ALP III: Datenstrukturen und Datenabstraktion
2. Aufgabenblatt
Übungsgruppe 1.8: Marcel Erhardt

Tobias Lohse/ Marvin Kleinert/ Anton Drewing

31.10.2014

Aufgabe 1

a)

Algorithmus zum Bestimmen des größten und zweitgrößten Elementes:

- Vergleiche das erste und zweite Element; speichere den Wert des kleineren in der Variablen max2Elem und den Wert des größeren in maxElem
- Durchlaufe die Folge ab dem dritten Element und vergleiche dabei jedes Element mit max2Elem und maxElem:
  - wenn das Element größer als max2Elem und kleiner als maxElem ist, speichere seinen Wert in max2Elem
  - wenn das Element größer als max2Elem und maxElem ist, speichere den Wert von maxElem in max2Elem und den Wert des neuen Elementes in maxElem
- Gib die Werte von max2Elem und maxElem zurück

Anzahl der Vergleiche:

$$C(n) = \underbrace{1}_{elem_1 \leftrightarrow elem_2} + \underbrace{2 * (n-2)}_{\forall elem_i, i \geq 3: elem_i \leftrightarrow max2Elem, elem_i \leftrightarrow maxElem} = 2n - 3$$

b)

Idee des Algorithmus:

- Teile rekursiv Folgen der Länge größer zwei in zwei Hälften
- Vergleiche die Elemente der Zweierfolgen und sortiere diese aufsteigend
- Vergleiche rekursiv jeweils die letzten zwei Elemente einer Folge mit denen der Nachbarfolge und sortiere die Elemente aufsteigend zu einer neuen Teilfolge
- Gib nach der letzten Sortierung die letzten beiden Elemnete aus

Rekursionsgleichung:

$$C(1) = 0 \qquad C(2) = 1$$

$$C(n) = 2C(n/2) + 4$$

Lösen der Rekursionsgleichung:

1. Einsetzen

$$\begin{aligned} C(n) &= 2C(n/2) + 4 \\ &= 2 * 2 * C(n/4) + 8 + 4 \\ &= 2 * 2 * 2 * C(n/8) + 16 + 8 + 4 \\ &= \dots \end{aligned}$$

2. Formel

$$2^k * C(n/2^k) + \sum_{i=1}^{k} 2 * 2^i$$

3. k für Anker = 2 bestimmen

$$k = logn - 1$$

4. Anker und k einsetzen

$$\begin{aligned} C(n) &= 2^{logn-1} * C(n/2^{logn-1}) + \sum_{i=1}^{logn-1} 2 * 2^i \\ &= 1/2n * 1 - 2n - 2 + 2 * \sum_{i=0}^{logn} 2^i \\ &= 1/2n - 2n - 2 + 2 * \frac{1-2^{logn+1}}{1-2} \\ &= 1/2n - 2n - 2 - 2 + 2^{logn+2} \\ &= 5/2n - 4 \end{aligned}$$

Anzahl der Vergleiche:

$$C(n) = 5/2n - 4$$

Aufgabe 2

Code im Anhang.

Messwerte

n	Vergleiche	Laufzeit (in ns)
10000	34056	141660
20000	73733	229720
30000	119564	354050
50000	228084	605160
80000	448224	1053560
99000	599780	1469660

Nach Regression mithilfe der R - Funktion lm() ergeben sich folgende Formeln:

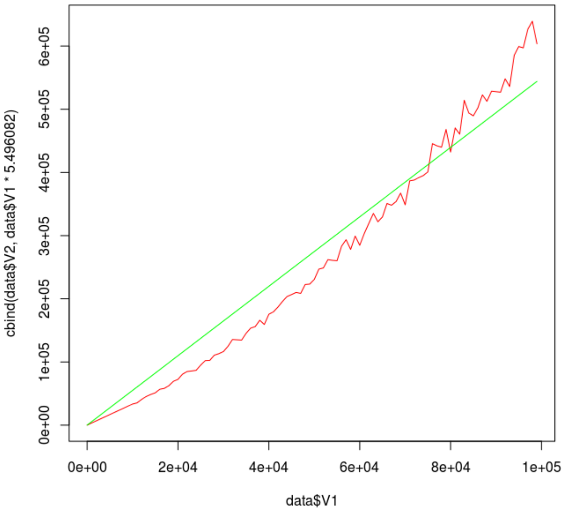
$$C(n) = 5.49Vgl * n$$

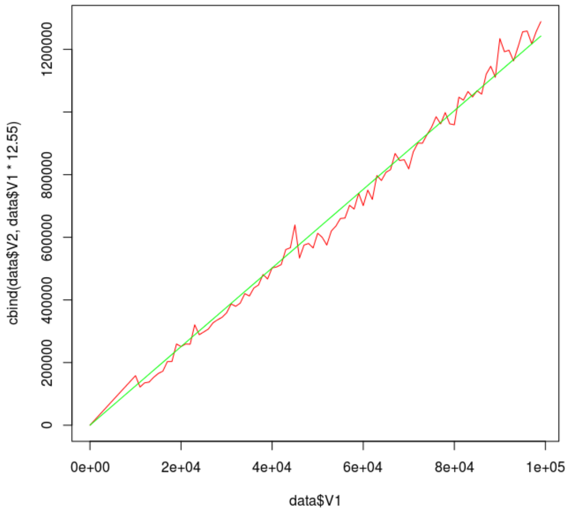
und

$$T(n) = 12.55ns * n$$

Plots

x-Achse: Größe der Liste, y-Achse: Anzahl Vergleiche/Zeit in ns
Rot: Echte Werte, Grün: Regressionsgraph





Aufgabe 3

a)

Variante des Quicksort mit O(nlogn): A sei deterministischer Algorithmus, der in O(n) den Median einer Eingabefolge S der Länge n bestimmt.

- falls n == 1, gib n zurück, sonst
- pivot = A(S);
- durchlaufe Folge, bilde Teilfolgen S_1 aller Elemente < pivot, und S_2 aller Elemente > pivot
- sortiere S_1 und S_2 rekursiv
- gib aus S_1 sortiert, pivot, S_2 sortiert

Anzahl der Vergleiche:

$$C(0) = 0 \qquad C(1) = 0$$

$$C(n) = C(\lceil n/2 \rceil - 1) + C(\lfloor n/2 \rfloor) + (n-1) + O(n)$$

Es gilt offensichtlich:

$$C(\lceil n/2 \rceil - 1) + C(\lfloor n/2 \rfloor) + (n-1) + O(n) \leq \underbrace{2 * C(n/2) + n}_{nlog(n)} + O(n) \leq O(nlog(n))$$

b)

Deterministischer Linearzeit-Algorithmus für das Auswahlproblem:

- if n==1 then return Element von S
- pivot = A(S)
- spalte S auf in $S_<$, $S_=$, $S_>$

- if $|S_<| \geq k$ then return quickSelect($S_<$,k)
- if $k \leq |S_<| + |S_=|$ then return pivot;
- quickSelect($S_>, k - |S_<| - |S_=|$)

Laufzeit:

$$T(n) = (n-1) + \begin{cases} T(n/2) & \text{für } k \neq n/2 \\ 0 & \text{für } k = n/2 \end{cases}$$

Daraus folgt:

$$T(n) \leq n + T(n/2) + O(n) \leq 2n - 1 + O(n)$$

```java
import java.util.Random;

public class Quickselect {

  public static int cmpCount = 0;

  public static void run(int size) {
    Random rn = new Random();
    int[] list = new int[size];
    for (int i = 0; i < size; i++) {
      list[i] = rn.nextInt(100);
    }
    int index = rn.nextInt(size);
    int val = quickselect(list, 0, size-1, index, rn);
  }

  public static long measureTime(int size) {
    Random rn = new Random();
    int[] list = new int[size];
    for (int i = 0; i < size; i++) {
      list[i] = rn.nextInt(100);
    }
    int index = rn.nextInt(size);
    long time = System.nanoTime();
    int val = quickselect(list, 0, size-1, index, rn);
    time = (System.nanoTime()-time);
    return time;
  }

  public static int quickselect(int[] list, int l, int r, int index, Random rn) {
    if (l == r) return list[l];
    int pivot = rn.nextInt(r-l+1) + l;
    pivot = partition(list, l, r, pivot);
    cmpCount++;
    if (index == pivot) return list[index];
    else if (index < pivot) {
      cmpCount++;
      return quickselect(list, l, pivot-1, index, rn);
    }
    else
      return quickselect(list, pivot+1, r, index, rn);
  }

  public static int partition(int[] list, int l, int r, int pivot) {
    int pVal = list[pivot];
    swap(list, pivot, r);
    int tmpIndex = l;
    for (int i = l; i < r; i++) {
      cmpCount++;
```

```java
      if(list[i] < pVal) {
        swap(list, tmpIndex, i);
        tmpIndex++;
      }
    }
    swap(list, r, tmpIndex);
    return tmpIndex;
  }

  static void swap(int[] list, int i, int j) {
    int tmp = list[i];
    list[i] = list[j];
    list[j] = tmp;
  }
}


public class Test {
  public static void main(String args[]) {
    float[] vals = new float[90];
    double[] time = new double[90];
    for (int i = 10; i < 100; i++) {
      vals[i-10] = 0.0f;
      time[i-10] = 0.0f;
      for (int j = 0; j < 100; j++) {
        Quickselect.cmpCount = 0;
        System.out.println("COUNT: "+i);
        Quickselect.run(i*1000);
        vals[i-10] += Quickselect.cmpCount;
        time[i-10] += Quickselect.measureTime(i*1000);
      }
      vals[i-10] /= 100.0;
      time[i-10] /= 100.0;
    }
    for (int i = 0; i < 90; i++) {
      System.out.println(((i+10)*1000)+", "+vals[i]);
    }
    System.out.println("_______________________");
    for (int i = 0; i < 90; i++) {
      System.out.println(((i+10)*1000)+", "+time[i]);
    }
  }
}
```