Beweis (Forts.):

Regeln für Phase 1:

Seien a und b die Elemente, die im Knoten x verglichen werden.

1.1: Falls $a \in A$ und $b \in A$, behalte x in T_A bei.

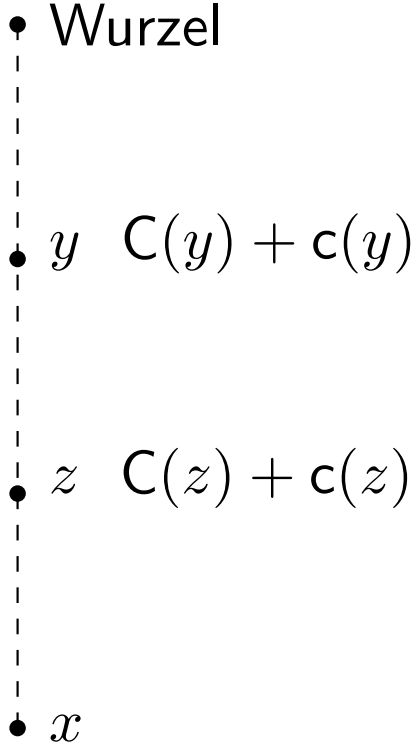
1.2: Falls $a \in \bar{A}$ und $b \in \bar{A}$, behalte x in T_A bei.

1.3: Sei nun o.B.d.A. $a \in A$ und $b \in \bar{A}$. Ersetze den Unterbaum in T mit Wurzel x mit dem Unterbaum, dessen Wurzel das Kind von x ist, das dem Ergebnis entspricht (d.h. lasse den Vergleich $a < b$ aus, da gemäß Strategie alle Elemente aus A kleiner als alle Elemente in $\bar{A}$ sind).

Phase 1 läuft, solange $|C(x)| \geq r$. Ein Knoten auf einem Pfad in T von der Wurzel, bei dem $|C(x)|$ erstmals $= r$ wird, heißt kritisch. Jeder Pfad in T von der Wurzel zu einem Blatt enthält genau einen kritischen Knoten.

Beweis (Forts.):

Betrachte in T_A einen Pfad von der Wurzel zu einem kritischen Knoten x. Sei y ein Knoten auf diesem Pfad, z sein Kind. Es gilt:



$$|C(z)| + |c(z)| \geq |C(y)| + |c(y)| - 1$$

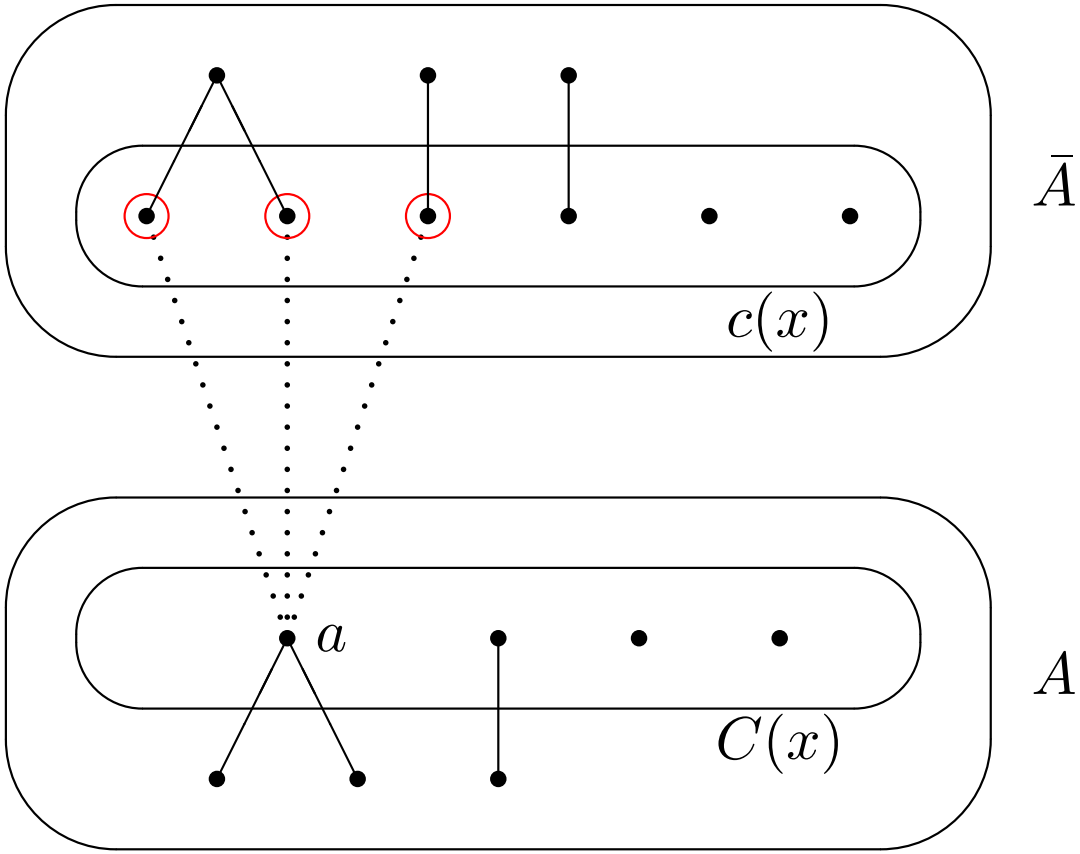
Da $|C(\text{Wurzel})| = |A| = i$ und $|c(\text{Wurzel})| = |\bar{A}| = n - i$, müssen überhalb eines jeden kritischen Knoten x mindestens

$$i - |C(x)| + n - i - |c(x)| = n - r - |c(x)|$$

Vergleiche erfolgen. Von jedem kritischen Knoten abwärts arbeitet der Gegenspieler nach einer Strategie für Phase 2. Sei x ein solcher kritischer Knoten. Dann ist $|C(x)| = r$.

Beweis (Forts.):

Phase 2: Sei $a \in C(x)$ ein Element mit minimalem $s(x, a)$.



$\#\odot = \mathsf{s}(x, a)$
für $a \in A$ fest

Beweis (Forts.):

Fall 1: $s(x,a) \geq s$. Betrachte irgendeinen Pfad von der Wurzel durch x zu einem Blatt w. Jeder solche Pfad muss mindestens $n-1$ Vergleiche enthalten, um answer(w) zu verifizieren: $\geq n-i$, für die answer(w) sich (direkt oder indirekt) als das kleinere Element ergibt, und $\geq i-1$, wo es sich als das größere ergibt. Damit sind $\geq (r-1)s$ Vergleiche redundant (nämlich alle die, die zwischen Elementen aus $C(x) \setminus \{\text{answer}(w)\}$ und Elementen in $\bar{A}$ erfolgt sind). Also:

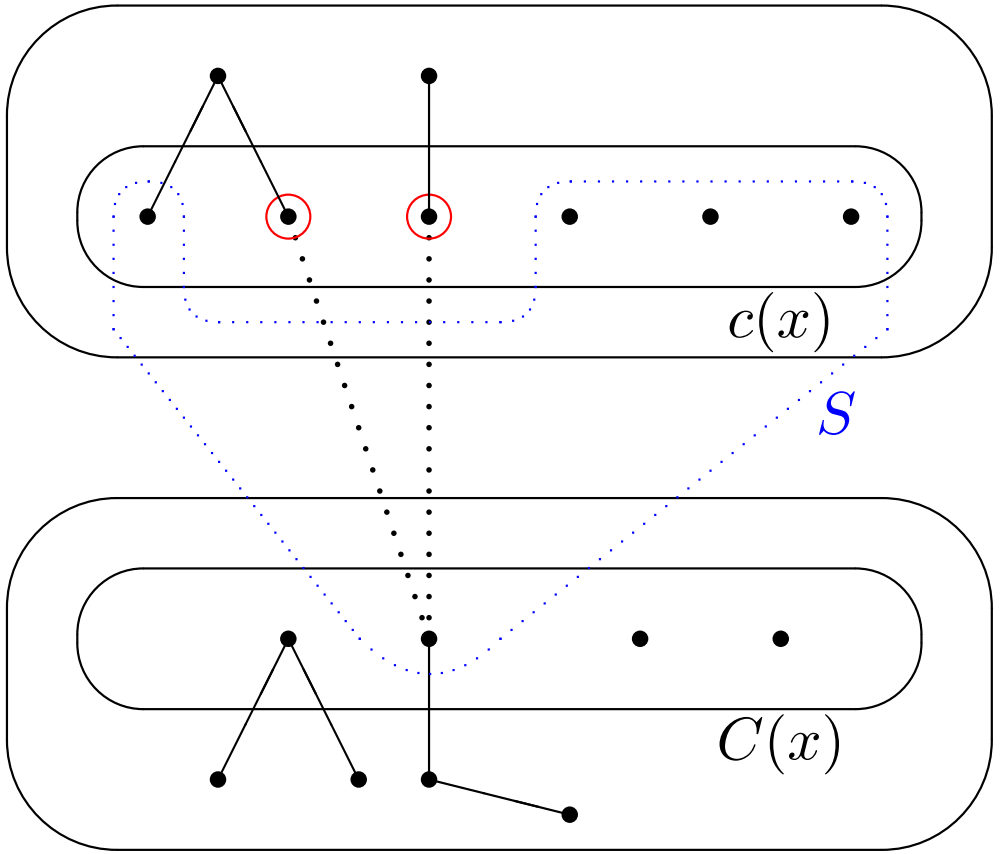
$$\begin{aligned}
\text{Höhe}(T) &\geq n-1+s(r-1) = n-r-s-1+(s+1)r \\
&= n-r-s+2+\log\left[\frac{\binom{n}{i}}{n-i+1}\right] \\
&> \log\left[\binom{n}{i}\frac{2^{n-p}}{n-i+1}\right]
\end{aligned}$$

In diesem Fall folgt also die Behauptung direkt!

Beweis (Forts.):

Fall 2: $s(x,a) < s$.

Fall 2.1: $|c(x)| \geq p$.



$\#\odot = \mathsf{s}(x,a)$
für $a \in A$ fest

Beweis (Forts.):

Sei $S := c(x) \cup \{a\}\backslash$ die Elemente, die in s(x, a) gezählt werden. Der Gegenspieler antwortet nun so, dass das Ergebnis das kleinste Element in S wird. Ist w ein Blatt in T_A unter x, so ist little$(w)=$ $A - \{a\}$. Der Entscheidungsbaum T wird also gemäß folgender Regeln gestutzt (sei y der aktuelle Knoten und seien a und b die beiden Elemente, die in y verglichen werden):

2.1: falls $a, b \in S$, dann bleibt y erhalten

2.2: andernfalls sei o.B.d.A. $a \in A \setminus S$ oder $b \in \bar{A} \setminus S$; ersetze y mit seinem Unterbaum durch das Kind von y und dessen Unterbaum, das der Antwort auf $a < b$ entspricht.

Da überhalb des kritischen Knoten x keine zwei Elemente in S verglichen wurden, muss der Unterbaum von T_A unterhalb von x Tiefe $\geq |S| - 1$ haben.

Beweis (Forts.):

Zusammen mit Phase 1 ergibt sich eine Tiefe von T_A:

$$\begin{aligned}
&\geq n - r - |c(x)| + |S| - 1 \\
&\geq n - r - |c(x)| + |c(x)| + 1 - (s-1) - 1 \\
&= n - r - s + 1 \\
&= n - p
\end{aligned}$$

Beweis (Forts.):

Fall 2.2: $|c(x)| < p$. Sei $S := C(x)$.
Die Regeln für Phase 2 sind in diesem Fall so, dass der Algorithmus als Antwort das Maximum von S bestimmt. Damit ergibt sich für die Tiefe von T_A:

$$\begin{aligned} &\geq n - r - |c(x)| + |S| - 1 \\ &\geq n - r - (p-1) + r - 1 \\ &= n - p\,. \end{aligned}$$

Beweis (Forts.):

Insgesamt ergibt sich also: Jeder Pfad in T_A von x zu einem Blatt hat mindestens die Länge $n - p$. Also enthält jedes T_A mindestens 2^{n-p} Blätter (von T).

Alle T_A's zusammen enthalten $\geq \binom{n}{i} 2^{n-p}$ Blätter von T, wobei jedes Blatt von T höchstens $n - i + 1$ mal vorkommt: Sei w Blatt von T, dann ist little(w) eindeutig bestimmt und es muss, falls T_A das Blatt w enthält, little$(w) \subseteq A$ sein.

Für das Element in $A \setminus$ little(w) gibt es $\leq n - i + 1$ Möglichkeiten. Damit ist die Anzahl der Blätter von $T \geq \frac{1}{n-i+1} \binom{n}{i} 2^{n-p}$ und

$$\text{Höhe}(T) \geq \log \left[\binom{n}{i} \frac{2^{n-p}}{n-i+1} \right] .$$

□

Literatur

L. Hyafil:
Bounds for selection
SIAM J. Comput. **5**, pp. 109–114 (1976)

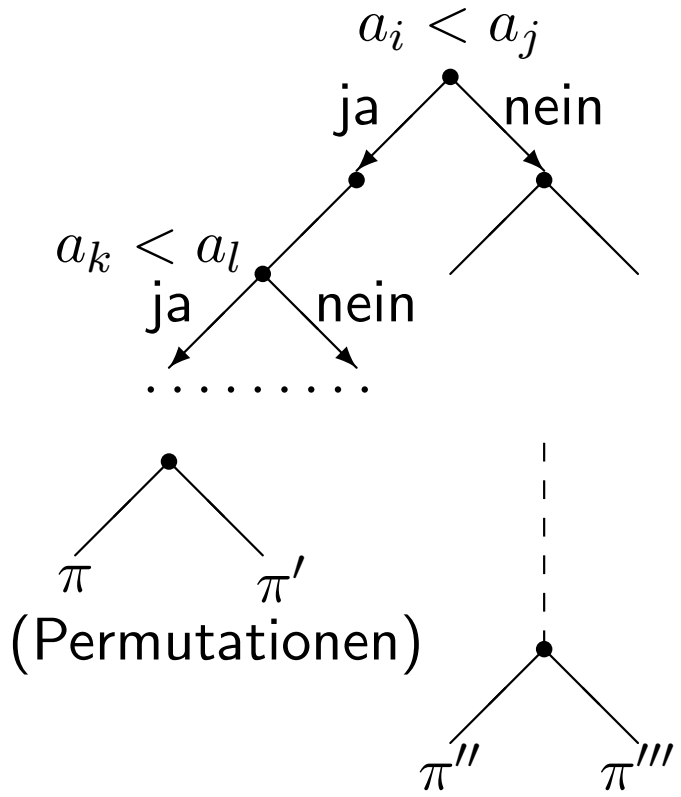
Sam Bent, John W. John:
Finding the median requires $2n$ comparisons
Proc. 17th Annual ACM Symposium on Theory of Computing, pp. 213–216 (1985)

John Welliaveetil John:
A new lower bound for the set-partitioning problem
SIAM J. Comput. **17**, pp. 640–647 (1988)

Dorit Dor, Uri Zwick:
Selecting the median
SIAM J. Comput. **28**(5), pp. 1722–1758 (1999)

7. Untere Schranke für (vergleichsbasiertes) Sortieren

Gegeben n Schlüssel, Queries $a_i \overset{?}{<} a_j$. Dies entspricht einem Entscheidungsbaum:



Beobachtung: Jede Permutation $\pi \in S_n$ kommt in mindestens einem Blatt des Entscheidungsbaums vor. Da $|S_n| = n!$, folgt, dass die Tiefe des Entscheidungsbaums

$$\geq \log_2 n! \,.$$

Stirling'sche Approximation: $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

Also

$$\begin{aligned}\text{Tiefe } \geq \log_2\left[\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right] &= n\cdot\log_2 n - n\cdot\log_2 e + \frac{1}{2}\log\left(2\pi n\right)\\ &= n\log_2 n - \mathcal{O}(n)\,.\end{aligned}$$

Satz 90

Jeder vergleichsbasierte Sortieralgorithmus benötigt für das Sortieren von n Schlüsseln mindestens

$$n \log_2 n - \mathcal{O}(n)$$

Vergleiche.

8. Bucketsort im Schnitt

Gegeben seien n zufällig und gleichverteilt gewählte Schlüssel im Intervall $[0, 1]$. Um diese zu sortieren:

1. Initialisiere Buckets $b_0, b_1, \ldots, b_n$
2. Lege a_i in Bucket $\lceil n \cdot a_i \rceil$
3. Sortiere Schlüssel innerhalb eines jeden Buckets
4. Konkateniere die Buckets

Satz 91

Wird zum Sortieren ein Sortieralgorithmus mit einer Zeitkomplexität von $\mathcal{O}(n^2)$ verwendet, dann hat obiger Bucketsort-Algorithmus im Durchschnitt eine lineare Laufzeit.

Beweis:

Sei n_i die Anzahl der Elemente im Bucket b_i (nach Schritt 2). Die Schritte 1,2 und 4 benötigen zusammen Zeit $\mathcal{O}(n)$. Die Zeit für Schritt 3 beträgt:

$$\mathcal{O}(\sum_{i=0}^{n} n_i^2)\,.$$

Die Erwartungswert für $\sum n_i^2$ lässt sich folgendermaßen abschätzen:

$$\mathbb{E}\left[\sum_{i=0}^{n} n_i^2\right] \leq c \sum_{0\leq i<j\leq n} \Pr[a_i \text{ und } a_j \text{ enden in gleichen Bucket}]$$

$$= c\left(\sum_{0\leq i<j\leq n} \frac{1}{n+1}\right) = \mathcal{O}(n)\,.$$

□

9. Quicksort

Wie bei vielen anderen Sortierverfahren (Bubblesort, Mergesort, usw.) ist auch bei Quicksort die Aufgabe, die Elemente eines Array $a[1..n]$ zu sortieren.

Quicksort ist ein Divide-and-Conquer-Verfahren.

Divide: Wähle ein Pivot-Element p (z.B. das letzte) und partitioniere $a[l..r]$ gemäß p in zwei Teile $a[l..i-1]$ und $a[i+1..r]$ (durch geeignetes Vertauschen der Elemente), wobei abschließend $a[i] = p$.

Conquer: Sortiere $a[l..i-1]$ und $a[i+1..r]$ rekursiv.

Algorithmus:

proc qs(a, l, r)
  if $l \geq r$ **then return fi**
  #ifdef Variante 2
    vertausche $a[$random$(l, r)]$ mit $a[r]$
  #endif
  $p := a[r]$
  $i := l$; $j := r$
  repeat
    while $i < j$ **and** $a[i] \leq p$ **do** $i++$ **od**
    while $i < j$ **and** $p \leq a[j]$ **do** $j--$ **od**
    if $i < j$ **then** vertausche $a[i]$ und $a[j]$ **fi**
  until $i = j$
  vertausche $a[i]$ und $a[r]$
  qs$(a, l, i-1)$
  qs$(a, i+1, r)$

Bemerkung:
Der oben formulierte Algorithmus benötigt pro Durchlauf für n zu sortierende Schlüssel in der Regel $n+1$ Schlüsselvergleiche. Lediglich wenn die beiden Indizes zusammenfallen (was nur bei einem Pivotduplikat passieren kann), benötigt er nur n Vergleiche. Durch geschicktes Einstreuen von **if**-Abfragen kann man in jedem Fall mit $n-1$ Schlüsselvergleichen auskommen.

Komplexität von Quicksort:

- Best-case-Analyse: Quicksort läuft natürlich am schnellsten, falls die Partitionierung möglichst ausgewogen gelingt, im Idealfall also immer zwei gleichgroße Teilintervalle entstehen, das Pivot-Element ist dann stets der Median.

  Anzahl der Schlüsselvergleiche:

$$\sum_{i=1}^{\log n}(n+1) = \log n(n+1) \approx n \log n$$

- Worst-case-Analyse: Z.B. bei einer aufsteigend sortierten Eingabe.

  Anzahl der Schlüsselvergleiche:

$$\Omega(n^2)$$

- Average-case: Da die Laufzeit von Quicksort sehr stark von den Eingabedaten abhängt, kann man die Frage stellen, wie lange der Algorithmus “im Mittel“ zum Sortieren von n Elementen braucht. Um jedoch überhaupt eine derartige Analyse durchführen zu können, muss man zuerst die genaue Bedeutung von “im Mittel“ festlegen. Eine naheliegende Annahme ist, dass alle möglichen Permutationen der Eingabedaten mit gleicher Wahrscheinlichkeit auftreten.

Satz 92

Die durchschnittliche Anzahl von Schlüsselvergleichen von Quicksort beträgt unter der Annahme, dass alle Permutationen für die Eingabe gleichwahrscheinlich sind, höchstens

$$C_n = 2(n+1)(H_{n+1}-1) \approx 2n\ln n - 0.846n + o(n) \approx 1.386 n \log n$$

wobei $H_n := \sum_{i=1}^{n} i^{-1}$ *die* n*-te Harmonische Zahl ist.*

Beweis:

(Variante mit $n-1$ Vergleiche pro Durchlauf)
Sei C_n die Anzahl der Vergleiche bei einem Array der Länge n.
$C_0 = C_1 = 0$.

$$C_n = n - 1 + \frac{1}{n}\sum_{j=1}^{n}(C_{j-1} + C_{n-j})$$

Beweis (Forts.):

Da

i) (in beiden Varianten) das j-kleinste Element bestimmt wird und

ii) auch für die rekursiven Aufrufe wieder alle Eingabepermutationen gleichwahrscheinlich sind:

$$\Rightarrow \qquad C_n = n - 1 + \frac{2}{n}\sum_{j=0}^{n-1} C_j;$$

$$nC_n = n^2 - n + 2\sum_{j=0}^{n-1} C_j;$$

$$(n-1)C_{n-1} = (n-1)^2 - (n-1) + 2\sum_{j=0}^{n-2} C_j;$$

Beweis (Forts.):

$$
\begin{aligned}
nC_n - (n-1)C_{n-1} &= 2n - 1 - 1 + 2C_{n-1}; \\
nC_n &= 2n - 2 + (n+1)C_{n-1}; \qquad / \frac{1}{n(n+1)} \\
\frac{C_n}{n+1} &= 2\frac{n-1}{n(n+1)} + \frac{C_{n-1}}{n} = \\
&= 2\frac{n-1}{n(n+1)} + 2\frac{n-2}{(n-1)n} + \frac{C_{n-2}}{n-1} = \\
&= 2\frac{n-1}{n(n+1)} + \cdots + \frac{C_2}{3} = \\
&= 2\left[\sum_{j=2}^{n} \frac{j-1}{j(j+1)}\right] =
\end{aligned}
$$

Beweis (Forts.):

$$= 2\left[\sum_{j=2}^{n}\left(\frac{j}{j(j+1)} - \frac{1}{j(j+1)}\right)\right] =$$

$$= 2\left[H_{n+1} - \frac{3}{2} - \sum_{j=2}^{n}\left(\frac{1}{j} - \frac{1}{(j+1)}\right)\right] =$$

$$= 2\left[H_{n+1} - \frac{3}{2} - \left(\frac{1}{2} - \frac{1}{(n+1)}\right)\right] =$$

$$= 2\left[H_{n+1} - 2 + \frac{1}{(n+1)}\right]$$

$\Rightarrow C_n = 2(n+1)\left(H_{n+1} - 2 + \frac{1}{n+1}\right)$;
Mit $H_n \approx \ln n + 0.57721\cdots + o(1)$
$\Rightarrow C_n \approx 2n\ln n - 4n + o(n) \approx 1.386 n \log n$ □