15.

Heaps

Buch Mark Weiss „Data Structures & Problem Solving Using Java“ siehe:
- 833-852 (Heap, Heapsort)

Lernziele Kapitel 15 Heaps

- Heap-Datenstruktur und ihre Implementierung verstehen
- Die Operationen insert und get_min angeben können
- Heapsort verstehen
- Zeitkomplexität Heapsort / Heap-Operationen begründen können

Thema / Inhalt

Heaps sind eine etwas verrückte (aber effiziente und sehr nützliche) Datenstruktur, da muss man erst mal draufkommen! In gewisser Weise stellen sie „halbwegs" sortierte Strukturen dar: Wenn man etwas entfernt oder einfügt, dann muss man nicht viel Aufwand spendieren, damit die Struktur danach auch wieder halbwegs sortiert ist. Und wenn man das kleinste oder das grösste Element haben möchte, oder wenn man gar den ganzen Inhalt in sortierter Weise geliefert bekommen möchte, dann geht das auch recht effizient, weil ja schon alles „halbwegs" sortiert vorliegt.

Heaps eignen sich tatsächlich in idealer Weise zum Sortieren (**Heapsort**): Alles in irgend einer Reihenfolge in den Heap einfügen, dann wiederholt das kleinste Element daraus entfernen. Beides geht bei einem Heap recht effizient. Nur: Wenn man dem Algorithmus bei der Arbeit zuschaut und das Verfahren nicht schon kennt, dann versteht man rein gar nichts – es scheinen

Thema / Inhalt (2)

einfach irgendwelche Elemente wild hin- und hergetauscht zu werden! Wir gehen die Sache aber systematisch an, wodurch sich die Funktionsweise offenbart.

Im Sinne einer **abstrakten Datenstruktur** realisiert ein Heap eine sogenannte „**priority queue**" – ein „Behälter", der Elemente mit einem geordneten Schlüsselwert (z.B. vom Typ int oder float) speichern kann und zwei Operationen anbietet: „insert", womit ein Element in den Behälter eingefügt wird, und „get_min", wodurch das Element mit dem kleinsten Schlüsselwert der im Behälter derzeit gespeicherten entfernt und ausgeliefert wird.

Heaps sind dabei als Binärbäume organisiert, bei denen alle inneren Niveaus vollständig gefüllt sind, an der Wurzel das kleinste Element steht und alle Pfade von der Wurzel zu einem Blatt aufsteigend sortiert sind. Wie wir sehen werden, kann dann sowohl „insert" als auch „get_min" in logarithmischer Zeit (bezogen auf die Gesamtzahl der Elemente) erfolgen. Vor allem dann, wenn man Heaps niveauweise in einem Array speichert, können die beiden Operationen sehr effizient ausgeführt werden – sie verwirren aber einen unbedarften Beobachter, der die Interpretation als Binärbaum nicht erkennt.

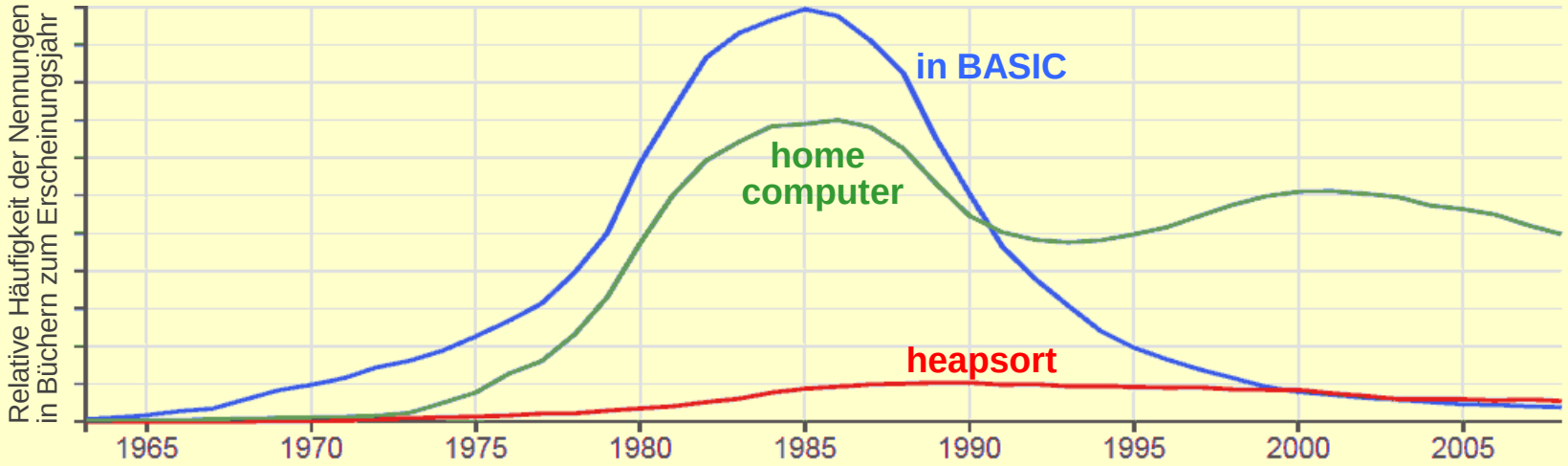
Heaps lassen sich verschiedentlich anwenden. Zum Beispiel qualifizieren sie sich als Datenstruktur für die Planungsliste bei der ereignisgesteuerten Simulation: Dynamisch entstehende Ereignisnotizen, die in Zukunft auszuführen sind, werden entsprechend ihres Eintrittszeitpunktes eingefügt; zyklisch wird vom Simulator die kleinste Ereignisnotiz entnommen und zur Ausführung gebracht. Häufig ist auch der Einsatz in Vorrangwarteschlangen, wie sie bei Servern oder Betriebssystemen zur Festlegung der Ausführungsreihenfolge von Aufgaben benötigt werden.

Java bietet mit **java.util.PriorityQueue** Heaps als direkt nutzbare Datenstruktur an.

Thema / Inhalt (3)

Heapsort wurde 1964 entdeckt und veröffentlicht. Im gleichen Jahr wurde die Welt mit der Programmiersprache **BASIC** beglückt: Eine gegenüber Algol, Fortran und Cobol deutlich einfachere, für Anfänger geeignete Programmiersprache, die vor allem interaktiv auf den bald danach aufkommenden Heim- und Hobbycomputern genutzt werden konnte und auf einem Laufzeitinterpreter statt einem Compiler beruhte, was die eigentliche Ausführung zwar verlangsamte, aber das System einfacher und direkter anwendbar machte. In unserem historischen Strang zeigen wir eine Implementierung von Heapsort in BASIC und vermitteln einen kurzen Eindruck davon, wie seinerzeit mit den Heimcomputern, die noch nicht das Prädikat „PC" trugen, programmiert wurde.

Ein **visueller Wettlauf** verschiedener Sortierverfahren und eine **Vertonung von Heapsort** beschliessen das Kapitel.



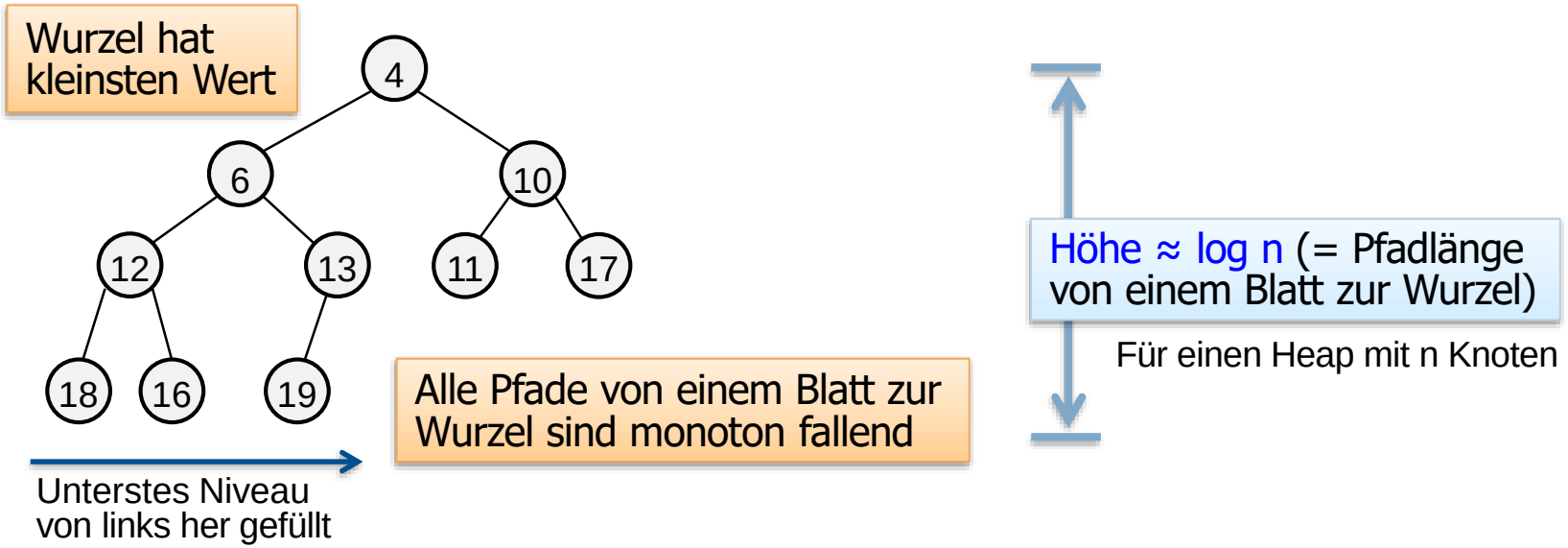
Quelle: Google Books

Die Heap-Datenstruktur

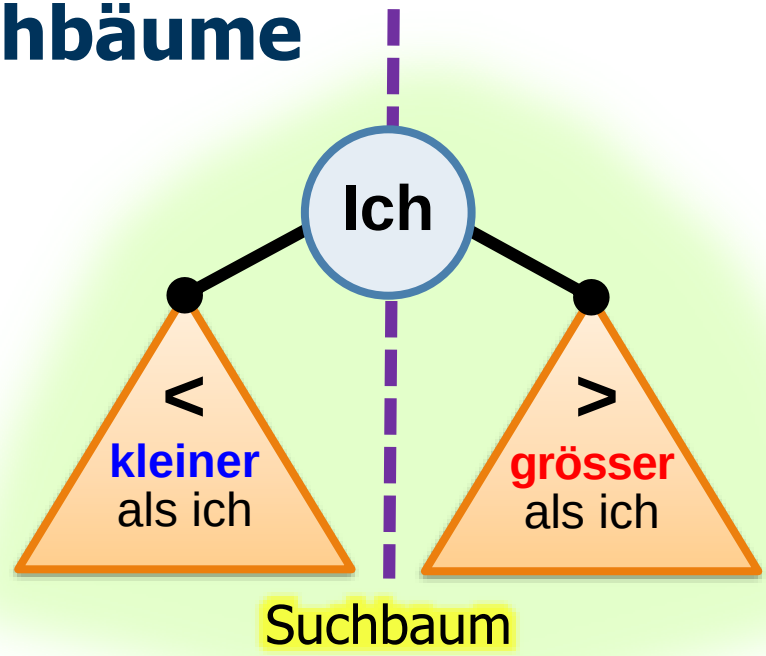
- Heap = Binärbaum, für den gilt:
  - Er ist (fast) vollständig
  - Knoten besitzen einen Wert (wir gehen vereinfachend davon aus, dass dieser eindeutig ist, dass also die Koten unterschiedliche Werte haben)
  - Für alle Knoten k ≠ Wurzel: Wert(Vorgänger(k)) < Wert(k)

    (alternative Def. mit ≤, >, ≥ auch möglich)

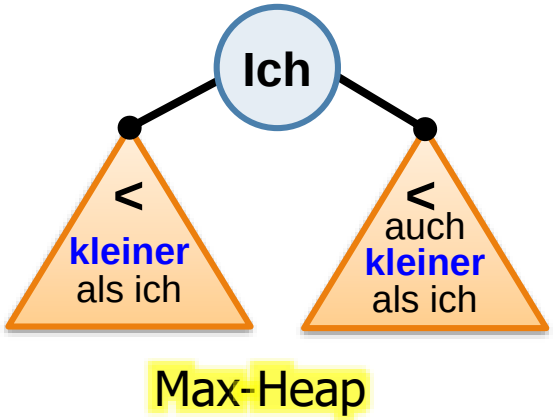
Min-Heap

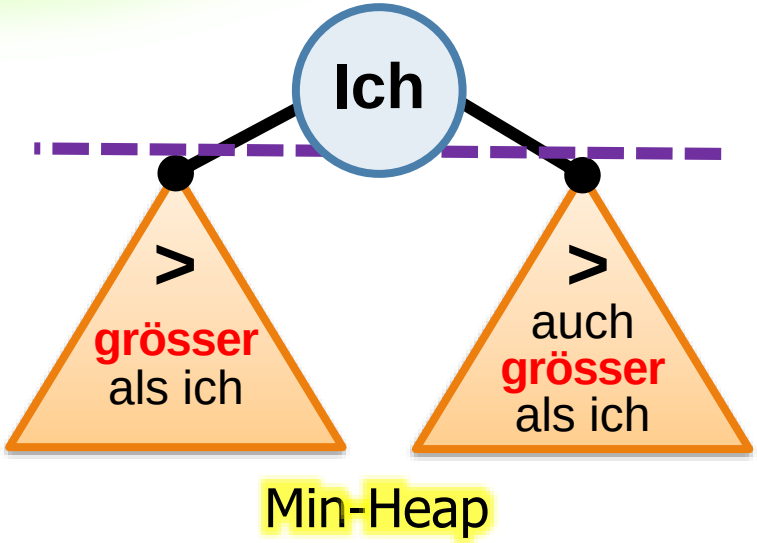


Heaps ⇔ Suchbäume



Beachte auch:
Ein x-beliebiges Element im linken Unterbaum ist bei einem *Suchbaum* kleiner als ein y-beliebiges Element des rechten Unterbaums. Eine solche Eigenschaft bzgl. der Unterbäume gilt aber nicht bei den „schwächer" geordneten *Heaps*!



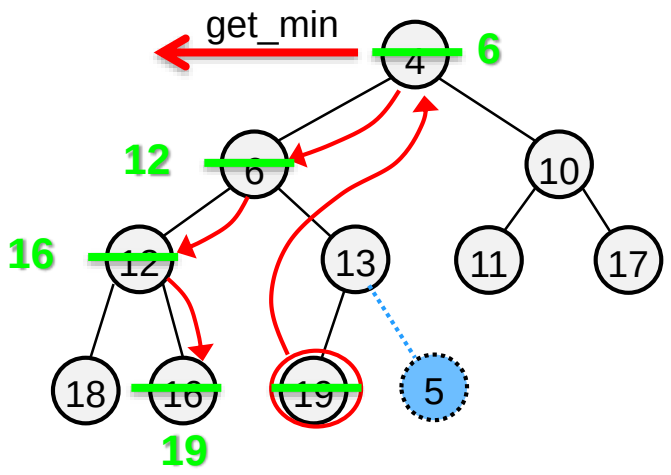


Heap-Operationen: get_min und insert

Man überlege sich genau, dass das der Fall ist!

Für get_min *und* insert gilt:
- Lässt Heap-Eigenschaft invariant
- Benötigt O(log n) Schritte

Im Unterschied zur Implementierung von priority queues als sortierte / unsortierte verkettete Liste!

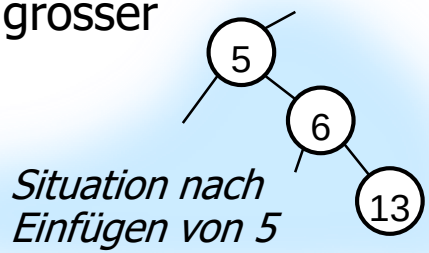


get_min:

- Wurzel (hier: 4) entfernen
- Letzten Knoten des untersten Niveaus (hier: 19) an die Wurzelposition setzen
- Neue Wurzel so weit wie möglich nach unten sinken lassen: Mit kleinerem der beiden Nachfolger vertauschen; dies rekursiv (oder iterativ) auf entsprechenden Unterbaum anwenden

insert:

- Als neues nächstes Blatt (hier: 5) auf unterstem Niveau einfügen
- „Hochbubbeln“: Soweit wie möglich nach oben wandern lassen – iterativ mit Vorgänger vertauschen, wenn dieser grösser

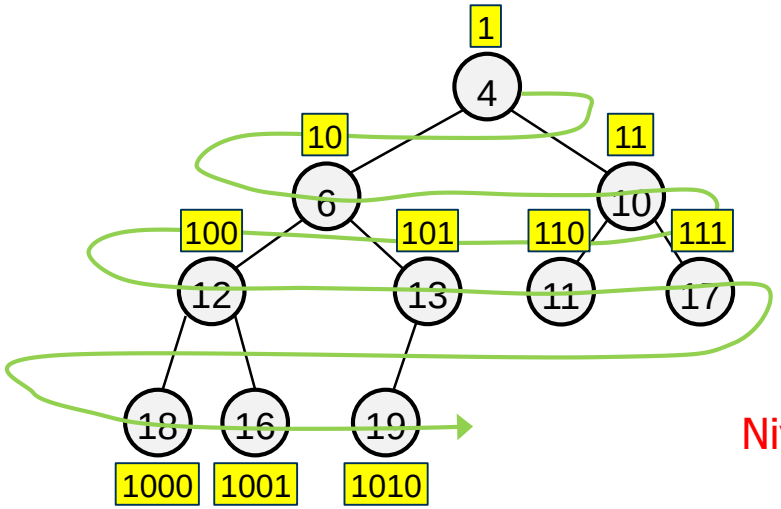


Situation nach Einfügen von 5

Heaps niveauweise als Array

Vgl. dazu frühere Slide „Binärbäume in Arrays“

- Wurzel steht bei Array-Index 1
- Direkte Nachfolger eines Knotens mit Index i haben die Indizes 2i und 2i+1



Beispiel für einen Heap, der solcherart „niveauweise“ gespeichert ist

4	6	10	12	13	11	17	18	16	19
1	2	3	4	5	6	7	8	9	10

Niveau 1 2 3 4 ...

- Damit ist Aufsteigen / Absteigen entlang eines Astes bei get_min / insert besonders einfach!
  - Halbieren / Verdoppeln von Indizes

PARENT (i)
→ return $\lfloor i/2 \rfloor$
LEFT_CHILD (i)
→ return $2i$
RIGHT_CHILD (i)
→ return $2i+1$

Heap: Implementierung von „insert“
(bei niveauweiser Speicherung des Heaps in einem int-Array „a“)

Als neues Blatt auf unterstem Niveau einfügen.

Soweit wie möglich nach *oben wandern* lassen: Iterativ mit Vorgänger vertauschen, wenn dieser grösser.

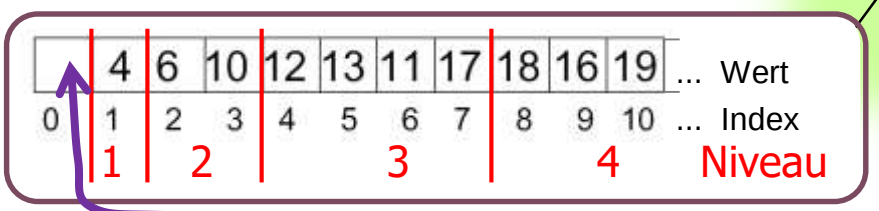
```
void insert (int x) {
  int k = ++N;
  a[0] = -1;
  while (a[k/2] >= x) {
    a[k] = a[k/2];
    k = k/2;
  }
  a[k] = x;
}
```

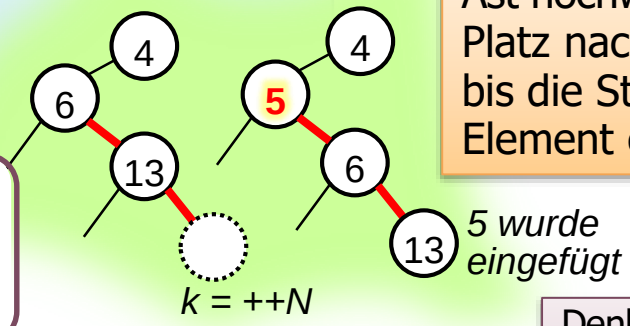
N bezeichnet die Anzahl der gespeicherten Elemente

Ein „Trick“, damit die Schleife *immer* verlassen wird

k/2 wird evtl. abgerundet

Ast hochwandern, alles einen Platz nach unten schieben, bis die Stelle für das neue Element gefunden ist





5 wurde eingefügt

Denkübung: (1) Kann man „>“ statt „>=“ verwenden? (2) Was geschieht, wenn man ein schon gespeichertes Element einfügt?

- Trick mit „superkleinem“ Wert -1 bei a[0] klappt so nur, wenn keine negativen Werte eingefügt werden!
- Beachte: Bei Ersetzen von k/2 durch k-1 erhält man insertion sort
  ⇒ Interpretation: Beim Heap wird insertion sort entlang eines einzigen Astes angewandt; dieser hat hier aber nur eine Länge von ≈ log n

Heap: „get_min“

```
int get_min() {
  int k = 1; int j; int x = a[1];
  a[1] = a[N--];
  int w = a[1];
  while (k <= N/2) {
    j = k + k;
    if (j<N && a[j+1] < a[j])
      j++;
    if (w <= a[j])
      break;
    a[k] = a[j];
    k = j;
  }
  a[k] = w;
  return x;
}
```

Letzter Knoten wird neue Wurzel

Shortcut! (Wieso nicht „&“ ?)

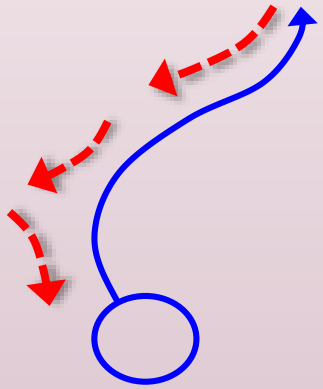
Platz k gefunden: kleiner als beide Nachfolger (falls vorhanden)

Wurzel w hin-absin-ken lassen

Beispiel für j und k

k	1	3	6	12
j	3	6	12	25

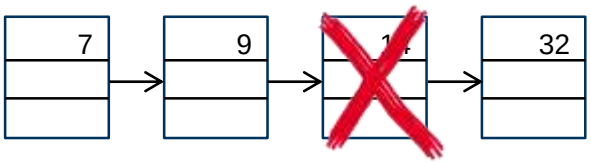
Wurzel entfernen, danach letzten Knoten des untersten Niveaus an die Wurzelposition setzen.

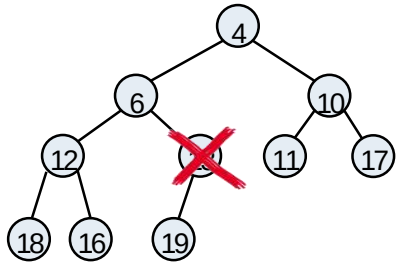


Neue Wurzel so weit wie möglich nach *unten sinken* lassen: Iterativ mit kleinerem der beiden Nachfolger vertauschen…

Denkübung: Was geschieht bei Anwendung auf einen leeren Heap?

Heap: „cancel“





- Manchmal ist eine cancel-Operation praktisch
  - Ein gewisses früher eingefügtes Element x („vorzeitig“) herausnehmen
- Beispiele (bei Priority-Queues):
  - Reisebüro-Szenario: Bei rechtzeitiger Bedienung die (nur für den Eventualfall vorgemerkte) Ereignisnotiz für „Geduld-Ende“ canceln
  - Druckauftrag annullieren bzgl. einer Drucker-Warteschlange
  - Entfernen eines „timer events“, das einem nur im seltenen Notfall (z.B.: Kommunikationspartner antwortet lange nicht) erlösen sollte
- Lösungsidee („Anti-Heap“):
  - Element x nicht wirklich entfernen, sondern für ungültig erklären
  - Ein Anti-Heap verwaltet die Ungültigkeitsnotizen; das zeitlich nächste ungültige Element steht dort immer an der Wurzel
  - cancel(x): x in den Anti-Heap einfügen
  - get_min modifizieren: wenn das zurückgelieferte Element x auch das nächste Element im Anti-Heap ist, dann ist es ungültig → x aus beiden Heaps (an der Wurzel) entfernen und get_min rekursiv aufrufen

Nutzung von Priority Queues

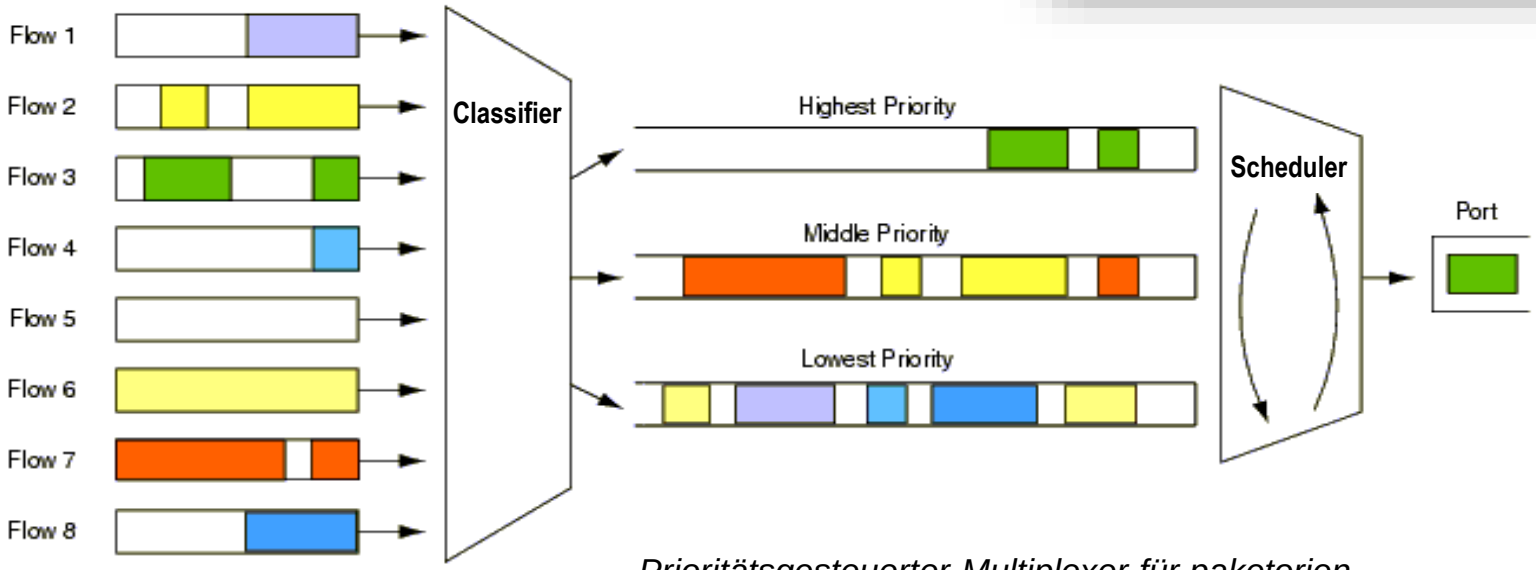
- Simulator-Ereignislisten
- Betriebssysteme, Kommunikationssysteme
  - Job-Scheduling: Job mit *höchster* Priorität als nächsten
  - Routing: Datenpakete *hoher Priorität* (z.B. für Interaktion) bevorzugen
- Spielbaumanalyse, Suchverfahren
  - *Besten* Zug zuerst analysieren
  - *Aussichtsreichstes* Teilproblem zuerst bearbeiten
- Codierungstheorie, Dateikomprimierung
  - Kurze Codewörter für *häufigste* Zeichen („Huffman-Code")

Wir rufen ein früheres Problem in Erinnerung: Eignen sich Heaps (besser / schlechter – und in welcher Hinsicht?) zur Implementierung der dynamischen Terminplanungsliste (Bsp.: „Reservation von Landezeitpunkten")

Another example is in weapons systems, say in a navy cruiser. Numerous threats—airplanes, missiles, submarines, and so on—are detected and must be prioritized. For example, a missile that's a short distance from the cruiser is assigned a higher priority than an aircraft a long distance away so that countermeasures (surface-to-air missiles, for example) can deal with it first. -- Robert Lafore

Priority Q's für wenige Prioritätsklassen

- Dann evtl. wie am Flughafen
  - Pro fester Prioritätsklasse eine eigene FIFO-Warteschlange
  - Auswahl („dequeue") prioritär aus höherrangierter Teilwarteschlange



Prioritätsgesteuerter Multiplexer für paketorientierte Datenströme bei drei Prioritätsklassen

Heapsort

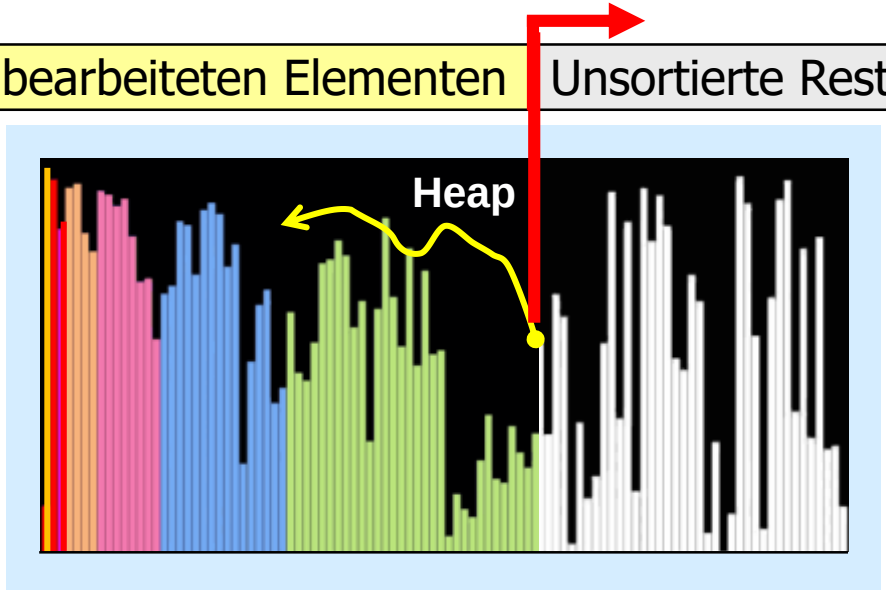
Beide Phasen fügen jeweils n Elemente ein bzw. aus, was jeweils log n Einzelschritte benötigt (Die Astlänge ist nur „kurzzeitig" wesentlich kleiner als log n); jede Phase hat daher O(n log n) Aufwand, zusammen daher ebenfalls O(n log n)

- Sortieren mit einem Heap:
  1) n Elemente (aus unsortierter Ausgangsfolge) nacheinander einfügen
  2) danach n mal *get_min* auf den Heap anwenden

  ⇒ Sortierverfahren mit Zeitkomplexität O(n log n) im worst case (und best case)

- Man kann den Heap im Array selbst („in situ", ohne zusätzlichen Platzbedarf) aufbauen, indem dort der Heap von links heranwächst, während nacheinander Elemente des unsortierten (rechten) Teils entfernt werden:

Phase 1 | Heap aus bereits bearbeiteten Elementen | Unsortierte Resteingabe

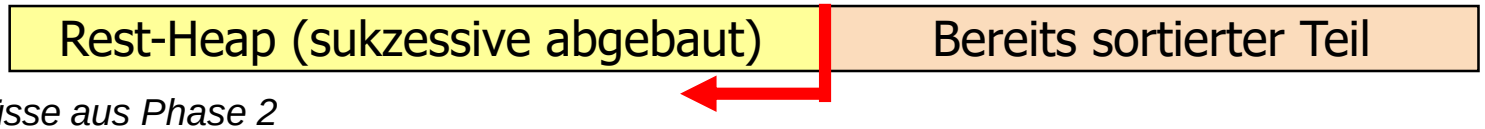


Schnappschuss aus Phase 1 (Hier mit einem Max-Heap)

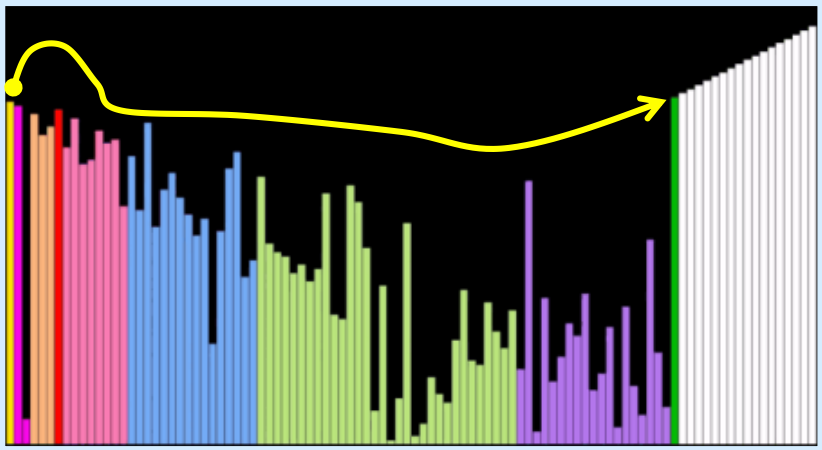
Heapsort (2)

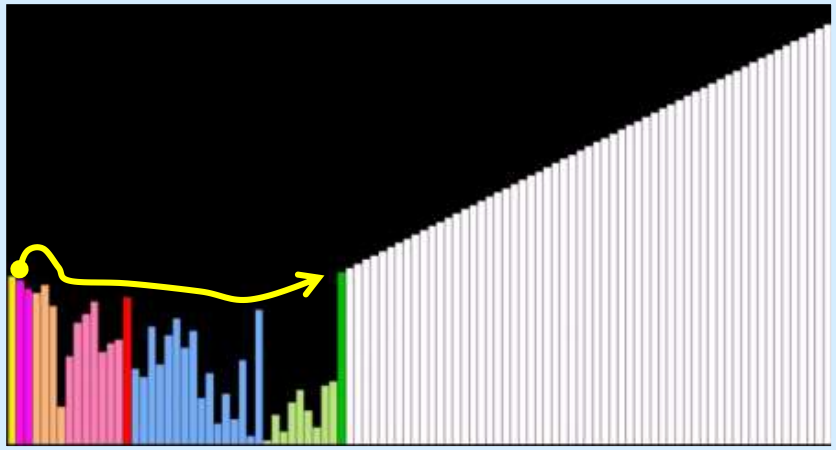
- Anschliessend wird der Heap schrittweise von rechts abgebaut und das jeweils entfernte Element an eine (im ganz rechts sukzessiv freigeräumten Bereich) heranwachsende sortierte Folge angefügt:

Phase 2



Zwei Schnappschüsse aus Phase 2

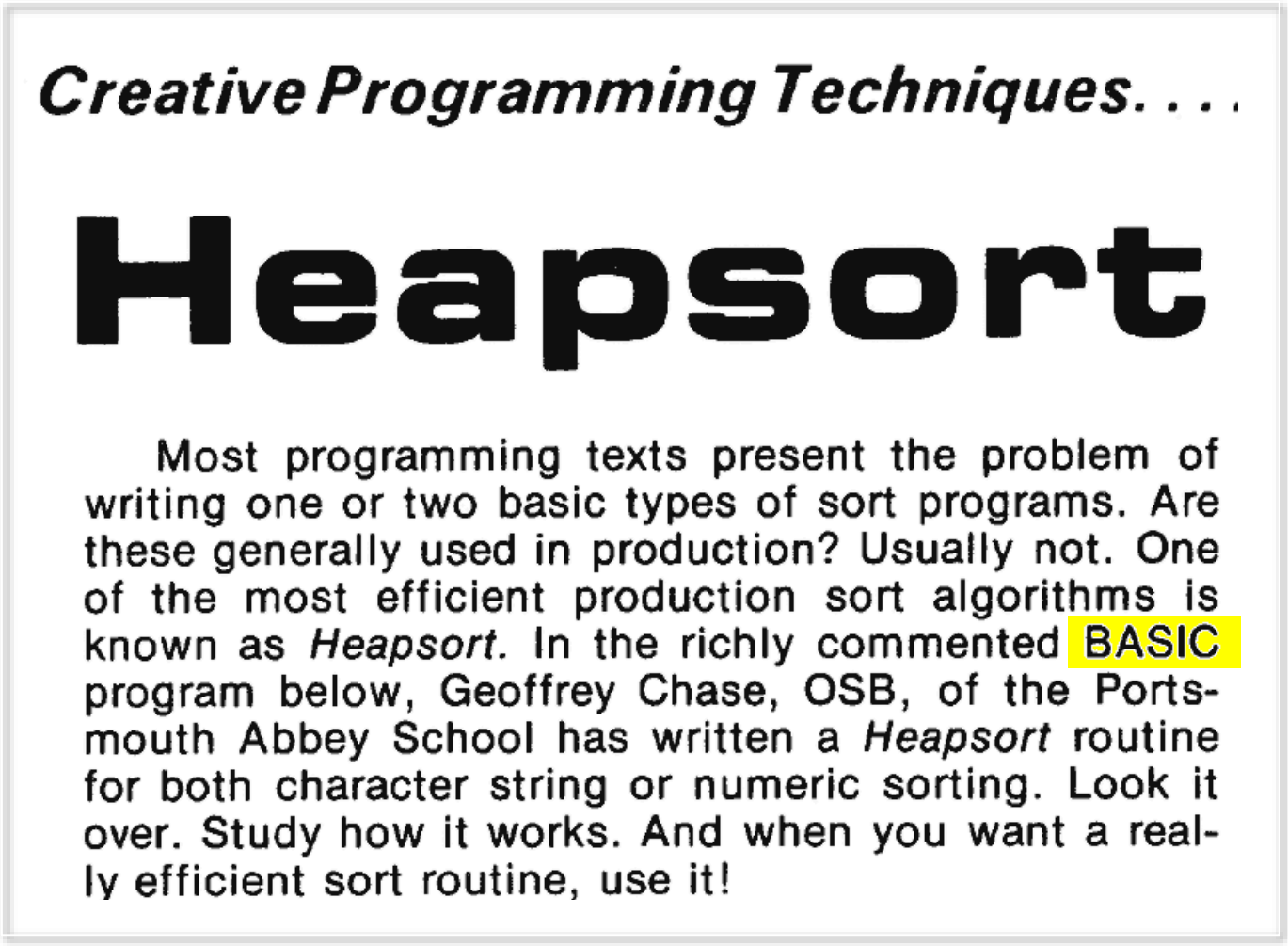




Man kann die erste Phase noch beschleunigen: Statt n mal insert anzuwenden, wendet man auf das initiale unsortierte array eine Operation „buildheap" an, die in O(n) einen Heap erzeugt: Dazu wendet man rekursiv buildheap auf den linken und den rechten Unterbaum der Wurzel an und lässt sodann die Wurzel (wie bei get_min) durch paarweises Vertauschen so weit wie möglich hinabsinken. Oder bottom-up: Die Blätter sind bereits heaps; betrachte Knoten über den Blättern bzw. über linken und rechten Teilheaps und „repariere"...

Heapsort im Jahre 1974
(Aus: Creative Computing Magazine)

Heapsort wurde 1964 entdeckt



Creative Programming Techniques. . . .

Heapsort

Most programming texts present the problem of writing one or two basic types of sort programs. Are these generally used in production? Usually not. One of the most efficient production sort algorithms is known as *Heapsort.* In the richly commented BASIC program below, Geoffrey Chase, OSB, of the Portsmouth Abbey School has written a *Heapsort* routine for both character string or numeric sorting. Look it over. Study how it works. And when you want a really efficient sort routine, use it!

Die folgenden drei Slides stellen lediglich eine „historische Kuriosität" dar.

Interessierte mögen aber Spass am „reverse engineering" des Programms haben und nebenbei einen Eindruck der seinerzeit populären Programmiersprache „BASIC" bekommen.

Die Programmiersprache BASIC wurde 1964 entwickelt: Einfach, interpretiert, interaktiv; passend zum Heimcomputer der 1970er-Jahre.

```
010 REM. KNUTH/WILLIAMS/FLOYD  HEAPSORT ALGORITHM.
020 !  PAS '74
030 DIM N(150),C$(150)
040 PRINT
050 PRINT
060 PRINT "TYPE  C  FOR CHARACTER STRING SORT,"
070 PRINT "TYPE  N  FOR NUMBER SORT.";
080 INPUT W$
090 N=0                              ! START COUNT=N AT 0
100 PRINT
110 IF W$="N" THEN 480
120 IF W$<> "C" THEN 60              ! BAD REPLY
130 REM ********* CHARACTER STRING SORT ROUTINE: *********
140 GOSUB 770                        ! ASK FOR STOP CODE
150 INPUT S$                         ! GET STOP CODE
160 PRINT
170 N=N+1                            ! INPUT LOOP:
180 INPUT C$(N)
190 IF C$(N)<>S$ THEN 170
200 N=N-1                            ! END OF INPUT...
210 PRINT
220 L=INT(N/2)+1                     ! HEAPSORT PROPER:
230 N1=N                             ! PRESERVE N, USE N1
240 IF L=1 THEN 280
250 L=L-1
260 A$=C$(L)
270 GOTO 320
```

Keine Klein-buchstaben

Das musste man Buchstabe für Buchstabe aus der Zeitschrift abtippen – es gab je keinen Server und kein Internet, um es herunterzuladen

Keine Schleifen, son-dern bedingte und unbedingte Sprünge

```
280 A$=C$(N1)
290 C$(N1)=C$(1)                    ! MOVE TOP OF HEAP TO END
300 N1=N1-1                         ! HEAP IS ONE SMALLER NOW
310 IF N1=1 THEN 430                ! ONLY ONE LEFT? THEN WE'RE DONE.
320 J=L                             ! NO, CONTINUE
330 I=J
340 J=2*J                           ! LOOK FOR "SONS" OF I
350 IF J=N1 THEN 390
360 IF J>N1 THEN 420                ! "N1" IS SIZE OF ACTIVE LIST
370 IF C$(J)>=C$(J+1) THEN 390      ! CHOOSE LARGER "SON"
380 J=J+1
390 IF A$>=C$(J) THEN 420
400 C$(I)=C$(J)
410 GOTO 330                        ! LARGER SON REPLACES PARENT
420 C$(I)=A$
425 GOTO 240                        ! END OF SORT...
430 C$(1)=A$
440 FOR I=1 TO N
450 PRINT C$(I)                     ! OR REVERSE ORDER: I=N TO 1 STEP -1
455 NEXT I
460 GOTO 60
```

unwesentliche Teile hier weggelassen

```
760 REM **********SUBROUTINE TO INPUT STOP CODE**********
770 PRINT "PLEASE INDICATE A STOP CODE--SOMETHING NOT IN YOUR"
780 PRINT "LIST, WHICH WILL ACT AS AN 'END OF LIST' SIGNAL: ";
790 RETURN
800 END
```

Hobby-BASIC-Programmierer und Home-Computer-Nutzer der 1970er Jahre


```
1249 BYTES FREE
READY.
? LOAD "HEAPSORT"
PRESS PLAY ON TAPE
SEARCHING
FOUND HEAPSORT
? LOAD
READY.
? LIST 280
280 A$=C$(N1)
290 C$(N1)=C$(1)
300 N1=N1-1
310 IF N1=1 THEN 430
320 J=L
330 I=J
340 J=2*J
350 IF J=N1 THEN 390
360 IF J>N1 THEN 420
370 IF C$(J)>=C$(J+1) THEN
380 J=J+1
390 IF A$>=C$(J) THEN 420
400 C$(I)=C$(J)
...
```

Das dauerte einige Minuten

& machte nervige Piepstöne

Wie konnte man damals nur (über)leben? Sind wir froh, im Hier und Heute zu leben!

Magnetband-Kassettenrekorder als Speichermedium


```
Die Kassette 'DEMO85/4' soll Ihnen, werter Anwender, den
KC85/4 vorstellen. Insgesamt sind 4 Programme auf der Kasset-
te enthalten. Soll ein Programm in den Computer geladen
werden, sind folgende Schritte notwendig (siehe auch Punkt
1.4.5. im System-Handbuch KC85/4):
 - Suchen Programmanfang
 - Einschalten KC85/4
 - Cursor auf Zeile mit der Anweisung LOAD bewegen oder LOAD
   auf einer neuen Zeile eingeben
 - Recorder starten
 - bei Ertönen des Vortons (Pfeifton) ENTER-Taste betätigen

Die Programme auf der Kassette sind alle selbststartend. Vor
dem Einlesen eines neuen Programms ist entweder die RESET-
Taste zu betätigen oder der KC kurz auszuschalten.
```

Die meisten BASIC-Interpreter stammten von Microsoft. BASIC war Microsofts erstes und in den frühen Jahren wichtigstes Produkt, mehrere Jahre bevor mit MS-DOS das erste Betriebssystem der Firma auf den Markt kam.

Laden von Programmen

1. Verbinden Sie Ihren COMMODORE-Computer und die COMMODORE-Datassette miteinander.
2. Legen Sie die Programm-Cassette in die Datassette ein, die erste Seite nach oben.
3. Drücken Sie die Taste REWIND, um an den Bandanfang zu spulen.
4. Wenn das Band anhält, auf STOP-Taste drücken.
5. Geben Sie über die COMMODORE-Computer-Tastatur das Wort LOAD ein und drücken Sie auf die RETURN-Taste.
6. Auf dem Bildschirm erscheint PRESS PLAY ON TAPE. Drücken Sie bei der Datassette auf die Taste PLAY.
7. Der Bildschirm wird gelöscht, während das Programm gesucht wird.
8. Nach einigen Sekunden wird der COMMODORE-Computer auf dem Bildschirm melden: FOUND (dahinter der Programmname).
9. Der Bildschirm wird automatisch gelöscht und das Programm wird geladen.
10. Nachdem das Programm geladen ist, gibt Ihnen der COMMODORE-Computer auf dem Bildschirm die Meldung READY.
11. Um das Programm laufen zu lassen, tippen Sie nun über die Tastatur einfach RUN ein und drücken auf die RETURN-Taste. Das Programm startet.
12. Sollte es nicht funktionieren, dann noch einmal bei Punkt 3 starten!

(Es funktionierte mit dieser Tonhöhenkodierung eher selten, meist gab es einen LOAD ERROR)

1978

Let the computer do it!

"Computers are finding their way into the home"

Madison Wisconsin State Journal, September 18, 1978:

If your schedule is too hectic to worry with the details of life such as the balancing of your checking account, or if one more game of checkers with the kids will drive you crazy, for a small investment you can now buy a computer to do the things you don't want to do.

For around $800, any family can buy a home computer […] According to Huron Smith, owner of the Madison Computer Store, […] the computer will do practically anything you tell it to do, but it never does anything on its own. "They're absolutely stupid," she said. "They have no smarts of their own. They have to be programmed." A home computer can be programmed with ordinary words so a buyer doesn't need any special skills. The domestic variety of computers are programmed with BASIC language, which in computer talk is simple English. Len Lindsay, a salesman at the Computer Store, said, "The name implies it's very easy; it took me three days to learn it."

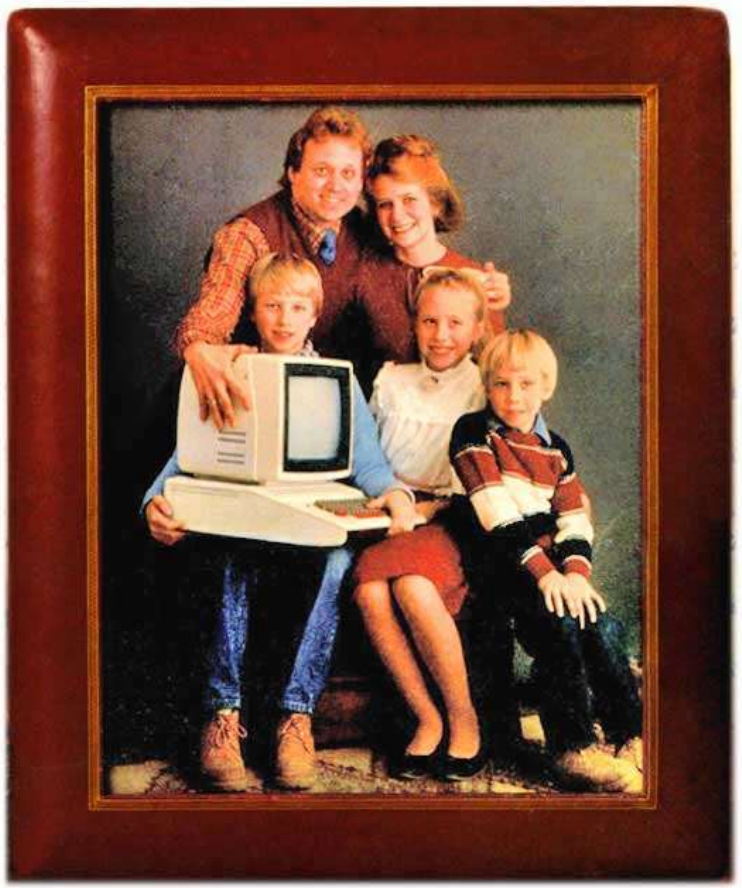
Since May, more than 40 Madison families have turned over their tedious memory and budget chores to their computers. More and more Madison families are programming machines to keep track of Aunt Minnie's birthday and the grandparents' anniversary. Keeping up with the addresses of mobile friends is now the computer's job. And it's no problem to plan for four dinner guests, she can relegate that duty to the machine. The computers are unbeatable as managers of household finances. […]

→

1978 Let the computer do it! (2)

The computer is also an able educator. There are programs to teach children to tell time, and programs for math games, decimal division, history quizzes and to tutor morse code. Lindsay said computers are perfect for kids who need a high level of motivation because computers can be programmed to offer any type of reward from words of praise flashed on the screen to shooting stars. […]

Once you get your computer home, you can add accessories. According to Lindsay, a lot of people in Madison are adding sound. It only takes two wires and costs about $10. Anyone who comes into the store and tinkers with the machines for half an hour wants to have one, Lindsay said. "It's a time saver," he said. "It'll do all the hard things for you plus, as a side benefit, you can sit and play games for free."



Die meisten Käufer von damals hatten weniger einen konkreten Nutzen im Sinn – auch wenn sie den gern vorschoben –, sondern sie wollten einfach auch einen Computer haben, den Inbegriff des Fortschritts und der Moderne. -- Wolf Lotte

1978

Computers in the homes? Wave of the future is now!

It's a weekday evening, sometime in the near future. This Bucks-Mont family has just finished dinner and is settling into its normal weeknight routine. The children go to the VDTs in their room to work on their homework. […] Mom putting the dishes in the dishwasher and programs the central processing unit to do the dishes and, while she's at it, gives the unit new climate control instructions. Meanwhile, Dad is on his way to the store to buy that much-needed additional 2K memory he and the wife have been meaning to get. […]

What it is is the family of the future — completely equipped with its own personal computer. The world that many thought could only exist in a dream or a science fiction […] is finally here.

Computers […] are finding their way into the home. The breakthrough came last year when several firms started marketing home computers — usually composed of a typewriter panel, a television screen and a cassette recorder. The machines now being marketed cost no more than a high quality stereo, and require little more than reading an instruction manual to learn how to operate. The home models presently available can store recipes, do math problems, and even record telephone calls when a person is not home. […] Someday the machines will be doing everything from the wash to income tax returns.

Donald French, merchandising manager for the Tandy Corporation, said his company's market research shows a "very good" potential for home computer sales. […] And many people involved in the industry are predicting computers will become the ultimate home life status symbol. […] French said he and others in the industry are certain the home computer will revolutionize home life, making still more time available for leisure and other activities.

Doylestown Intelligencer, March 18, 1978

Homecomputer – wie es begann

1975 Auszug aus https://blog.hnf.de/der-revolutionaer-altair-8800/:

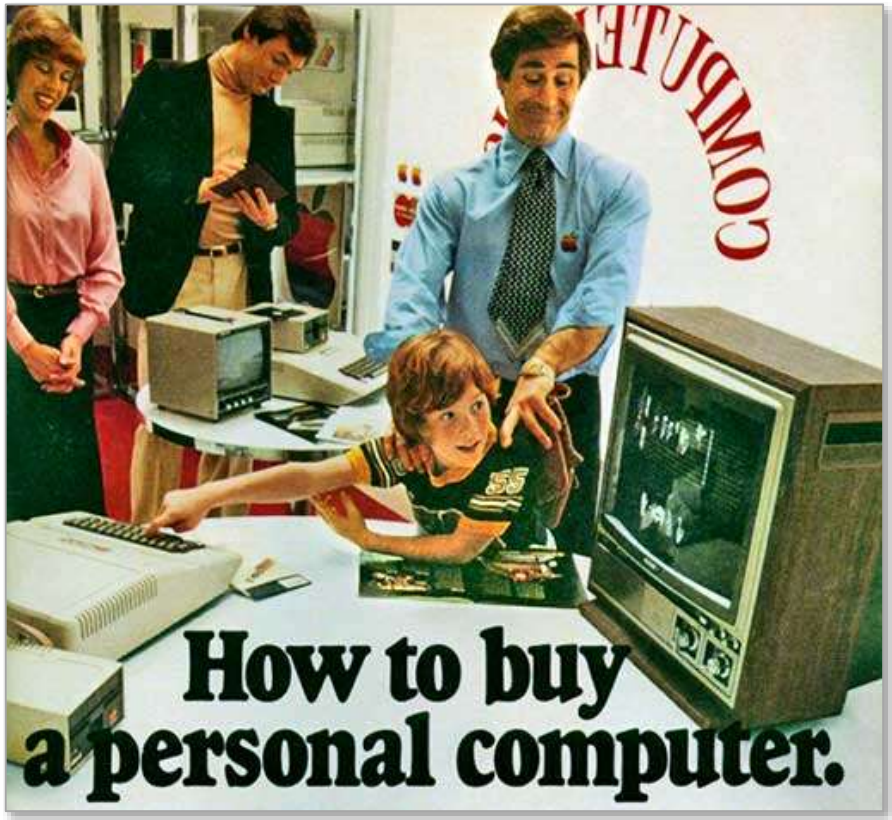
THE era of the computer in every home – a favorite topic among science-fiction writers – has arrived! It's made possible by the POPULAR ELECTRONICS/MITS Altair 8800, a full-blown computer that can hold its own against sophisticated minicomputers now on the market. And it doesn't cost several thousand. dollars.

So begann der Artikel über den Altair 8800 im Januar-Heft 1975 der „Popular Electronics". Das Cover pries den Altair als ersten Minicomputer-Bausatz der Welt an. Bastler erhielten ihn für 397 Dollar. Die Gattung, in die der Altair gesteckt wurde, umfasste die kleinsten Elektronenrechner. Minicomputer bildeten die Kategorie unterhalb der Mainframes von IBM oder Control Data. Sie waren so groß wie Kühlschränke und verbreiteten sich seit den 1960er-Jahren in Firmen, Ämtern, Universitäten und Schulen. Die Minis hatten aber vierstellige Preise und standen nicht in Privathaushalten. „Popular Electronics" erweiterte die Gattung mit dem Altair nach unten. Der Altair nutzte den Intel 8080; er war seit April 1974 erhältlich. Schöpfer des Altair war der 33-jährige Ed Roberts. Seine Firma MITS – Micro Instrumentation and Telemetry Systems – saß in Albuquerque im US-Bundesstaat New Mexiko.

Ohne Zubehör war der Altair fast nutzlos; die Eingaben geschahen per Kippschalter, die Ausgaben durch LEDs. Der S-100-Bus nahm aber Karten mit Speicherchips und für den Anschluss von Peripheriegeräten auf. Der User musste dafür natürlich zahlen. Ab Juli 1975 lieferte MITS auch zwei Versionen einer Programmiersprache, Altair BASIC 4K und 8K. Sie stammten von zwei jungen Männern namens Paul Allen und Bill Gates. Sie gründeten anschließend eine kleine Software-Firma und blieben bis Ende 1978 in Albuquerque. Danach zog man in die Gegend von Seattle, wo Microsoft noch heute sitzt. Der Altair war also Geburtshelfer des größten Software-Unternehmens der Welt.

Sinnvolle Anwendung verzweifelt gesucht

1984 Stiftung Warentest →



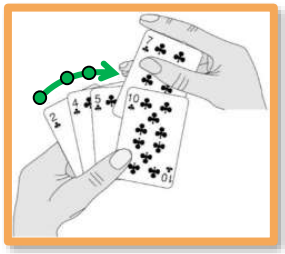
Die Enttäuschung ist vorprogrammiert

Kleine Denksportaufgabe: Man braucht es nicht und trotzdem wird es wie verrückt gekauft. Was ist das? Ganz einfach: ein Heimcomputer. Wir prüften sieben Modelle und suchten verzweifelt nach sinnvollen Einsatzmöglichkeiten. Unser Fazit: Wer auf die elektronische Aufrüstung seines Heimes verzichtet, büßt keine Lebensqualität ein.

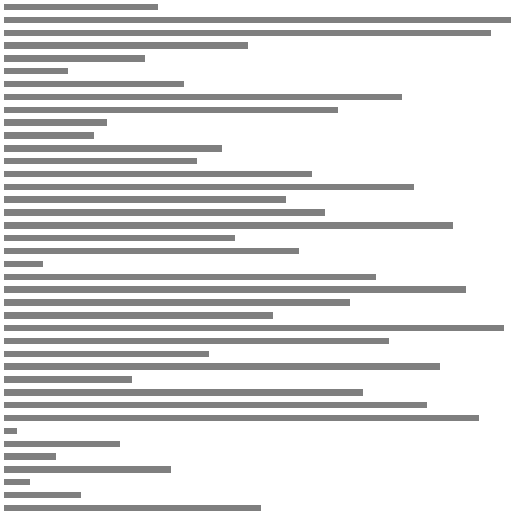
Ende der historischen Notiz

Ein Wettlauf

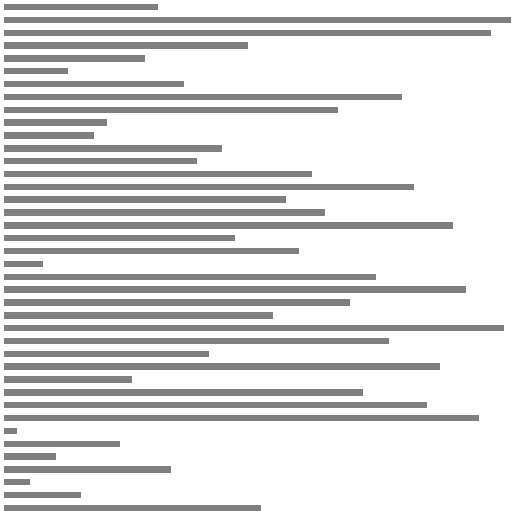
Achtung, fertig,…



Heapsort

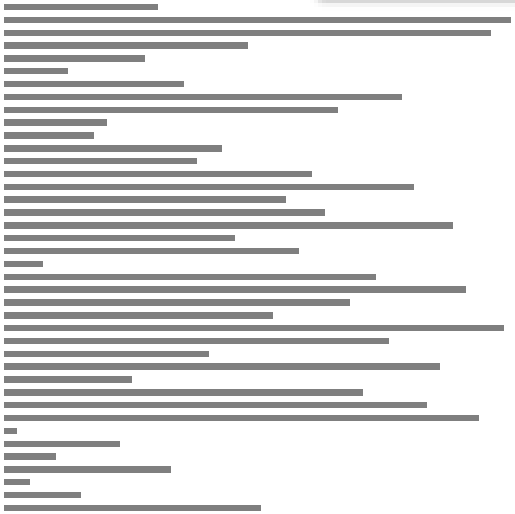


Mergesort



```
// Intervall halbieren: m = n / 2;
// Rekursiv sortieren:
// sort a[1..m]; sort a[m+1..n]
// Sortierte Teil-Arrays „mergen“
```

Insertion sort



```
for (int i = 1; i < n; i++)
{ int t = a[i]; int k;
  for (k = i-1; k >= 0 &&
        a[k] > t; k--)
            a[k+1] = a[k];
  a[k+1] = t;
}
```

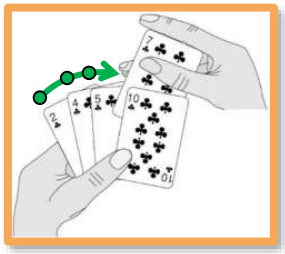
Bekannt aus den Übungen zu Info I

Gleiche Ausgangssituation für alle 3 Algorithmen: 40 Balken der Grösse nach sortieren

www.sorting-algorithms.com

Ein Wettlauf

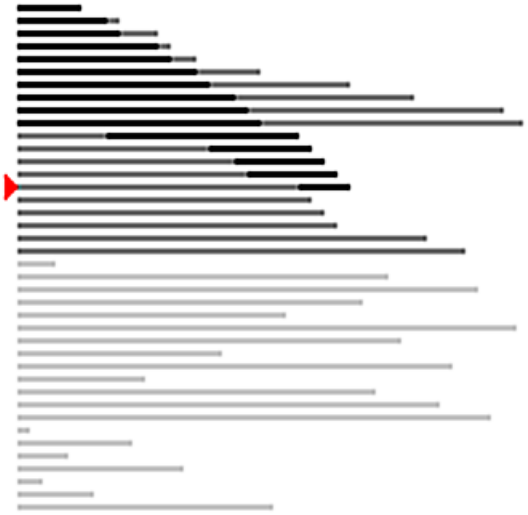
➔ **Schnappschuss nach 4 s:**



Heapsort

- Rotes Dreieck: aktuell betrachtete Stelle des Array
- Graue Werte: unsortiert
- Schwarze Werte: sortiert

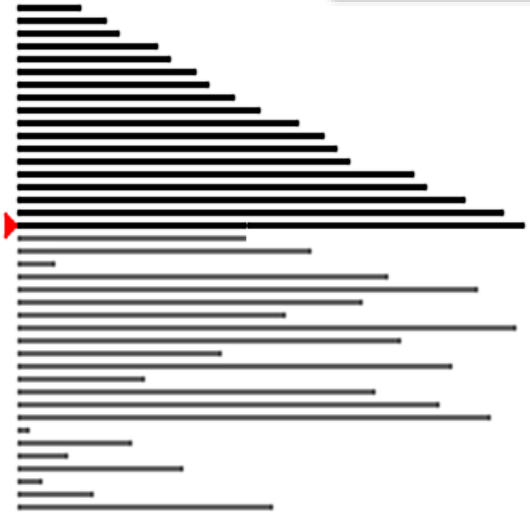
Mergesort



```
// Intervall halbieren: m = n / 2;
// Rekursiv sortieren:
// sort a[1..m]; sort a[m+1..n]
// Sortierte Teil-Arrays „mergen“
```

- Dunkelgrau: aktuelles Intervall bei Mergesort

Insertion sort



```
for (int i = 1; i < n; i++)
{ int t = a[i]; int k;
  for (k = i-1; k >= 0 &&
        a[k] > t; k--)
              a[k+1] = a[k];
  a[k+1] = t;
}
```

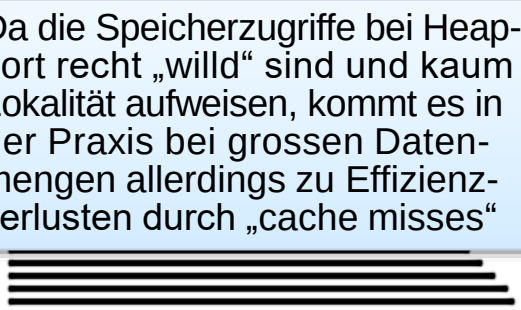
Bekannt aus den Übungen zu Info I

Ein Wettlauf

Heapsort

10 s O(n log n)

Da die Speicherzugriffe bei Heapsort recht „willd“ sind und kaum Lokalität aufweisen, kommt es in der Praxis bei grossen Datenmengen allerdings zu Effizienzverlusten durch „cache misses“



- Rotes Dreieck: aktuell betrachtete Stelle des Array
- Graue Werte: unsortiert
- Schwarze Werte: sortiert

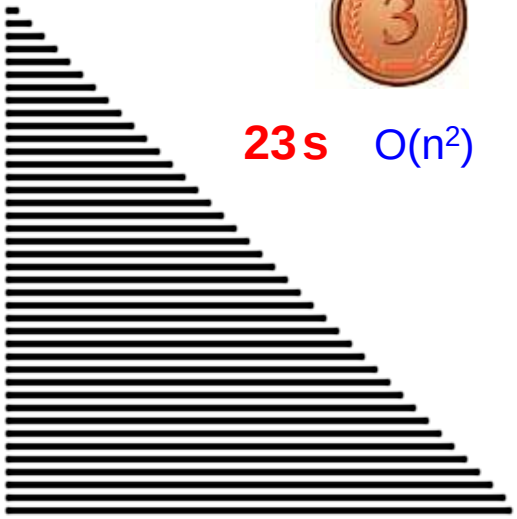
Mergesort

11 s O(n log n)

```
// Intervall halbieren: m = n / 2;
// Rekursiv sortieren:
// sort a[1..m]; sort a[m+1..n]
// Sortierte Teil-Arrays „mergen“
```

- Dunkelgrau: aktuelles Intervall bei Mergesort

Insertion sort



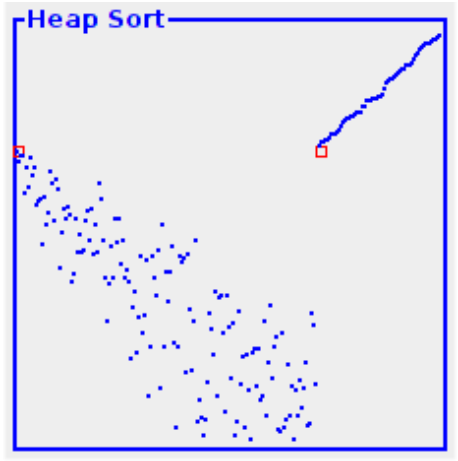
23 s O(n^2)

```
for (int i = 1; i < n; i++)
{ int t = a[i]; int k;
  for (k = i-1; k >= 0 &&
        a[k] > t; k--)
              a[k+1] = a[k];
  a[k+1] = t;
}
```

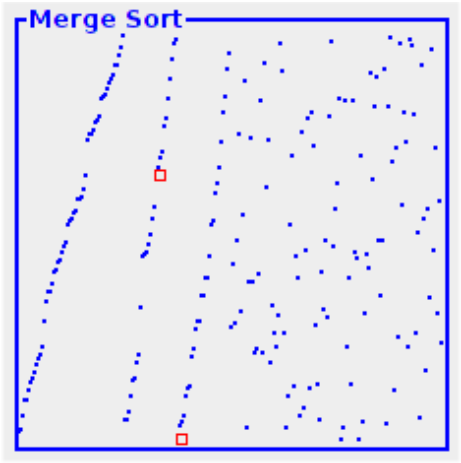
Bekannt aus den Übungen zu Info I

Ein Wettlauf

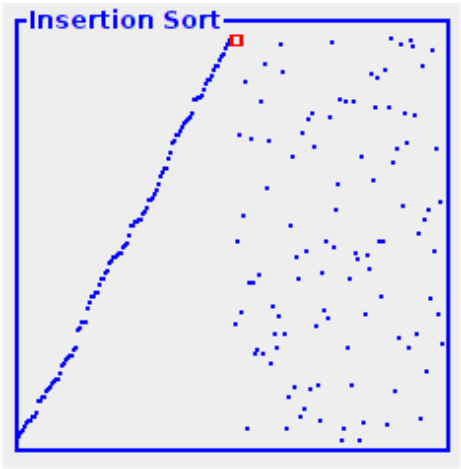
Heapsort



Mergesort



Insertion sort



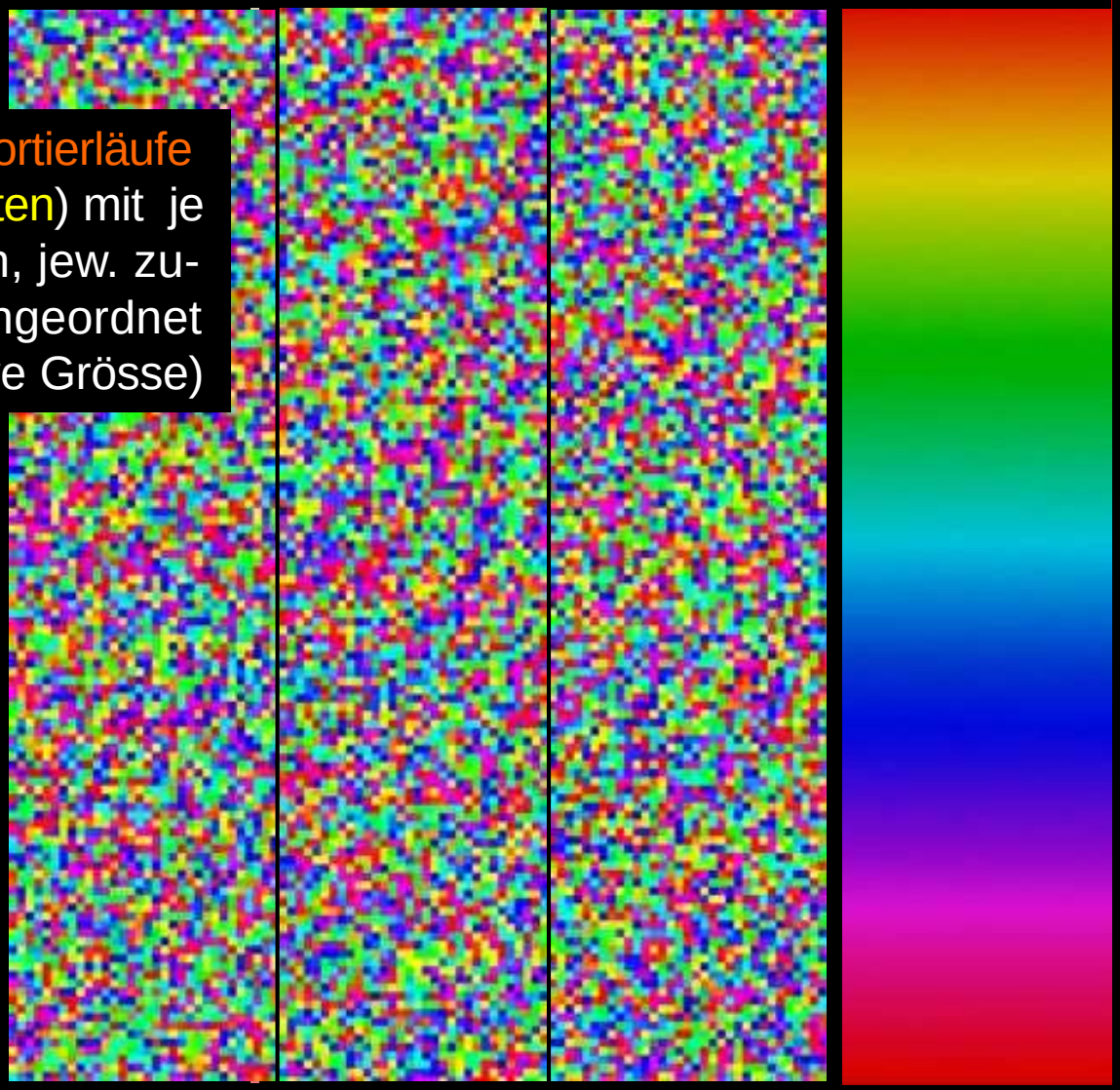
Eine etwas andere Art der Visualisierung durch Schnappschüsse

Bei Heapsort ist die noch zu sortierende Restmenge bereits vorver-
arbeitet und liegt „halbwegs“ (umgekehrt!) sortiert in Heapform vor

Animation: https://i.imgur.com/rWFaMvP.mp4

So sieht das Resultat aus

Quicksort Heapsort Mergesort



35 simultane Sortierläufe (parallele Spalten) mit je 150 Elementen, jew. zufällig anders angeordnet (Farbe = relative Grösse)

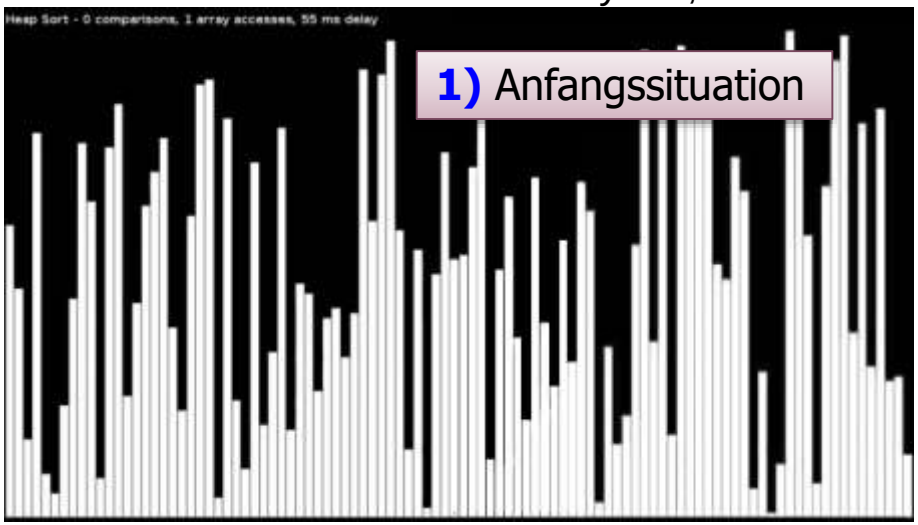
Tipp: Ebenfalls nett anzusehen ist die Sortierscheibe www.youtube.com/watch?v=y9Ecb43qw98

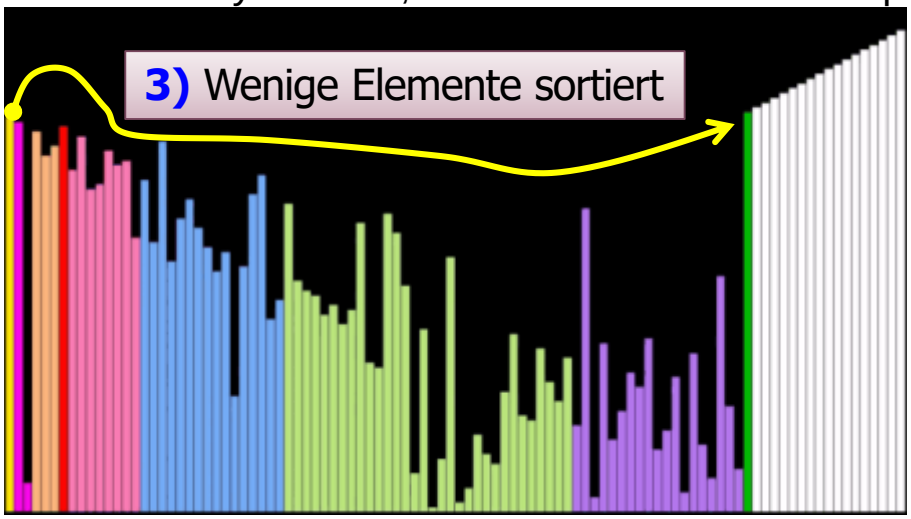
The Sound of Heapsort

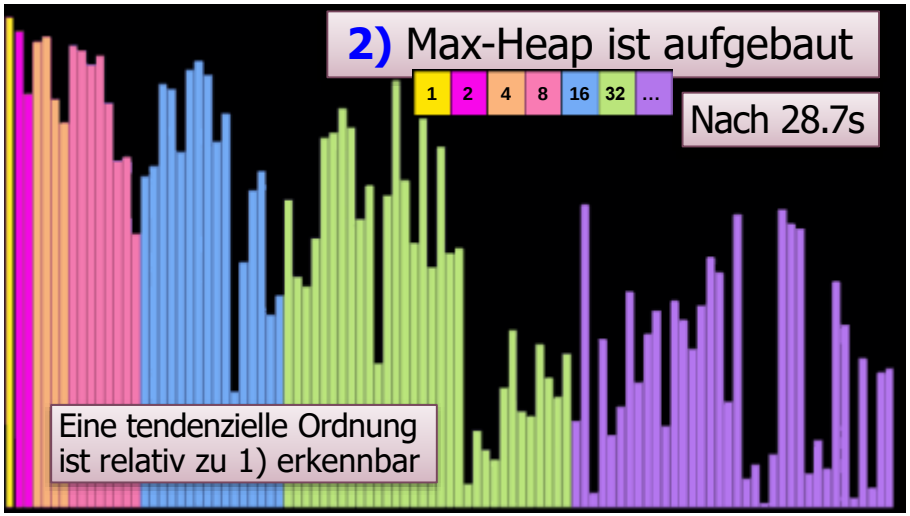
Sortieren von 100 Balken der Länge 1,..,100

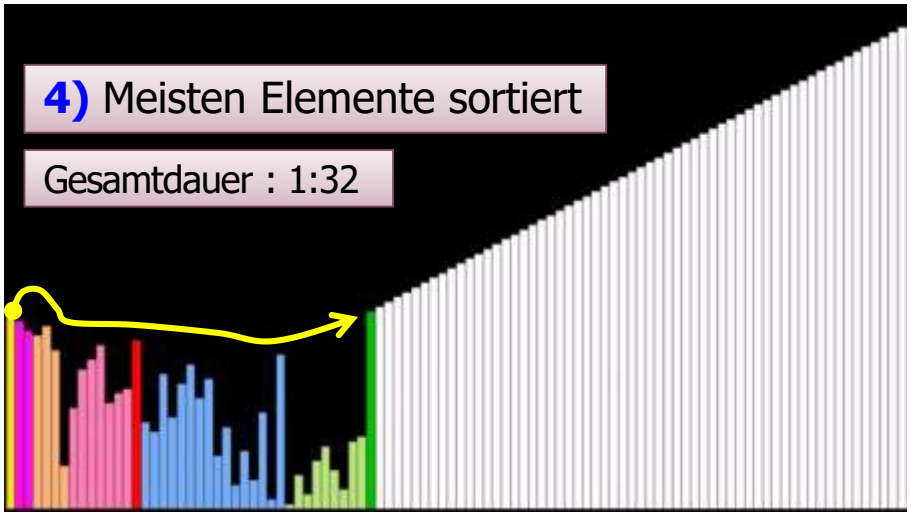
Timo Bingmann, KIT
www.youtube.com/watch?v=_bkow6IykGM

Weiss: unsortierter / sortierter Array-Teil; rot: aktuell betrachtete Array-Position; div. Farben: Niveau im Heap.









Zeitkomplexität bekannter Sortierverfahren

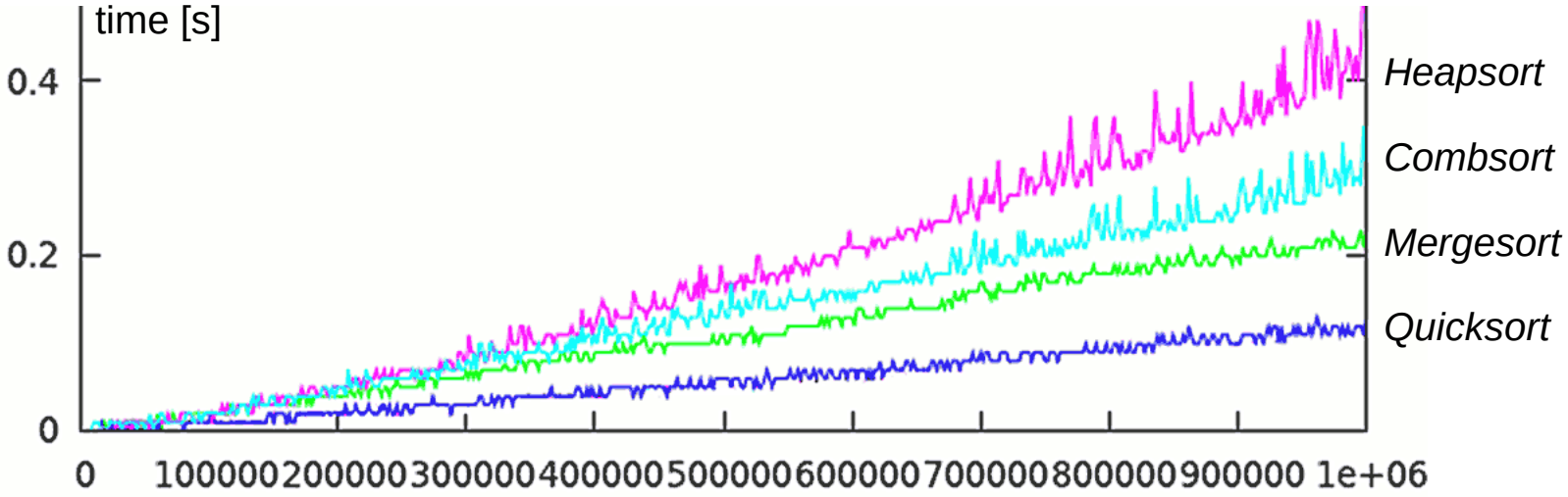
Algorithmus	Best case	Average case	Worst case	Stabil?
Insertion sort	O (n)	O (n^2)	O (n^2)	ja
Selection sort	O (n^2)	O (n^2)	O (n^2)	nein
Bubble sort	O (n)	O (n^2)	O (n^2)	ja
Binary tree sort	O (n log (n))	O (n log (n))	O (n^2)	ja
Mergesort	O (n log (n))	O (n log (n))	O (n log (n))	ja
Quicksort	O (n log (n))	O (n log (n))	O (n^2)	nein
Heapsort	O (n log (n))	O (n log (n))	O (n log (n))	nein

Ein stabiles Sortierverfahren bewahrt die Reihenfolge derjenigen Datensätze, deren Sortierschlüssel gleich sind.

Bei bubble sort wird pro Phase die Folge der Elemente von links nach rechts durchlaufen. Dabei wird in jedem Schritt das aktuelle Element mit dem rechten Nachbarn verglichen; falls die beiden Elemente das Sortierkriterium verletzen, werden sie vertauscht. Solange die Folge nicht sortiert ist, wird eine weitere Bubble-Phase initiiert.

Visualisierung Bubblesort [Wikipedia]

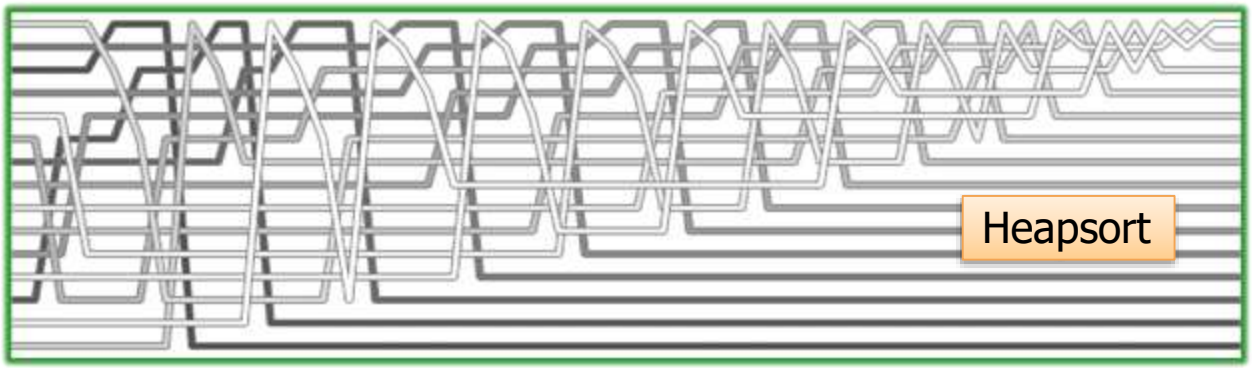
Zeitbedarf von Sortierverfahren in der Praxis



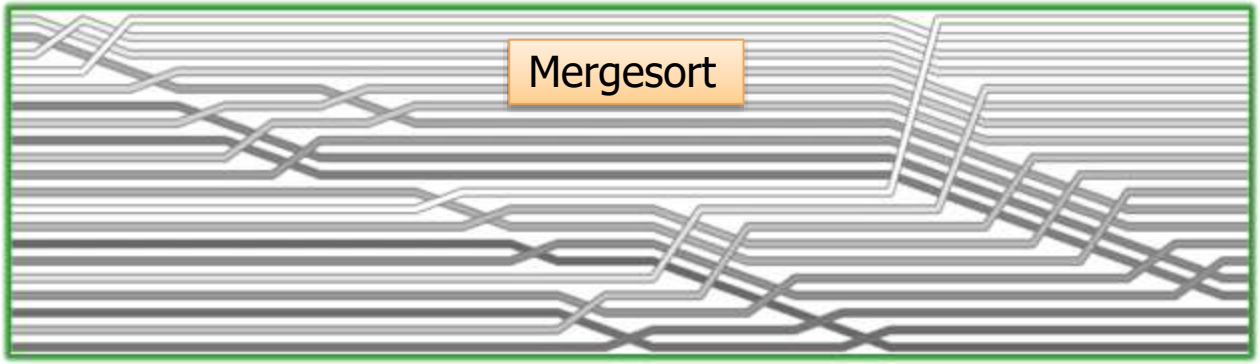
In der Praxis hängt der konkrete Zeitbedarf von der „richtigen“ Implementierung ab, aber auch von den typischen Eigenschaften der Daten (eher eine Zufallspermutation oder oft in Teilen schon vorsortiert?) und sogar von der Computerarchitektur. Heapsort zum Beispiel springt im Speicher wild hin und her, andere Sortierverfahren halten sich eher an das Lokalitätsprinzip – wenn die Daten nicht in den Cache passen, kann sich dies bei Heapsort, wenn keine speziellen Gegenmassnahmen getroffen werden, daher sehr negativ auf die Leistung auswirken.

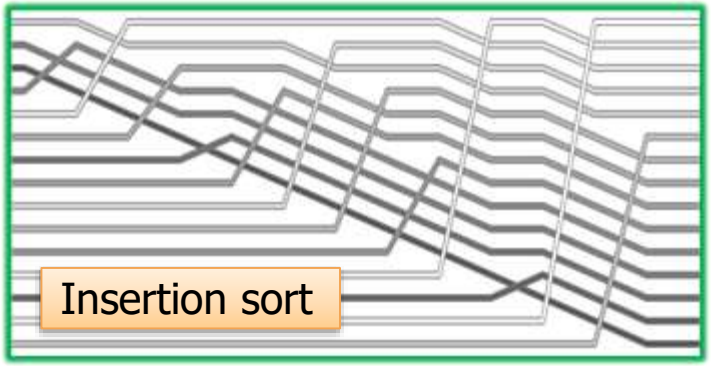
Obiges Bild als Resultat einer Serie von Experimenten stammt von http://c0de-x.com/i-have-to-sort-that-out/; dort heisst es u.a.: “I didn’t care much about optimizing the code by hand, so don’t expect miracles. I repeated the experiments on increasingly larger data sets. 1,000,000 random integers was the largest dataset I fed them with.”

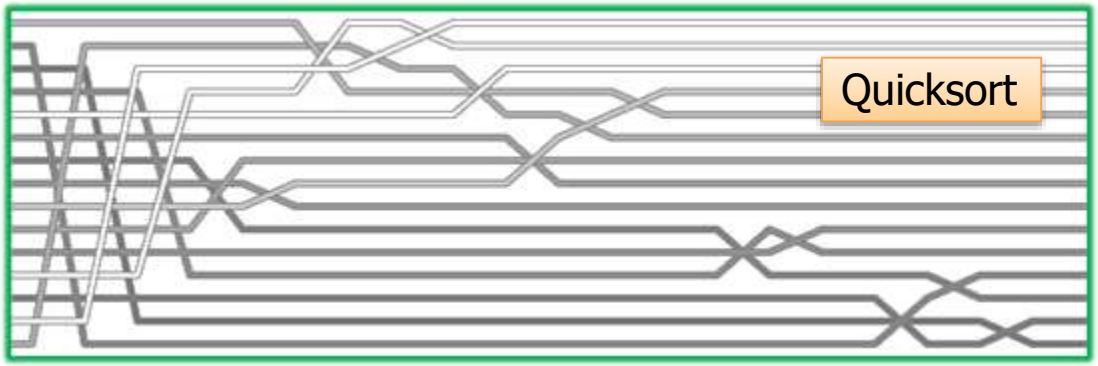
Visualisierung von vier Sortierverfahren



“We can clearly see the heap building step – it is the part of the diagram before the point where the largest element in the array is slotted into place. After that, we can see a repeated pattern – the heap is re-established and the greatest element is moved to below the heap again and again until the array is sorted.”

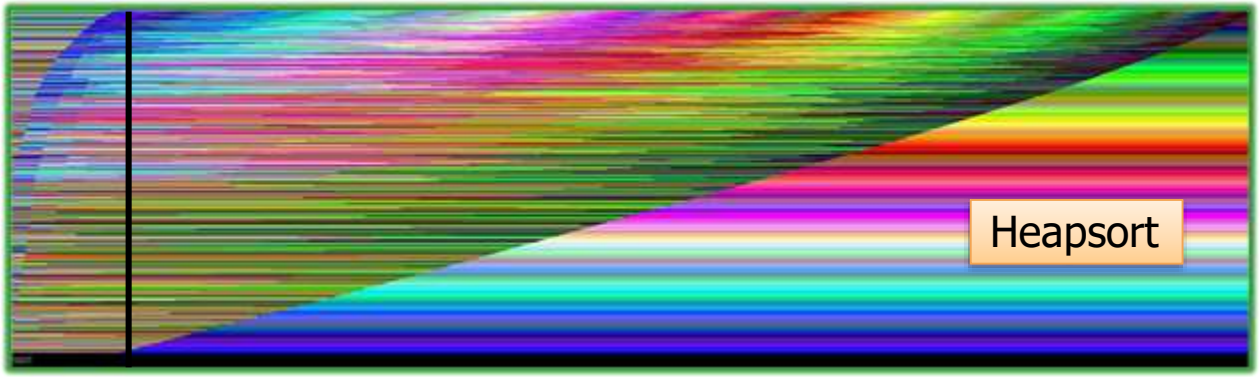


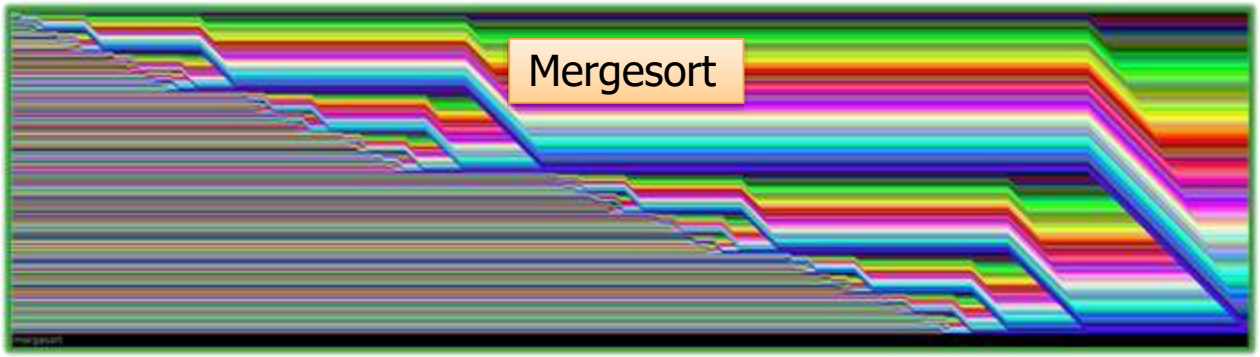




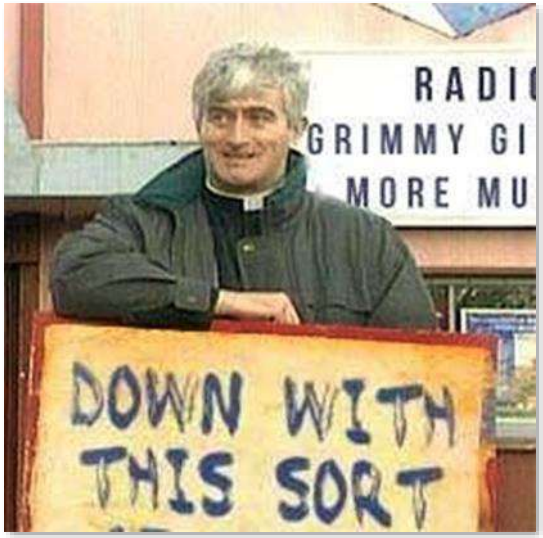
http://corte.si/%2Fposts/code/visualisingsorting/index.html

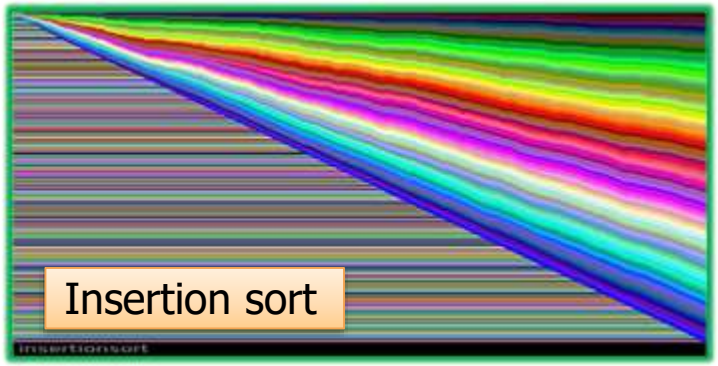
Visualisierung von vier Sortierverfahren

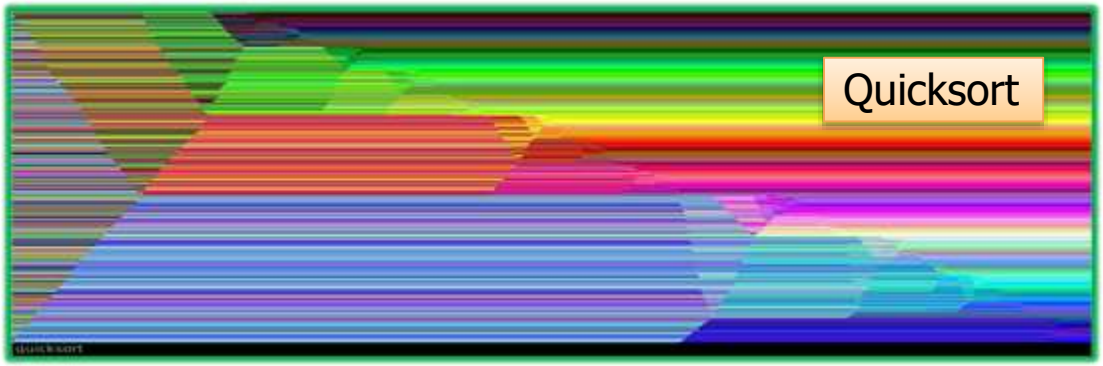




Eine andere Art der Visualisierung, ebenfalls von Aldo Cortesi – mehr Informationen dazu bei sortvis.org

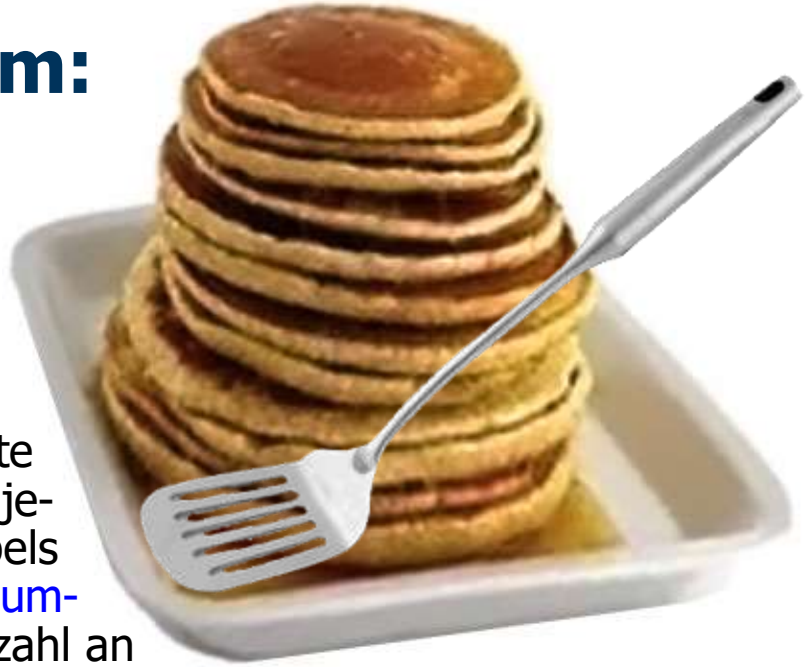






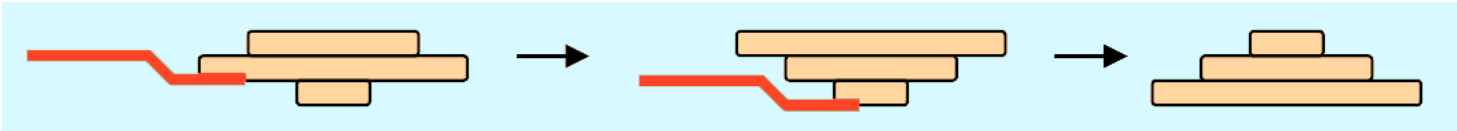
http://sortvis.org/index.html

Und noch ein Sortierproblem: „Pancake sorting“



[Wikipedia:] Beim Pfannkuchen-Sortierproblem geht es anschaulich darum, einen Stapel Pfannkuchen der Grösse nach zu sortieren. Der Stapel soll so sortiert werden, dass am Ende der grösste Pfannkuchen an unterster Stelle ist, der zweitgrösste darüber bis zum kleinsten ganz oben. Dazu darf in jedem Schritt ein von der aktuellen Spitze des Stapels ausgehender Teilstapel ausgewählt und als ganzer umgekehrt werden. Gefragt ist nach der kleinsten Anzahl an Schritten dieser Art, die benötigt werden, um unabhängig von der Ausgangslage den gesamten Stapel zu sortieren. Veröffentlicht wurde das Problem 1975 von Jacob Goodman unter dem Pseudonym Harry Dweighter (reimt mit harried waiter, „gestresster Kellner“):

The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are n pancakes, what is the maximum number of flips (as a function of n) that I will ever have to use to rearrange them? [American Mathematical Monthly 82 (1), 1975, pp. 1010]



Bsp. einer Folge von zwei Flips

Pancake sorting mit linearem Aufwand?

Man weiss, dass die Zahl notwendiger Flips für einen Stapel von n Pfannkuchen zwischen 15/14 n (≈ 1.07 n) und 18/11 n (≈ 1.64 n) liegt. (Heisst das, man kann mittels der mächtigen Flip-Operation in linearer Zeit, also O(n), sortieren?)

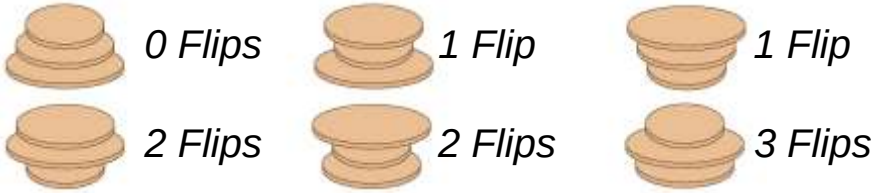
Die britische Zeitung *The Guardian* brachte am 14. Nov. 2013 einen Artikel von Simon Singh zur Geschichte des Pfannkuchen-Sortierproblems; nachfolgend einige Auszüge:

In around 1975, Jacob E. Goodman, a mathematician at the City College of New York, was at home folding towels for his wife. The final pile was somewhat messy, so he decided to restack the folded towels in order of size, smallest folded towel at the top, biggest at the bottom. The problem was that there was no room for a second pile, so he was forced to flip over the top few towels, reassess the situation, then flip over a few more from the top, and so on.

He recalls how a curious problem crossed his mind: "How many flips would I need in the worst case? I thought it was interesting enough to send to the American Mathematical Monthly, but a more 'natural' setting seemed to be pancakes." Thus the so-called pancake sorting problem was born. How many flips are required to turn a disordered stack of pancakes into an ordered stack?

Goodman was still building his reputation as a mathematician and did not want others to think that he was only interested in trivial pancake puzzles, so he adopted a false identity: "It was easy enough to come up with the jocular pseudonym Harry Dweighter ('harried waiter'), but what if the American Mathematical Monthly wanted to contact me in connection with publishing the problem? I told the secretaries at the department that any calls for Harry Dweighter should be diverted to me."

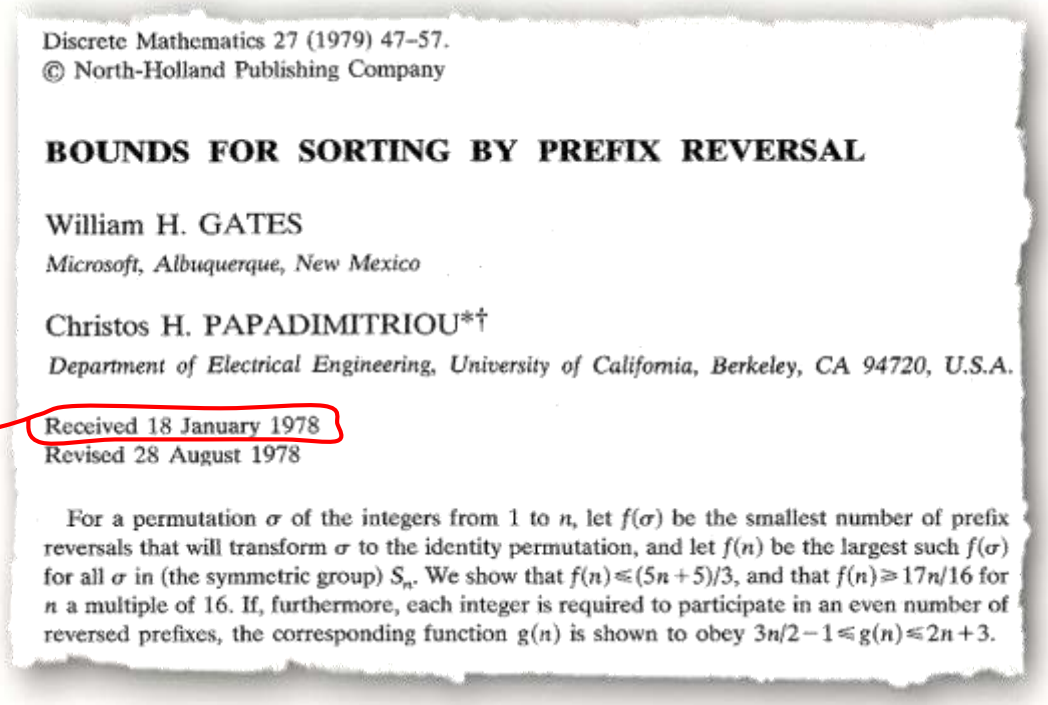
Pancake sorting und Bill Gates



Bei einem Dreierstapel schafft man es stets mit 3 oder weniger Flips, fl(3) = 3. Mit wachsender Stapelgrösse wird das Problem schwieriger, da es mehr Konfigurationen und Flip-Möglichkeiten gibt. Man kennt fl(17) = 19; fl(18) = 20; fl(19) = 22; die genauen Werte ab fl(20) sind aber unbekannt.

Interessanterweise veröffentlichte Bill Gates zusammen mit dem bekannten theoretischen Informatiker Christos Papadimitriou ein wissenschaftliches Paper zu diesem Aspekt [William Gates, Christos Papadimitriou: Bounds for Sorting by Prefix Reversal. Discrete Mathematics, 27 (1) 47-57, 1979]. Die Autoren zeigen, dass (5n+5)/3 eine obere Schranke für fl(n) darstellt.

Das Polizeifoto („mug-shot") von Bill Gates entstand im Dezember 1977; er nahm es seinerzeit in seinem Porsche mit den Verkehrsregeln nicht so ganz genau...



Discrete Mathematics 27 (1979) 47–57.
© North-Holland Publishing Company

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES
Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n+3$.

Christos Papadimitriou über Bill Gates

Papadimitriou recalled: “When I was an assistant professor at Harvard, Bill was a junior. My girlfriend back then said that I had told her: ‘There’s this undergrad at school who is the smartest person I’ve ever met.’ That semester, Gates was fascinated with a math problem called pancake sorting: How can you sort a list of numbers, say 3-4-2-1-5, by flipping prefixes of the list? You can flip the first two numbers to get 4-3-2-1-5, and the first four to finish it off: 1-2-3-4-5. Just two flips. But for a list of n numbers, nobody knew how to do it with fewer than 2n flips. Bill came to me with an idea for doing it with only 1.67n flips. We proved his algorithm correct, and we proved a lower bound—it cannot be done faster than 1.06n flips. […]. It was a silly problem back then, but it became important, because human chromosomes mutate this way. 2 years later, I called to tell him our paper had been accepted to a fine math journal. He sounded eminently disinterested. He had moved to Albuquerque to run a small company writing code for microprocessors, of all things. I remember thinking: ‘Such a brilliant kid. What a waste.’”

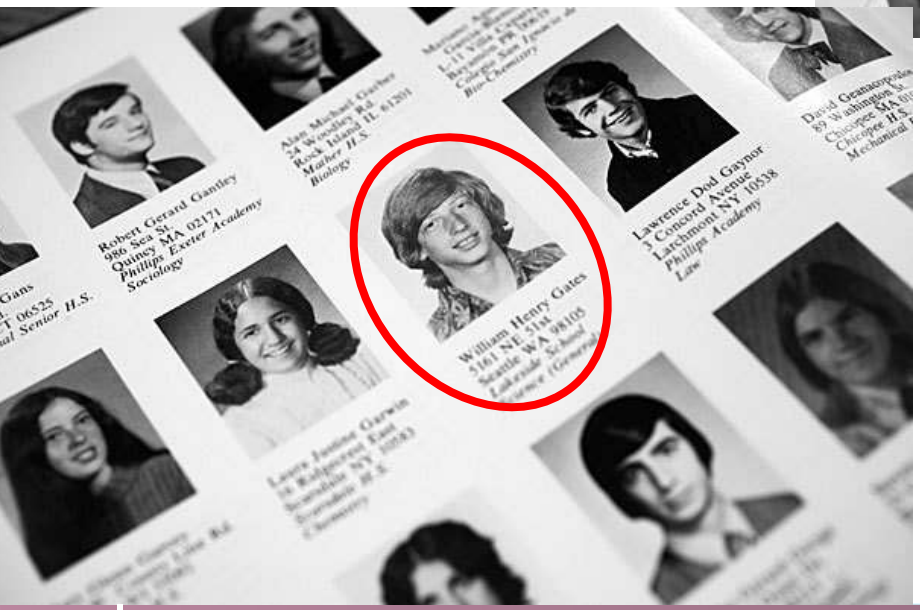
Image source: An Introduction to Bioinformatics Algorithms (Neil Jones, Pavel Pevzner, 2004), p. 130

www.businessinsider.com/a-story-about-bill-gatess-intelligence-2015-11

Young Bill Gates

Bill Gates (13, standing) and Paul Allen (15) connect to a PDP-10 computer at the University of Washington, through a teletype terminal at Lakeside School in 1968. Bill Gates' interest in computers was sparked when he was able to program an early computer to play tic-tac-toe with a human. When he and three others found a way to change the computer's programming so they could get free time on it, they were banned for the summer from school. However, they returned, and were then hired to write first a payroll program on the computer, and then one for scheduling classes. Bill modified the scheduling program so that he was placed in classes with "an inordinate number of interesting girls."

Bild: http://i.imgur.com/EaYMabg.jpg



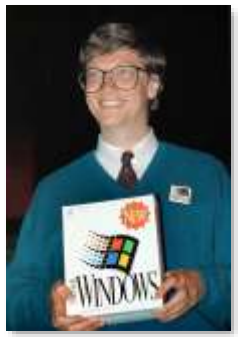
Quelle: http://triogenius.blogspot.com/2014/09/tic-tac-toe-pancakes-and-malaria-bill.html

Links: Bill Gates in seinem "freshman year-book"; er war von 1973 bis 1975 in Harvard. 1975 gründete er mit Paul Allen Microsoft. Dazu mehr bei
→ news.harvard.edu/gazette/story/2013/09/dawn-of-a-revolution/

Bill Gates... und Steve Jobs

https://app.wizer.me/preview/4BJXR

Steve Jobs (li.), der Gründer von Apple, zusammen mit Bill Gates (re.) im Jahr 1991. Beide sind 1955 geboren (Steve Jobs starb 2011), beide haben das Studium abgebrochen und beide „stahlen“ die Idee einer mausgesteuerten und fensterbasierten graphischen Benutzungsoberfläche von der Computer-Workstation Alto der Firma Xerox. Jobs und Gates wurden Rivalen, dominierten mit ihren schnell wachsenden Firmen aber teilweise unterschiedliche Marktsegmente und respektierten sich gegenseitig in späteren Jahren.



Rechts: Ca. 1984.

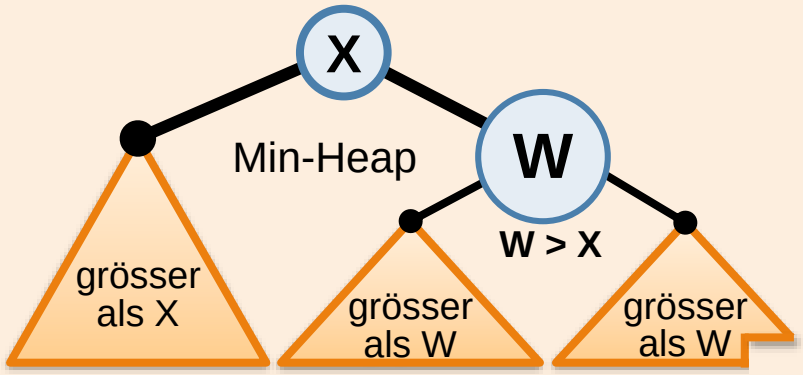
Links: Microsoft dominiert den Markt für PC-Betriebssysteme, Apple ist bei mobilen Geräten erfolgreicher.

www.epmconsult.it/wordpress/2017/06

http://my-musiclist.com/genius-rivals.html

Resümee des Kapitels

- Heap-Datenstruktur
  - (Fast) vollständiger Binärbaum
  - Werte auf jedem Ast sind sortiert
  - *insert*; *get_min* jeweils in O(log n)
  - Besonders effiziente Operationen bei niveauweiser Speicherung als Array
  - Java-Implementierung von *insert* und *get_min*



- Heapsort
  - O(n log n) Zeitaufwand (selbst im worst case!)
  - „In place“ im Array