Sortieralgorithmen

Inhalt

0 Sortieralgorithmen

0.1 Zum Aufbau des Referats

Dieses Referat bietet einen Einstieg in das Seminarthema „Algorithmen und Zufälligkeit“. Algorithmen sollen so vorgestellt werden, dass eine Verwendung auch in der Schule möglich ist. Dies ist auch deshalb notwendig, weil „Algorithmen“ als eine „fundamental idea“ des schulischen Mathematikunterrichts genannt werden.[1] Das Sortieren ist eines der wichtigsten Anwendungsbereiche von Algorithmen. Beispiele für sortierte Listen gibt viele: Statistiken, Telefonbücher, Wörterbücher, Listen u.a.

In diesem Referat sollen vier verschiedene Algorithmen zum Sortieren von Datenmengen analysiert werden. Dabei wird jeder einzelne Algorithmus vorgestellt und in einem Sortierbeispiel dokumentiert. Es schließen sich Betrachtungen zur Laufzeit (insbesondere zur Zahl der durchgeführten Vergleiche und der Austausche) an. Bemerkungen zu Vor- und Nachteilen und besonderen Merkmalen der einzelnen Algorithmen beschließen die Betrachtungen. Die Kurzen Bemerkungen zu Maple beziehen sich auf das Maple-Worksheet „sort.mws“.

Wie analysiert man Algorithmen? Diese Frage kann an dieser Stelle nicht ausführlich beantwortet werden.[2] Für die Laufzeit der hier behandelten Sortieralgorithmen sind u.a.

- die Anzahl der durchgeführten Vergleiche zwischen zwei Elementen,
- die Anzahl der auszutauschenden Elemente (Bewegungen, Vertauschungen...),
- die Zahl der Zuweisungen, die innerhalb eines Computerprogrammes notwendig sind,

relevant. Die beiden ersten Punkte lassen sich unabhängig von der Leistungsfähigkeit eines Programms oder eines Rechners diskutieren und werden deshalb für die hier behandelten Sortieralgorithmen angegeben. In der Regel besitzen Algorithmen einen Hauptparameter N (gewöhnlich die Anzahl der zu bearbeitenden Elemente), mit dessen Hilfe die Leistungsfähigkeit beschrieben wird. Häufig sind die Laufzeiten der Algorithmen proportional zu den folgenden Funktionen:

- $f(N) = 1$ (konstante Laufzeit)
- $f(N) = \log N$ (logarithmische Laufzeit)
- $f(N) = N^a$ mit $a = 1,2,3...$ (lineare, quadratische, kubische Laufzeit)

[1] Vgl. z.B. J. Humberger u.a., Fundamentale Ideen der angewandten Mathematik, Mannheim u.a. 1995

Zur Beschreibung der Laufzeit benutzt man häufig die „O-Schreibweise“, die lediglich eine Abschätzung liefert.

Eine Funktion $g(N)$ *wird* $O(f(N))$ *genannt, falls es Konstanten* c_0 *und* N_0 *gibt, so dass, für alle* $N > N_0$, $g(N)$ *kleiner als* $c_0 \cdot f(N)$ *ist.*

0.2 Erläuterung wichtiger Begriffe

- *Sortieren*

  „Angenommen, wir haben eine Liste n von verschiedenen Zahlen $a_1,...,a_n$ gegeben, und unsere Aufgabe besteht darin, sie in die richtige Reihenfolge, zum Beispiel in aufsteigender Folge, zu bringen. Das heißt wir müssen die eindeutige Permutation π bestimmen, so dass $a_{p(1)} < a_{p(2)} < a_{p(3)} < ... < a_{p(n)}$ gilt. Statt Zahlen können wir irgendeine Liste von Elementen betrachten, auf denen eine lineare Ordnung gegeben ist. Als Tests stehen uns paarweise Vergleiche $a_i : a_j$ zur Verfügung, mit den Antworten $a_i < a_j$ oder $a_i > a_j$.“[3]

- *Vergleiche und Austausche*

  Ergibt sich aus den Vergleichen $a_i : a_j$ ein „unerwünschtes“ Ergebnis, d.h. $a_i > a_j$ für $i < j$ bei Sortierziel „aufsteigende Reihenfolge“, so erfolgt bei den Sortieralgorithmen ein Austausch von zwei Elemente. Dieser Austausch ist bei den einzelnen Verfahren unterschiedlich und muss im Einzelfall betrachtet werden.

- *Untersuchte Fälle*

  Bei den einzelnen Sortieralgorithmen werden drei verschiedene Fälle betrachtet

  - Bester Fall („best case“) - i.d.R. bei bereits sortierter Liste
  - Schlechtester Fall („worst case“) - der Fall, der im jeweiligen Algorithmus zu den meisten Rechenschritten führt
  - Durchschnittlicher Fall („avaredge case“) - Erwartungswert bei „zufälliger Liste“

0.3 Literatur

- Aigner, Martin, Diskrete Mathematik, 2. Auflage, Wiesbaden (vieweg) 1996
- Sedgewick, Robert, Algorithmen in C++, 4. Auflage, Bonn u.a. (Addison-Wesley) 1998
- Geiger, Jochen, Algorithmen und Zufälligkeit, Vorlesungs-Skript, WS 1999/2000

[2] Ein Überblick gibt: Sedgewick, Algorithmen, 93ff.
[3] Aigner, Diskrete Mathematik, 162

1 (Straight-) Selection-Sort

1.1 Der Algorithmus

- Finde das kleinste Element der gesamten Liste. Tausche es mit dem an der ersten Position.
- Wähle nun („selection“) das zweitkleinste (= das kleinste in der verbliebenen Liste) und tausche es gegen das zweite aus.
- Verfahre so, bis die gesamte Liste sortiert ist (Zeiger auf Stelle n).

1.2 Maple

```
for i from 1 to (n-1) do                    Suche des kleinsten Wertes
m:=i;
        for j from i+1 to n do
        if X[j] < X[m] then m:=j; fi;
        od;
a:=X[m];                                    Tausch des Elements mit kleinstem Wert
X[m]:=X[i];                                 und Wert des Index i
X[i]:=a;
Od;
```

1.3 Laufzeitbetrachtungen

Sei bei allen folgenden Betrachtungen

V:= Anzahl der durchgeführten Vergleiche

A:= Anzahl der durchgeführten Austausche

1.3.1 Betrachtung des besten Falles (alle Elemente in aufsteigender Folge)

- $V = (n-1) + (n-2) + ... + 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$:
- $A = 0$

1.3.2 Betrachtung des schlechtesten Falls (kein Element ist auf seinem Platz)

- $V = (n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$
- $A = n - 1$

1.3.3 Betrachtung des durchschnittlichen Falls (Erwartungswert)

- $E(V) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$
- $E(A) = (1-\frac{1}{n})\cdot 1 + (1-\frac{1}{n-1})\cdot 1 + (1-\frac{1}{n-2})\cdot 1 + \ldots + (1-\frac{1}{2})\cdot 1$

  $= (n-1) - \sum_{k=2}^{n} \frac{1}{k} + 1$

  $= (n-1) - H_n + 1$

  $= n - H_n \approx n$

1.4 Bemerkungen

- Selection-Sort ist ein sehr geeignetes Verfahren für große Datensätze mit kleinen Schlüsseln (Sortierkriterium). Grund dafür ist, dass auf einen einzigen Vergleich aufgebaut wird (Suchen des jeweils kleinsten verbliebenen Elements), der bei kurzen Vergleichselementen schnell durchgeführt werden kann.
- Selection-Sort führt in allen möglichen Fällen ungefähr gleich viele gleich viele Vergleiche durch, ist also unabhängig von der Struktur der Eingabedaten.
- Wie bei der Simulation mit Maple sieht, ist Selection-Sort sehr ähnlich zu Bubble-Sort (s.Kapitel 3), arbeitet jedoch durch die direkte Auswahl des kleinsten Elementes deutlich schneller als dieser Sortieralgorithmus.

1.5 Beispiel für ein Sortiervorgang

	A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
1	A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
2		S	O	R	T	I	N	G	E	X	A	M	P	L	E
3			O	R	T	I	N	G	E	X	S	M	P	L	E
4				R	T	I	N	G	E	X	S	M	P	L	E
5					T	I	N	G	O	X	S	M	P	L	R
6						I	N	T	O	X	S	M	P	L	R
7							N	T	O	X	S	M	P	L	R
8								T	O	X	S	M	P	L	R
9									O	X	S	T	P	N	R
10										X	S	T	P	O	R
11											S	T	P	X	R
12												T	S	X	R
13													S	X	T
14														X	T
15	A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

2 (Straight-) Insertion-Sort

2.1 Der Algorithmus

- Durchlaufe das Feld bis ein Element an der „falschen“ Stelle steht (d.h. $a_{i+1} < a_i$).
- Setze dieses Element an die „richtige“ Stelle, indem die größeren Elemente nach rechts verschoben werden, so dass es an die frei werdende Stelle eingefügt werden kann.
- Fahre so fort, bis alles sortiert ist.

2.2 Maple

```
for i from 2 to n do                    Durchlaufen der Liste, Auswahl des
a:=X[i];                                einzufügenden Elements
j:=i;
        while (X[j-1]>a and j>1) do     Teilen der Liste als Vorbereitung zum
        X[j]:=X[j-1];                   Einfügen
        j:=j-1;
        od;
X[j]:=a;                                Einfügen des Elements an die „richtige“
od;                                     Stelle „j“
```

2.3 Laufzeitbetrachtungen

2.3.1 Betrachtung des besten Falles (alle Elemente in aufsteigender Folge)

- $V = n-1$
- $A = 0$

2.3.2 Betrachtung des schlechtesten Falles (alle Elemente in absteigender Folge)

- $V = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2} \approx \frac{n^2}{2}$
- $A = \sum_{i=2}^{n} \frac{i}{2} = \frac{1}{2}\sum_{i=2}^{n} i = \frac{1}{2}\sum_{i=1}^{n} i - \frac{1}{2} = \frac{1}{2}\frac{(n-1)n}{2} - \frac{1}{2} \approx \frac{n^2}{4}$

2.3.3 Betrachtung des durchschnittlichen Falls (Erwartungswert)

- $E(V) = \sum_{i=1}^{n-1} 1 + \frac{i-1}{2} = n - 1 + \frac{(n-2)(n-1)}{4} = \frac{n^2+3n-4}{4} \approx \frac{n^2}{4}$
- $E(A) = E(V) \cdot \frac{1}{2} \approx \frac{n^2}{8}$

2.3.4 Beweis zum Erwartungswert

Zur Berechnung von E(# Vergleiche in Runde k) berechen wir

$$E \sum_{i=1}^{k-1} 1\{X_i \geq X_k\} + 1,$$

da es für jedes $X_i \geq X_k$ einen Vergleich gibt und einen weiteren für das erste Element, das kleiner ist als X_k. Somit muss berechnet werden

$$1 + \sum_{i=1}^{k-1} P(X_i \geq X_k).$$

Zur Berechnung benutzen wir den Satz von der totalen Wahrscheinlichkeit:

$$P(A) = \sum_{i=1}^{n} P(A \mid B_i) \cdot P(B_i).$$

Das liefert uns Wahrscheinlichkeiten, die wir angeben können:

$$1 + \sum_{i=1}^{k-1} \sum_{j=1}^{n} P(X_i \geq X_k \mid X_k = j) \cdot P(X_k = j) = 1 + \sum_{i=1}^{k-1} \sum_{j=1}^{n} \frac{j-1}{n-1} \cdot \frac{1}{n}$$

Dabei ist $P(X_k = j) = \frac{1}{n}$, da aus n Elementen eines herausgegriffen wird. Weiterhin ergibt sich für $P(X_i \geq X_k \mid X_k = j)$ der Wert $\frac{j-1}{n-1}$, da aus den *(n-1)* Elementen - X_k ist ja gezogen - *(j-1)* betrachtet werden. Mit diesen Umformungen erhält man

$$1 + \sum_{i=1}^{k-1} \frac{1}{n} \cdot \left(\frac{0+1+2+3+\ldots+n-1}{n-1} \right)$$

$$= 1 + \sum_{i=1}^{k-1} \frac{1}{n} \cdot \left(\frac{n \cdot (n-1)}{2 \cdot (n-1)} \right) = 1 + \sum_{i=1}^{k-1} \frac{1}{2} = 1 + \frac{k-1}{2}.$$

Nun muss zur Berechnung des Erwartungswertes der Anzahl aller Vergleiche nur noch summiert werden und man erhält

$$\sum_{k=1}^{n-1} 1 + \frac{k-1}{2} = (n-1) \cdot \frac{(n-2)(n-1)}{4}$$

q.e.d.

2.4 Bemerkungen

- Insertion-Sort ist für solche Fälle geeignet, wo Daten an eine sortierte Datenmenge angefügt und danach sortiert werden sollen. „Für derartige Dateien ist Insertion-Sort in Bezug auf die Leistung sogar den komplizierten Verfahren überlegen [...].“[4]
- Denn für „fast sortierte Listen“ ist Insertion-Sort linear.[5] Entscheidend ist hierbei die Tatsache, dass zum Einfügen eines Element die links von ihm befindlichen gezählt werden, die größer sind. Von diesen Elementen gibt es aber in „fast sortierten Listen“ wenige.

2.5 Beispiel für den Sortiervorgang

	A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
1		S	O												
2		O	S	R											
3			R	S	T										
4					T	I									
5		I	O	R	S	T	N								
6			N	O	R	S	T	G							
7		G	I	N	O	R	S	T	E						
8		E	G	I	N	O	R	S	T	X					
9										X	A				
10		A	E	G	I	N	O	R	S	T	X	M			
11						M	N	O	R	S	T	X	P		
12									P	R	S	T	X	L	
13						L	M	N	O	P	R	S	T	X	E
14				E	G	I	L	M	N	O	P	R	S	T	X
15	A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

[4] Sedgewick, Sortieralgorithmen, 133
[5] vgl. Sedgewick, Sortieralgorithmen, 132f.

3 Bubble-Sort

3.1 Der Algorithmus

- Durchlaufe das Feld bis $a_{n+1} < a_n$.
- Vertausche die beiden Elemente.
- Iteriere bis alles sortiert ist.

3.2 Maple

```
for i from n to 1 by -1 do                  Durchlaufen der Liste
    for j from 2 to i do
        if X[j-1] > X[j] then               Größenvergleich benachbarter Elemente
        m:=X[j];
        X[j]:=X[j-1];                       evtl. Tauschen dieser Elemente
        X[j-1]:=m;
        fi;
    od;
od
```

3.3 Laufzeitbetrachtungen

3.3.1 Betrachtung des besten Falles (alle Elemente in aufsteigender Folge)

- $V = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$
- $A = 0$

3.3.2 Betrachtung des schlechtesten Falles (alle Elemente in absteigender Folge)

- $V = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \approx \frac{n^2}{2}$
- $A = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \approx \frac{n^2}{2}$

3.3.3 Betrachtung des durchschnittlichen Falls (Erwartungswert)

- $E(V) \approx \frac{n^2}{2}$ (ohne Beweis)

- $E(A) \approx \frac{n^2}{4}$ (ohne Beweis)

3.4 Bemerkungen

- Der Einsatz von Bubble-Sort lohnt sich nur für den Einsatz bei kleinen Listen, da der Programmieraufwand geringer ist. Für große Datenmengen ist Bubble-Sort unbrauchbar.
- Bubble-Sort ist dagegen bei „fast sortierten Listen“, wie auch Insertion-Sort, linear.

3.5 Beispiel für den Sortiervorgang

	A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
1	A	O	S	R	T	I	N	G	E	X	A	M	P	L	E
2	A	O	R	S	T	I	N	G	E	X	A	M	P	L	E
3	A	O	R	S	I	T	N	G	E	X	A	M	P	L	E
4	A	O	R	I	S	T	N	G	E	X	A	M	P	L	E
5	A	O	I	R	S	T	N	G	E	X	A	M	P	L	E
6	A	I	O	R	S	N	T	G	E	X	A	M	P	L	E
7	A	I	O	R	N	S	T	G	E	X	A	M	P	L	E
8	A	I	O	N	R	S	T	G	E	X	A	M	P	L	E
9	A	I	N	O	R	S	T	G	E	X	A	M	P	L	E
10	A	I	N	O	R	S	T	G	E	X	A	M	P	L	E
11	A	I	N	O	R	S	G	T	E	X	A	M	P	L	E
12	A	I	N	O	R	G	S	T	E	X	A	M	P	L	E
13	A	I	N	O	G	R	S	T	E	X	A	M	P	L	E
14	A	I	N	G	O	R	S	T	E	X	A	M	P	L	E
15	A	I	G	N	O	R	S	T	E	X	A	M	P	L	E
16	A	G	I	N	O	R	S	T	E	X	A	M	P	L	E
17	A	G	I	N	O	R	S	E	T	X	A	M	P	L	E
18	A	G	I	N	O	R	E	S	T	X	A	M	P	L	E
19	A	G	I	N	O	E	R	S	T	X	A	M	P	L	E
20	A	G	I	N	E	O	R	S	T	X	A	M	P	L	E
21	A	G	I	E	N	O	R	S	T	X	A	M	P	L	E
22	A	G	E	I	N	O	R	S	T	X	A	M	P	L	E
23	A	E	G	I	N	O	R	S	T	A	X	M	P	L	E
24	A	E	G	I	N	O	R	S	A	T	X	M	P	L	E
25	A	E	G	I	N	O	R	A	S	T	X	M	P	L	E

26	**A**	**E**	**G**	**I**	**N**	**O**	**A**	**R**	**S**	**T**	**X**	**M**	**P**	**L**	**E**
27	**A**	**E**	**G**	**I**	**N**	**A**	**O**	**R**	**S**	**T**	**X**	**M**	**P**	**L**	**E**
28	**A**	**E**	**G**	**I**	**A**	**N**	**O**	**R**	**S**	**T**	**X**	**M**	**P**	**L**	**E**
29	**A**	**E**	**G**	**A**	**I**	**N**	**O**	**R**	**S**	**T**	**X**	**M**	**P**	**L**	**E**
30	**A**	**E**	**A**	**G**	**I**	**N**	**O**	**R**	**S**	**T**	**X**	**M**	**P**	**L**	**E**
31	**A**	**A**	**E**	**G**	**I**	**N**	**O**	**R**	**S**	**T**	**X**	**M**	**P**	**L**	**E**
32	**A**	**A**	**E**	**G**	**I**	**N**	**O**	**R**	**S**	**T**	**M**	**X**	**P**	**L**	**E**
33	**A**	**A**	**E**	**G**	**I**	**N**	**O**	**R**	**S**	**M**	**T**	**X**	**P**	**L**	**E**
34	**A**	**A**	**E**	**G**	**I**	**N**	**O**	**R**	**M**	**S**	**T**	**X**	**P**	**L**	**E**
35	**A**	**A**	**E**	**G**	**I**	**N**	**O**	**M**	**R**	**S**	**T**	**X**	**P**	**L**	**E**
36	**A**	**A**	**E**	**G**	**I**	**N**	**M**	**O**	**R**	**S**	**T**	**X**	**P**	**L**	**E**
37	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**R**	**S**	**T**	**X**	**P**	**L**	**E**
38	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**R**	**S**	**T**	**P**	**X**	**L**	**E**
39	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**R**	**S**	**P**	**T**	**X**	**L**	**E**
40	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**R**	**P**	**S**	**T**	**X**	**L**	**E**
41	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**X**	**L**	**E**
42	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**L**	**X**	**E**
43	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**P**	**R**	**S**	**L**	**T**	**X**	**E**
44	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**P**	**R**	**L**	**S**	**T**	**X**	**E**
45	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**P**	**L**	**R**	**S**	**T**	**X**	**E**
46	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**O**	**L**	**P**	**R**	**S**	**T**	**X**	**E**
47	**A**	**A**	**E**	**G**	**I**	**M**	**N**	**L**	**O**	**P**	**R**	**S**	**T**	**X**	**E**
48	**A**	**A**	**E**	**G**	**I**	**M**	**L**	**N**	**O**	**P**	**R**	**S**	**T**	**X**	**E**
49	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**X**	**E**
50	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**E**	**X**
51	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**E**	**T**	**X**
52	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**R**	**E**	**S**	**T**	**X**
53	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**E**	**R**	**S**	**T**	**X**
54	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**E**	**P**	**R**	**S**	**T**	**X**
55	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**N**	**E**	**O**	**P**	**R**	**S**	**T**	**X**
56	**A**	**A**	**E**	**G**	**I**	**L**	**M**	**E**	**N**	**O**	**P**	**R**	**S**	**T**	**X**
57	**A**	**A**	**E**	**G**	**I**	**L**	**E**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**X**
58	**A**	**A**	**E**	**G**	**I**	**E**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**X**
59	**A**	**A**	**E**	**G**	**E**	**I**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**X**
60	**A**	**A**	**E**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**X**

4 Quick-Sort

4.1 Der Algorithmus

- Zerlege die Liste in eine untere und eine obere Hälfte H_1 und H_2, d.h. $a_i < a_j \forall i \in H_1, j \in H_2$. Dabei wird hier ein randomisierter Quicksort betrachtet, d.h. das Vergleichselement, mit dessen Hilfe die Liste geteilt wird, wird zufällig gewählt.
- Sortiere die Teillisten rekursiv.
- Füge die sortierten Listen zusammen.

4.2 Maple

quicksort:=proc(l,r)	Quick-Sort als procedure
zufall:=rand(l..r);	Finden eines „zufälligen“ Pivot-Elements
quicksort(l,i-1);quicksort(i+1,r);	Rekursion von quicksort

4.3 Laufzeitbetrachtungen

4.3.1 Betrachtung des besten Falles (Pivot-Element ist immer Median der Liste)

- $V \approx n \log_2 n$ (ohne Beweis)
- $A = 0$

4.3.2 Betrachtung des schlechtesten Falles (Pivot-E. ist immer das kleinste/größte)

- $V = \sum_{i=1}^{n-1} i \approx \frac{n^2}{2}$ (ohne Beweis)
- $A \leq \frac{n^2}{4}$ (ohne Beweis)

4.3.3 Betrachtung des durchschnittlichen Falls (Erwartungswert)

- $E(V) = 2 \cdot (n+1) H_n - 4n \approx 2n \ln n \approx 1{,}38 n \log_2 n$
- $E(A) \leq E(V) \cdot \frac{1}{2} \approx 2n \ln n \cdot \frac{1}{2}$

4.3.4 Beweis zum Erwartungswert

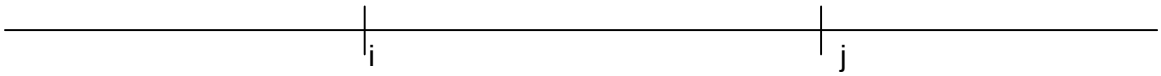
Sei *V* die Anzahl der von Quick-Sort durchgeführten Vergleiche. Sie liegt wie gesehen zwischen ca. $n \log_2 n$ (falls der stets der Median der zu sortierenden Mengen gewählt wird) und ca. $\frac{n^2}{2}$ (falls man als Vergleichselement stets das kleinste oder größte Element auswählt). Dieses ungünstige Verhalten ist im Mittel unwahrscheinlich, die mittlere Laufzeit ist gegeben durch

$$E(V) = 2(n+1)(1+\tfrac{1}{2}+\tfrac{1}{3}+\tfrac{1}{4}+...+\tfrac{1}{n}) - 4n .$$

Zum Beweis schreiben wir *V* als Summe von Indikatorvariablen und erhalten

$$V = \sum_{1 \le i < j \le n} 1(A_{ij})$$

, wobei A_{ij} das Ereignis sei, dass es zum Vergleich kommt zwischen den Elementen l_i und l_j der Liste L. Es gilt $P(A_{ij}) = \frac{2}{j-i+1}, i < j$, denn A_{ij} ist das Ereignis, dass entweder l_i oder l_j als erstes Vergleichselement unter den $l_i, l_{i+1},...,l_j$ ausgewählt wird und jedes der *j-i+1* Elemente $l_i, l_{i+1},...l_j$ wird mit gleicher Wahrscheinlichkeit zuerst ausgewählt.



Denn es gilt (i oder j werden gegriffen oder es wird neben das Intervall gegriffen):

$$P_n(A_{ij}) = \frac{2}{n} + \frac{n-(j-i+1)}{n} \cdot P_{n-1}(A_{ij}) .$$

Somit ist auch $P_{j-i+1}(A_{ij}) = \frac{2}{j-i+1}$ und es gilt

$$\begin{aligned} & P_{(j-i+1)+1}(A_{ij}) \\ &= \frac{2}{j-i+2} + \frac{j-i+2-(j-i+1)}{j-i+2} \cdot P_{j-i+1}(A_{ij}) \\ &= \frac{2}{j-i+2} + \frac{(j-i+2-(j-i+1)) \cdot 2}{(j-i+2) \cdot j-i+1)} \\ &= \frac{2}{j-i+2} \\ &= P_{j-i+1}(A_{ij}) \end{aligned}$$

Zum Erwartungswert der Anzahl der Vergleiche („Durchschnitt“)

Vor dem eigentlichen Beweis müssen noch einige Eigenschaften der sog. „Harmonischen Zahlen“ erwähnt werden. Als „n-te harmonische Zahl“ H_n bezeichnet man

$$H_n = \sum_{k=1}^{n} \frac{1}{k} .$$

Für den weiteren Beweis benötigen wir die Eigenschaft

$$\sum_{j=2}^{n} H_j = (n+1)(H_n - 1) .$$

Dies zeigen wir durch Induktion über n.

Induktionsverankerung für n=2

$$H_2 = 1 + \tfrac{1}{2} = 3 \cdot (1 + \tfrac{1}{2} - 1) = (2+1) \cdot (H_2 - 1)$$

Induktionsschritt von n auf n+1

$$\begin{aligned}
&\sum_{j=2}^{n+1} H_j = \sum_{j=2}^{n} H_j + H_{n+1} = (n+1)(H_n - 1) + H_{n+1} = nH_n - n + H_n - 1 + H_{n+1} \\
&= n(H_n + \tfrac{1}{n+1} - \tfrac{1}{n+1} + 1 - 1) - n + (H_n + \tfrac{1}{n+1} - \tfrac{1}{n+1} - 1) + H_{n+1} \\
&= n(H_{n+1} - 1) - \tfrac{n}{n+1} + n - n + (H_{n+1} - 1) - \tfrac{1}{n+1} + (H_{n+1} - 1) + H_{n+1} \\
&= (n+2) \cdot (H_{n+1} - 1) - \tfrac{n}{n+1} - \tfrac{1}{n+1} + 1 \\
&= (n+2) \cdot (H_{n+1} - 1) + \tfrac{(-n) - 1 + n + 1}{n+1} \\
&= (n+2) \cdot (H_{n+1} - 1)
\end{aligned}$$

Zum eigentlichen Beweis des Erwartungswertes:

$$\begin{aligned}
&E(V) = \sum_{1 \le i < j \le n} P(A_{ij}) = \sum_{1 \le i < j \le n} \tfrac{2}{j-i+1} \\
&= 2 \cdot \left(\tfrac{1}{2} + \tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{4} + \ldots \right) \\
&= 2 \cdot \left(1 + \tfrac{1}{2} + 1 + \tfrac{1}{2} + \tfrac{1}{3} + 1 + \tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{4} + \ldots - (n-1) \right) \\
&= 2 \cdot \sum_{j=2}^{n} H_j - 2(n-1)
\end{aligned}$$

Hieraus ergibt sich mit der oben berechneten Formel

$$\begin{aligned}
&= 2 \cdot \left((n+1)(H_n - 1) \right) - 2(n-1) \\
&= 2 \cdot \left((n+1)H_n - (n+1) \right) - 2(n-1) \\
&= 2(n+1)H_n - 2n - 2 - 2n + 2 \\
&= \boldsymbol{2(n+1)H_n - 4n}
\end{aligned}$$

Schätzt man

$$H_n = \sum_{k=1}^{n} \tfrac{1}{k} \approx \int_1^n \tfrac{1}{x}$$

ab, so ergibt sich für den Erwartungswert ein Wert von

$$2n \ln n + O(n) \approx 2n \ln n \,.$$

Formt man den letzten Wert in

$$1{,}38 \cdot n \cdot \log_2 n$$

um, so erkennt man, dass die durchschnittliche Zahl der Vergleiche bei Quick-Sort nur etwa 38% über dem besten Fall liegt.

4.4 Bemerkungen

- Quick-Sort ist für große Datenmengen gut geeignet. Dabei spielt die Struktur der Liste kaum eine Rolle. Wie bereits bemerkt ist der Rechenaufwand bim Durchschnitt nur ca. 38% höher als im besten anzunehmenden Fall.
- Quick-Sort lässt sich auf unterschiedliche Weisen noch verändern (verbessern)
  - Beim Finden des Pivot-Elements: zufällig, immer der Median,...
  - Bei der Implikation der Rekursion.[6]
- Quick-Sort ist ein sehr ausführlich erforschter Algorithmus. In den o.g. Büchern findet man eine Reihe weiterer Literaturhinweise.

4.5 Beispiel für den Sortiervorgang

	A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
1	A	S									A	M	P	L	E
2	A	A	O						E	X	S	M	P	L	E
3	A	A	E	R	T	I	N	G	O	X	S	M	P	L	E
4	A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
5	A	A	E												
6	A	A													
7					L	I	N	G	O	P	M	R	X	T	S
8					L	I	G	M	O	P	N				
9					G	I	L								
10						I	L								
11									N	P	O				
12										O	P				
13													S	T	X
14														T	X
	A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

[6] vgl. Sedgewick, Sortieralgorithmen, 150ff.