Algorithmen & Datenstrukturen

Übungsstunde 4

Michelle Honegger - CAB G59 - 13.10.2024

Miniquiz 3

- Passwort: **searching6**
- 10 Fragen
- 11 Minuten
- 0.1 Punkte pro richtige Antwort

Programm

- Miniquiz
- Rückmeldung Serie 2
- Besprechung Aufgaben
- Theory Recap
- Prüfungsaufgabe HS19 T1 g) FS19 T1 c)

Rückmeldung Serie 2

- Bei Induktion nicht auf beide Seiten die Hypothese anwenden!
- Peergrading: ist ok, wenn alles richtig ist und die Darstellung auch schön, schreibt trotzdem noch was hin.

Besprechung Miniquiz

Seien $f, g : \mathbb{N} \to \mathbb{R}^+$ zwei Funktionen. Dann gilt $\max\{f(n), g(n)\} \leq O(f(n) + g(n))$.

- Wahr
- Falsch

$n^n \geq \Omega(100^n)$

- Wahr
- Falsch

$n^5 + 10n^4 \log(n) = \Theta(n^5 \log(n))$.

- Wahr
- Falsch

$2^{\log_{10}(n)} = \Theta(2^{\ln(n)})$.

- Wahr
- Falsch

Gegeben sei ein Algorithmus A und eine wachsende Funktion $T(n)$, die die Laufzeit für eine Eingabe der Grösse n berechnet. Nehmen Sie an, dass $T(n) \leq T(n/2) + C$ für eine Konstante $C > 0$ gilt, und dass $T(1) = 1$. Dann gilt $T(n) \leq O(\log n)$.

- Wahr
- Falsch

Gegeben sei ein Algorithmus A und eine wachsende Funktion $T(n)$, die die Laufzeit für eine Eingabe der Grösse n berechnet. Nehmen Sie an, dass $T(n) \geq 2 \cdot T(n/2) + dn$ für eine Konstante $d > 0$ gilt, und dass $T(1) = 10$. Dann gilt $T(n) \geq \Omega(n^{1.5})$.

- Wahr
- Falsch

Lineare Suche hat auf sortierten Arrays eine Worst-case Laufzeit von $\Theta(n)$.

- Wahr
- Falsch

Alice hat einen neuen vergleichsbasierten Suchalgorithmus entwickelt, der als Eingabe ein sortiertes Array von n ganzen Zahlen und einen Zielwert x erhält und die Suchspanne iterativ quadriert. Der Algorithmus ist korrekt. Alice behauptet, dass der Algorithmus eine Worst-case Laufzeit von $O(\log(\log(n)))$ hat.

Wählen Sie eine Antwort:

a. Alice liegt falsch, ihr Algorithmus ist langsamer als $O(\log(\log(n)))$.

b. Ja, iteratives Quadrieren führt zu einer Laufzeit von $O(\log(\log(n)))$.

c. Ohne die Details des Algorithmus zu kennen können wir keine Aussage über die Worst-case Laufzeit treffen.

Welcher Sortieralgorithmus aus der Vorlesung ist im folgenden Pseudocode implementiert?

for $j = 1, 2, \ldots, n-1$ **do:**

    $max = 1$

    for $k = 1, 2, \ldots, n-j+1$ **do**:

        if $A[k] > A[max]$ **then**:

            $max = k$

    if $max < n-j+1$ **then**:

    Tausche $A[max]$ und $A[n-j+1]$

- a. Selection Sort
- b. Insertion Sort
- c. Merge Sort
- d. Bubble Sort

Wir betrachten den folgenden Pseudocode-Ausschnitt:

$i \leftarrow 1$

while $i \leq n$:

    $j \leftarrow 1$

    while $j \leq i$:

        $f()$

        $j \leftarrow j + 1$

    $i \leftarrow i + 1$

Welcher der folgenden Ausdrücke beschreibt die genaue Anzahl der Aufrufe von f korrekt?

- a. $\sum_{i=1}^{n} \left(\sum_{j=1}^{i} 1 \right)$
- b. $\sum_{i=1}^{n} i \cdot \left(\sum_{j=1}^{n} 1 \right)$
- c. $\sum_{i=1}^{n} \left(1 + \sum_{j=1}^{i} 1 \right)$
- d. $\sum_{i=1}^{n} \left(\sum_{j=1}^{n} 1 \right)$

Theory Recap

Inhalt

- Invarianten
- Suchalgorithmen
- Sortieralgorithmen

Invarianten

- halten nach jeder loop-Iteration (besonders auf 1. und letzte achten)
- werden meist induktiv bewiesen

Linear Search

- Durch alle Elemente gehen, bis gefunden oder beim letzten angekommen
- Elemente müssen nicht sortiert sein
- $O(n)$
- implementiert durch 1 for-loop

```java
public static int linearSearch(int[] arr, int x) {
    for(int i = 0; i < arr.length; i++) {
        if(arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

Binary Search

- in der Mitte Starten und dann immer auf der richtigen Seite weitersuchen (auch jeweils Mitte)
- Elemente müssen sortiert sein
- $O(\log(n))$
- rekursiv implementiert

```java
public static int binarySearch(int arr[], int l, int r, int x) {
    if (r >= l) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);
        return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}
```

Bubble Sort

- Immer grösstes Element zum Ende bringen durch Vertauschungen mit Nachbarn
- in place
- $O(n^2)$
- Implementiert durch 2 for-Loops

```java
public static int[] bubbleSort(int[] arr) {
    for(int i = 0; i < arr.length-1; i++) {
        for(int j = 0; j < arr.length-1; j++) {
            if(arr[j] > arr[j+1]) {
                int tmp = arr[j+1];
                arr[j+1] = arr[j];
                arr[j] = tmp;
            }
        }
    }
    return arr;
}
```

Selection Sort

- Finde grösstes Element und tausche mit letztem unsortiertem Element
- in place
- $O(n^2)$
- implementiert durch 2 for-Loops

```java
public static int[] selectionSort(int[] arr) {
    for(int i = 0; i < arr.length; i++) {
        int max = Integer.MIN_VALUE;
        for (int j = 0; j < arr.length-i; j++) {
            if(arr[j] > max) {
                max = j;
            }
        }
        int tmp = arr[arr.length-i-1];
        arr[arr.length-i-1] = arr[max];
        arr[max] = tmp;
    }
    return arr;
}
```

Insertion Sort

- Die ersten Elemente sind schon sortiert, finde Stelle wo nächstes Element hinkommt
- in place
- $O(n^2)$
- implementiert durch 2 for-Loops

```java
public static int[] insertionSort(int[] arr) {
    int tmp1;
    int tmp2;
    for(int i = 0; i < arr.length-1; i++) {
        int index = BinarySearch.searchRecursive(arr, arr[i+1], 0, i);
        tmp1 = arr[index];
        arr[index] = arr[i];
        for(int j = index+1; j < i+1; j++) {
            tmp2 = arr[j];
            arr[j] = tmp1;
            tmp1 = tmp2;
        }
    }
    return arr;
}
```

Merge Sort

- Mache kleinere Sub-Arrays und sortiere diese, danach bringe sie zurück zusammen
- nicht in place
- $O(n \cdot \log(n))$
- rekursiv implementiert

```java
public static int[] mergeSort(int[] arr, int l, int r) {
    if(l < r) {
        int mid = (l+r)/2;
        mergeSort(arr, l, mid);
        mergeSort(arr, mid+1,r);
        merge(arr,l,mid,r);
    }
    return arr;
}
```

```java
public static int[] merge(int[] arr, int l, int mid, int r) {
    int[] B = new int[r-l+1];
    int i = l;
    int j = mid + 1;
    int k = 0;
    while( i <= mid && j <= r) {
        if(arr[i] <= arr[j]) {
            B[k] = arr[i];
            i++;
        }else {
            B[k] = arr[j];
            j++;
        }
        k++;
    }
    while(i<=mid) {
        B[k] = arr[i];
        i++;
        k++;
    }
    while(j<=r) {
        B[k] = arr[j];
        j++;
        k++;
    }
    return B;
}
```

Exercise 5.2 *Sorting algorithms.*

Below you see four sequences of snapshots, each obtained in consecutive steps of the execution of one of the following algorithms: `InsertionSort`, `SelectionSort`, `QuickSort`, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

3	6	5	1	2	4	8	7
3	6	5	1	2	4	8	7
3	5	6	1	2	4	8	7

3	6	5	1	2	4	8	7
3	5	1	2	4	6	7	8
3	1	2	4	5	6	7	8

3	6	5	1	2	4	8	7
3	6	1	5	2	4	7	8
1	3	5	6	2	4	7	8

3	6	5	1	2	4	8	7
1	6	5	3	2	4	8	7
1	2	5	3	6	4	8	7

Prüfungsaufgaben

g) *Computing Powers:*

Provide an algorithm that takes two positive integers a and n as input, and outputs a^n.

You can use addition, subtraction, multiplication and division of integers as basic operations without further explanation. You do not need to proof correctness or analyse the running time of your algorithm. However, for full points the number of basic operations that your algorithm uses needs to be $\mathcal{O}(\log n)$.

c) *Bubblesort*: Bubblesort is implemented as two nested loops. Draw how the following array looks like after the first iteration of bubblesort's outer loop.

19	18	20	9	7	33	1	2	6	5

Exercise Recommendations

Aufgabenblatt 4

- 4.1 mindestens 1 davon lösen, kann bei Prüfung kommen. Wenn keine Zeit -> merken für Prüfungsvorbereitung
- 4.2 Machen, geht nicht lange und wichtig für Prüfung
- 4.3 Bonus
- 4.4 Bonus
- 4.5 Bonus

Leetcode of the Week

- 88. Merge Sorted Array
- leetcode.com/problems/merge-sorted-array

Peergrading

- Aufgabe 3.1 a
- Grading Scheme in Lösungen beachten