einfachen Funktionen „Additionen“, „Multiplikation“ und „Exponentialfunktion“. Sie ist wie folgt definiert:

$$
\begin{aligned}
A(0, n) &= n+1 \\
A(m+1, 0) &= A(m, 1) \\
A(m+1, n+1)) &= A(m, A(m+1, n))
\end{aligned}
$$

Die Ackermann-Funktion wächst stärker als jede *primitiv rekursive* Funktion, dies sind Funktionen mit einem einfachen Rekursionsschema. Die Ackermann-Funktion hat aber ein *verschachteltes Rekursionsschema*, d.h. das Argument für den rekursiven Aufruf wird selbst durch einen rekursiven Aufruf bestimmt. Das hat gravierende Folgen für das Wachstum der Funktion.

Programmieren Sie die Ackermann-Funktion und stellen Sie die Ergebnisse in einer Tabelle dar. Was beobachten Sie?