Mathematik für Informatiker
Kombinatorik, Stochastik und Statistik

Übungsblatt 7

Tom Paßberg , Iain Dorsch

Aufgabe 1

a)

Wir erreichen $\binom{n}{2}$ Vergleiche wenn wir als Pivot Element immer das Kleinste Element der Eingabe Wählen.
Wir Testen diese Ausage mit einem Rust Programm, die Vergleiche, die zur Ermittlung des Minimums benötigt werden, werden nicht mitgezählt. Die gemessene Laufzeit kjann in einem Randomisierten Quicksort mit sehr ungünstiger Pivot-Wahl auftreten.
Algorithmus in Rust:

```
fn main() {
    let input = vec![56,64,58,61,75,86,17,62,8,50,87,99,67,10,74];

    let (_, num_comparisons) = bad_quicksort(input);

    println!("Number of comparisons: {}", num_comparisons);
}

fn bad_quicksort(mut vec: Vec<i32>) -> (Vec<i32>, u64) {
    if vec.len() <= 1 {
        return (vec, 0);
    }
    let num_comparisons = vec.len() as u64 - 1;
    let pivot = *vec.iter().min().unwrap();
    vec.retain(|&x| x != pivot);
    let (left, right) = vec.into_iter().partition(|&x| x < pivot);

    let (mut result, count) = bad_quicksort(left);
    let (right_result, right_count) = bad_quicksort(right);
    result.push(pivot);
    result.extend(right_result);
    (result, count + right_count + num_comparisons)
}
```

Ausgabe:

```
Number of comparisons: 105
```

Die Wahl der Pivotelemnte erfüllt die Anforderung $105 = \binom{15}{2}$

b)

Wenn wir als Pivot Element immer das Median der Eingabe Wählen erhalten wir die best case Laufzeit. Wir zählen die Vergleiche zur Ermittlung des Medians nicht mit, die Laufzeit ist also die best case Laufzeit eines Randomisierten Quicksort Algorithmus.
Wir testen die Aussage mit einem Rust Programm.

```
fn good_quicksort(mut vec: Vec<i32>) -> (Vec<i32>, u64) {
    if vec.len() <= 1 {
        return (vec, 0);
    }
    let num_comparisons = vec.len() as u64 - 1;
```

```
6       let pivot = median(&vec);
7       vec.retain(|&x| x != pivot);
8       let (left, right) = vec.into_iter().partition(|&x| x < pivot);
9
10      let (mut result, count) = good_quicksort(left);
11      let (right_result, right_count) = good_quicksort(right);
12      result.push(pivot);
13      result.extend(right_result);
14      (result, count + right_count + num_comparisons)
15  }
```

Ausgabe:

```
1  Number of comparisons: 34
```

Die Wahl des Medians als Pivot Element erfüllt die Anforderung $34 < 15 \cdot \ln(15)$

Aufgabe 2

Als Laufzeit betrachten wir die Anzahl der Multiplikationen.

a)

$$A \cdot x \tag{1}$$

$$= \left(\sum_{j=1}^{n} a_{ij} x_j \right)_{i=1..n} \tag{2}$$

Die Summenformel wird n mal berechnet. Wir substituieren $a_{ij}x_j$ durch 1, damit zählen wir die Anzahl der multiplikations Operationen

$$n \cdot \left(\sum_{j=1}^{n} 1 \right) \tag{3}$$

$$= n \cdot n \tag{4}$$

$$= n^2 \tag{5}$$

b)

$$A \cdot B \tag{6}$$

$$= \left(\sum_{j=1}^{n} a_{ij} x_{jk} \right)_{i,k=1..n} \tag{7}$$

Die Summenformel wird $n \cdot n$ mal berechnet (für $i, k \in 1..n$). Wir substituieren $a_{ij}x_{jk}$ durch 1 (zähle die Anzahl der multiplikations Operationen):

$$n \cdot \left(n \cdot \left(\sum_{j=1}^{n} 1 \right) \right) \tag{8}$$

$$= n \cdot n \cdot n \tag{9}$$

$$= n^3 \tag{10}$$

$$\tag{11}$$

c) Sei $C = A \cdot B$, dann gilt

$$C \cdot x = A \cdot (B \cdot x)$$

Wir testen für zufällige x ob die Gleichung gilt. Einen vektor mit einer Matirx zu multiplizieren ist eine Operation der Laufzeit $\mathcal{O}(n^2)$. Der nachfolgende Rust Code testet die Gleichung für zufällige x in $O(n^2)$:

```
1  fn monte_carlo_test(A: &Matrix, B: &Matrix, C: &Matrix) -> bool {
2      for i in 0..10 {
3          let x = (0..C.cols).map(|_| random::<f32>() as f64).collect::<
               Vec<f64>>();
4          let P = C.vec_multiply(&x);
5          let Q = A.vec_multiply(&B.vec_multiply(&x));
6          if P != Q {
7              return false;
8          }
9      }
10     true
11 }
```

Aufgabe 3

a)

Algorithmus in Rust:

```
fn quicksort(vec: Vec<i32>) -> (Vec<i32>, u64) {
    if vec.len() <= 1 {
        return (vec, 0);
    }
    let num_comparisons = vec.len() as u64 - 1;
    let mut iter = vec.into_iter();
    let pivot = iter.next().unwrap();
    let (left, right) = iter.partition(|&x| x < pivot);

    let (mut result, count) = quicksort(left);
    let (right_result, right_count) = quicksort(right);
    result.push(pivot);
    result.extend(right_result);
    (result, count + right_count + num_comparisons)
}
```

Als pivot-Element wird immer das erste Element der Eingabe gewählt. Die Funktion gibt ein Tupel zurück, bestehend aus dem sortierten Vektor und der Anzahl der Vergleiche. Wir zählen nur die Vergleiche zwischen Listenelementen, nicht den Vergleich der Länge der Liste mit der Zahl 1 (Für die assymptitische Laufzeit macht das keinen Unterschied).
Funktionsaufruf:

```
fn main() {
    let input = vec![3, 2, 1, 7, 6, 4, 5, 8, 10, 9];
    let (sorted, count) = quicksort(input);
    println!("Sorted: {:?}\nComparisions: {}", sorted, count);
}
```

Ausgabe:

```
Sorted: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Comparisions: 20
```

b)

Daten erzeugen:

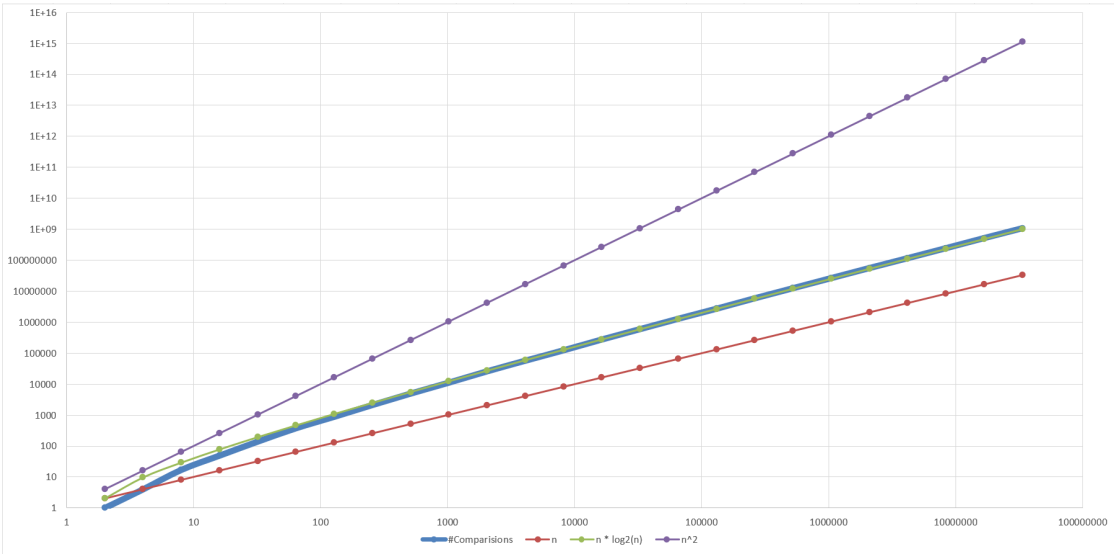
```
fn main() {
    let mut workbook = Workbook::new();
    let worksheet = workbook.add_worksheet();

    worksheet.write(0, 0, "#Elements").unwrap();
    worksheet.write(1, 0, "#Comparisions").unwrap();

    let sample_size = 20;
    for i in 1..=25 {
        let count_sum = (0..sample_size).into_par_iter().map(|_| {
            let mut vec = Vec::with_capacity(2usize.pow(i));
```

```
12                  for _ in 0..2usize.pow(i) {
13                      vec.push(rand::random::<i32>());
14                  }
15                  let mut truth = vec.clone();
16                  truth.sort_unstable();
17
18                  let (sorted, count) = quicksort(vec);
19                  // teste ob das Ergebnis korrekt ist
20                  assert_eq!(sorted, truth);
21                  count
22              }).sum::<u64>();
23
24              // berechne den Durchschnitt der Vergleiche
25              let count = (count_sum as f32 / sample_size as f32) as i32;
26
27              println!("Elements: {:10}, Comparisions: {:13}", 2usize.pow(i),
                    count);
28
29              // Schreibe Ergebiss in Excel Datei
30              worksheet.write(0, i as u16, 2usize.pow(i) as i32).unwrap();
31              worksheet.write(1, i as u16, count).unwrap();
32          }
33          workbook.save("quicksort.xlsx").unwrap();
34      }
```

Grafische Darstellung der Ausgabe:



Die Laufzeit verhält sich wie erwartet mit $\mathcal{O}(n \log n)$.