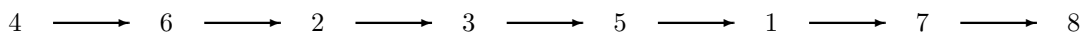
Optimales List ranking

Bis jetzt haben wir einen Algorithmus, der das Problem in $O(\log n)$ Zeit und $O(n \cdot \log n)$ Arbeit mittels Pointer-Jumping löst. Durch iteriertes Färben und Verkürzen der Liste bekommen wir einen arbeitsoptimalen Algorithmus der parallele Zeit $O(\log n \cdot \log\log n)$ und Arbeit $O(n)$ braucht.
Unser Ziel ist es das List–ranking Problem in Zeit $O(\log n)$ und Arbeit $O(n)$ zu lösen.

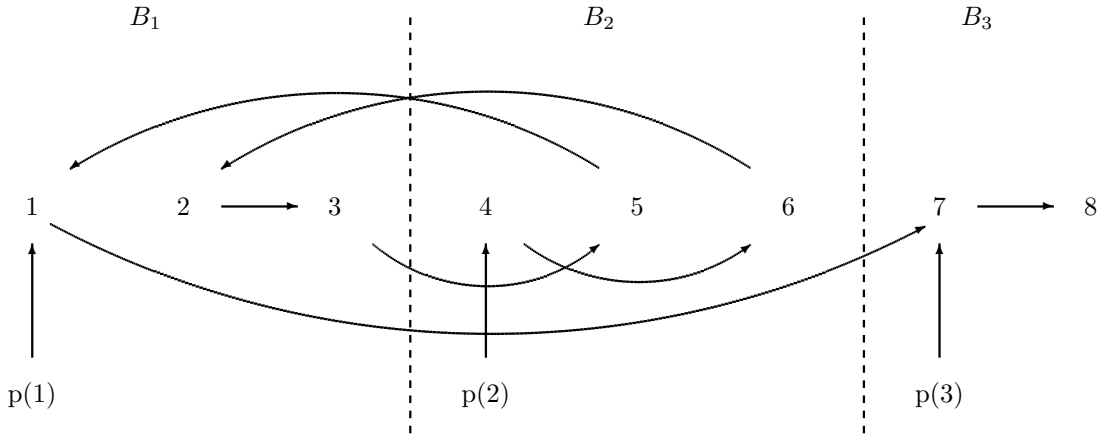
Um den Faktor $\log\log n$ verschwinden zu lassen, müssen wir die Methode zum Schrumpfen der Liste verbessern.
Wir suchen also wie in Algorithmus 3.11 wiederholt unabhängige Mengen von Knoten und entfernen diese, bis die Liste von n Knoten auf $\frac{n}{\log n}$ Knoten geschrumpft ist. Anschließend führen wir Pointer-Jumping aus und fügen die Knoten wieder ein.

Unsere Strategie dazu ist, daß S–Array in $\frac{n}{\log n}$ viele Blöcke B_i der Größe $\log n$ zu teilen. Jedes B_i kann nun sequentiell von einem Prozessor und parallel zu allen anderen verarbeitet werden. Dazu führen wir eine neue Bezeichnung ein:

Bezeichnung: Für jeden Block B_i bezeichnet $p(i)$ den Knoten, der gerade bearbeitet wird. Wir bezeichnen diesen Knoten mit $N(p(i))$. Anfänglich ist $p(i) = (i-1) \cdot \log n + 1$ für $1 \leq i \leq \frac{n}{\log n}$. Die Aufteilung erfolgt in Bezug auf die Speicherreihenfolge, nicht auf die Listenreihenfolge.
Ideal wäre es in jedem Schritt einen Knoten pro Block zu entfernen. Das soll in Zeit $O(1)$ und Arbeit $O(\frac{n}{\log n})$ geschehen, dann sind wir nach $O(\log n)$ Runden fertig.
Dazu ein kleines Beispiel, im folgenden Bild sieht man die Listendarstellung:



Die einzelnen Knoten liegen aber an ganz anderen Stellen im Speicher. Die Speicherreihenfolgen, sowie die Einteilung in Blöcke, sieht man besser in folgendem Bild:



Sie setzen als erstes jedes $N(p(i)) = aktiv$, die restlichen Knoten erhalten den Wert *inaktiv*. Im Beispiel sieht man, daß diese Knoten keine unabhängige Menge bilden, die beiden Knoten $1 \rightarrow 7$ sind beide *aktiv*. Es ergeben sich also 2 Möglichkeiten für *aktive* Knoten:

1. $N(p(i)) = aktiv$ und der Vorgänger sowie der Nachfolger in der Liste sind *inaktiv*. Dann wird der Knoten als *isoliert* bezeichnet und kann direkt entfernt werden. Er ist in einer unabhängigen Menge enthalten.

2. ein *aktiver* Knoten ist Vorgänger oder Nachfolger, dann bilden diese Knoten eine Teilliste. Sie können nicht direkt entfernt werden.

Wie bekommt man nun eine unabhängige Menge so, daß in jedem Schritt möglichst viele Knoten entfernt werden können? Dazu erst einmal einige allgemeine Betrachtungen zum Entfernen:
Wenn wir in jeder Teilliste eine 3–Färbung ermitteln, dann bilden die Farbminima eine unabhängige Menge. Diese können wir löschen. Das Problem dabei ist, die Rechenzeit ist mit $O(\log n)$ viel zu hoch, wir haben nur $O(1)$ Zeit.
Wir können in $O(1)$ eine $\log \log n$ Färbung der Knoten ermitteln. das geht indem wir den einfachen Färbungsalgorithmus zwei mal anwenden. Das Problem hierbei ist, das wir nur einen Teil von $O(\frac{1}{\log \log n})$ Knoten entfernen können (Farbminima).
Wie können wir nun mehr Knoten entfernen? Angenommen wir haben eine unabängige Menge gefunden. Dann bilden die Nachfolger dieser Knoten wieder eine unabhängige Menge und wir können sie danach entfernen. Damit wird in jedem Schritt jeder $(\log \log n) - te$ Knoten ausgeklinkt. Allerdings haben wir nach $\log \log n$ Iterationen keine Nachfolger mehr. Um dieses Problem zu umge-

hen, setzen wir die Technik des *Pipelining* ein. Das funktioniert folgendermaßen:

Knoten haben immer einen der folgenden Zustände:

inaktiv: Knoten, der noch nicht bearbeitet wurde , sind alle Nachfolger von $p(i)$ im Block B_i.

aktiv: Knoten auf dem der Zeiger $p(i)$ gerade steht, vor der weiteren Verarbeitung.

isoliert: ein *aktiver* Knoten wird isoliert, falls weder sein Vorgänger noch sein Nachfolger *aktiv* ist.

removed: Knoten, der erfolgreich entfernt wurde.

ruler: Das lokale Farbminima bei einer $(\log\log n)$–Färbung der Teilliste.

subject: alle anderen aktiven Knoten einer Teilliste.

Der Algorithmus arbeitet informal so: Die Knoten $N(p(i))$ werden auf aktiv gesetzt. Jetzt wird entschieden ob diese Knoten *isoliert* sind oder in einer Teilliste von aktiven Knoten enthalten sind. Teillisten haben maximal die Größe $O(\frac{n}{\log n})$. Ist der Knoten isoliert, wird er entfernt und der Zeiger auf $p(i) = p(i) + 1$ gesetzt. Andernfalls brechen wir die Teilliste in $O(1)$ und linear vielen Operationen auf, indem wir den einfachen Färbungsalgorithmus zweimal, auf die anfängliche Färbung $c(i) = i$, anwenden. Die Färbung kann dann direkt mit in den Algorithmus eingebunden werden, damit vermeidet man, daß die Knoten in einen fortlaufenden Speicherbereich geschrieben werden müssen. Wir erhalten eine $O(\log\log n)$ Färbung. Jedes lokale Minima wird als *ruler* markiert, der Rest wird *subject*.
Die *ruler* teilen diese Liste wieder in einzelne Ketten ein. Aus technischen Gründen wird der erste Knoten einer Teilliste immer zum *ruler* gemacht. Die Kette beginnt also mit einem *ruler*, danach folgen *subject*–Knoten, bis zum nächsten *ruler*. Die Länge der Kette ist nach oben durch $O = (\log\log n)$ beschränkt. Nun werden alle $p(i)$, die auf ein *subjekt* zeigen weitergesetzt $p(i) = p(i) + 1$. Ist $N(p(i)) = ruler$, dann ist der nachfolgende Knoten ein *subject*. Dieses wird vom *ruler* entfernt. War es das letzte *subject* in der Kette, so wird der *ruler* wieder auf *aktiv* gesetzt.

Nun beginnt der Algorithmus von neuem.

Algorithmus 3.16

Eingabe: verkettete Liste, gegeben durch das Successor Array S, die in $O(\frac{n}{\log n})$ viele Blöcke B_i geteilt ist. Pointer $p(i)$ auf den ersten Knoten jedes B_i. $N(p(i)) = aktiv$, alle anderen Knoten sind inaktiv.
Ausgabe: Verkürzte Liste der Länge $O(\frac{n}{\log n})$, damit kann mit Schritt 3 von Algorithmus 3.3 weiter gemacht werden.

begin
for 1 to $O(\log n)$ do
for $1 \leq i \leq \frac{n}{\log n}$ pardo
{1. Subjekt entfernen }
  if $N(p(i)) = ruler$ then
    Entferne den Knoten $S(p(i))$ und Speichere die Informationen
    if $S(p(i))$ war letztes $subject$ der Kette then
      $N(p(i)) = aktiv$;
{2. isolierte Knoten entfernen }
  if $N(p(i)) = aktiv$ und $N(S(p(i))) = N(P(p(i))) = inaktiv$ then
    entferne $p(i)$, speichere seine Knoteninformationen
    $p(i) = p(i) + 1$;
    $N(p(i)) = aktiv$;
{3. bestimmen von $ruler$ und $subjekt$ }
  if $N(p(i)) = aktiv$ then
    $c(i) = i$;
    wende den einfachen Färbungsalgorithmus zwei mal auf alle aktiven Knoten an
    if $c(p(i))$ ist lokales Farbminima und $p(i) \neq$ der erste Knoten der Teilliste und
        $p(i)$ ist nicht der letzte Knoten der Teilliste then
      $N(p(i)) = ruler$;
    else if $p(i)$ ist erster Knoten der Teilliste then
      $N(p(i)) = ruler$;
    else $N(p(i)) = subjekt$;
    if $N(p(i)) = subjekt$ then
      $p(i) = p(i) + 1$
end

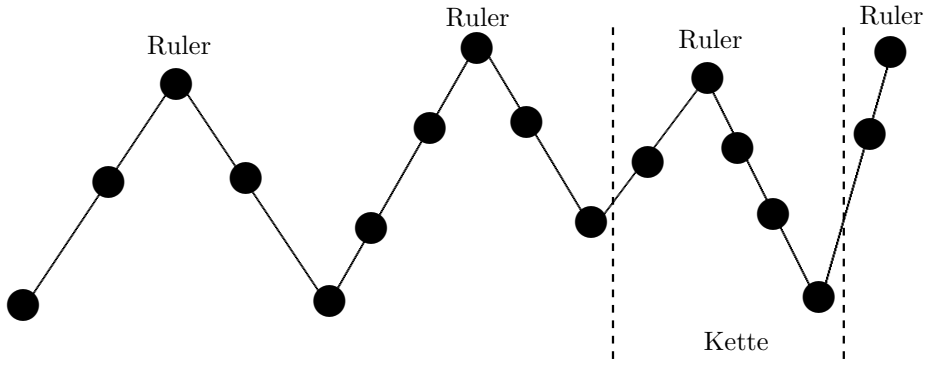
Unser Ziel für die Laufzeitanalyse ist es, zu zeigen, daß Algorithmus 3.16 $O(\log n)$ Zeit und $O(n)$ Arbeit braucht. Dazu analysieren wir zuerst die Zeit und Arbeit einer Runde

Satz: Algorithmus 3.16 verkleinert pro Runde die gegebene Liste durch das Entfernen *isolierter* Knoten und jeweils eines *subjects* pro *ruler*. Aktive Knoten werden also entweder entfernt oder in *subject* und *ruler* gewandelt. Jedes *subjekt* gehört zu genau einem *ruler*. Jedem *ruler* sind maximal $O(\log \log n)$ *subject*–knoten zugeordnet. Das bedeutet, jeder nicht leere Block B_i hat einen Pointer $p(i)$ und $N(p(i))$ ist entweder *aktiv* oder *ruler*. Die Laufzeit der Prozedur für eine Iteration ist $O(1)$ und benötigt $O(\frac{n}{\log n})$ Operationen.

Es Arbeiten in jeder Iteration also nur $O(\frac{n}{\log n})$ Prozessoren. es bleibt zu zeigen, daß nach $O(\log n)$ Iterationen nur noch $O(\frac{n}{\log n})$ in der Liste übrig bleiben. Dann wird $O(\log n) \cdot O(\frac{n}{\log n})$ viel Arbeit ausgeführt, was zu einer Gesamtarbeit von $O(n)$ führt. Problematisch für die Analyse ist, daß nicht alle $p(i)$ in einer Iteration inkrementiert werden und daß nicht sicher ist, ob die $p(i)$ Zeiger nach $O(\log n)$ Runden überhaupt das Ende des Blocks erreichen.
Um zu zeigen, daß die Behauptung trotzdem gilt, müssen wir das Verfahren zur Einteilung von *ruler* und *subject* noch etwas erweitern. Für die Analyse ist wichtig, daß der Index eines *ruler* in seinem Block mindestens so groß wie der seiner *subjecte* ist.
Angenommen wir haben eine Kette der Länge $O(\log \log n)$ mit den Knoten $N(p(i_1)), N(p(i_2)), \cdots N(p(i_k))$. Wir führen eine neue Bezeichnung $\alpha(i_j)$ ein. $\alpha(i_j)$ gibt die Position des Knoten $p(i_j)$ innerhalb des Blocks, zu dem $p(i_j)$ gehört an. Wir ergänzen nun die Menge der *ruler* in einer Teilliste um die aktiven Knoten, die bezüglich ihrer Blockposition $\alpha(i)$ ein lokales Maximum bilden. Dazu müssen wir Teil 3 von Algorithmus 3.16 anpassen:

{3. bestimmen von $ruler$ und $subjekt$ }
if $N(p(i)) = aktiv$ then
$c(i) = i$;
wende den einfachen Färbungsalgorithmus zwei mal auf alle aktiven Knoten an
if $p(i), S(p(i))$ sind aktiv und $P(p(i))$ ist inaktiv then
if $\alpha(p(i)) > \alpha(S(p(i)))$ then {letzter Knoten ist lokales Maximum}
$c(p(i)) = -1$;
else if $p(i), P(p(i))$ aktiv und $S(p(i))$ inaktiv then
if $\alpha(p(i)) > \alpha(P(p(i)))$ then {erster Knoten ist lokales Maximum}
$c(p(i)) = -1$;
else {hier werden nun alle Knoten nochmals bezüglich ihrer Blockposition betrachtet}
if $\alpha(p(i)) \leq min\{\alpha(S(p(i))), \alpha(P(p(i)))\}$ und $\alpha(p(i)) < max\{\alpha(S(p(i))), \alpha(P(p(i)))\}$ then
{p(i) ist ein lokales Minimum und muß damit ein $subject$ werden}
$c(p(i)) = \infty$;
if $\alpha(p(i)) > min\{\alpha(S(p(i))), \alpha(P(p(i)))\}$ und $\alpha(p(i)) \geq max\{\alpha(S(p(i))), \alpha(P(p(i)))\}$ then
{bedeutet p(i) ist ein lokales Maximum und muß damit ein $ruler$ werden}
$c(p(i)) = -1$;
if $c(p(i))$ =lokales Minimum then
$N(p(i)) = ruler$;
else $N(p(i)) = subject$;
if $N(p(i)) = subject$ then
$p(i) = p(i) + 1$;

Auch diese Operationen können in $O(1)$ ausgeführt werden. Jedem $ruler$ werden nun die $subject$-Knoten auf beiden Seiten zugeordnet, solange ihre Blockposition eine monoton fallende Folge bilden. Wird ein Minimum bezüglich der Folge der Blockpositionen erreicht, ordnen wir dieses Subjekt den Ruler auf dessen linker Seite zu. Beide Ketten enden an dieser Stelle.



Da wir die Liste auch rückwärts mittels P-array durchlaufen können um ein Subjekt zu entfernen, bleibt die Zeit zum entfernen in $O(1)$ beschränkt.
Da nun zu jedem $ruler$ die Subjektknoten auf beiden Seiten zugeordnet sind,

und potentielle Ruler, derren Blockposition ein lokales Minimum bilden, zu Subjektknoten werden, kann sich die Kette vergrößern. Da zwischen zwei lokalen Minima ein lokales Maxima liegen muß, bleibt die Größe der Kette weiterhin durch $O(\log \log n)$ beschränkt.
Gesondert müssen hier die *ruler* betrachtet werden, denen von beginn an keine Subjekte zugeordnet sind. Das kann am Rand der Teilliste auftreten, diese Knoten werden wie isolierte Knoten behandelt und auch als solche entfernt.

Da die Knoten in verschiedenen Böcken unterschiedlich schnell verarbeitet werden, müssen wir den Beweis auf Grundlage der Potentialmethode führen. Wir weisen jedem Element ein Gewicht, entsprechend seiner Position im Block, zu.
Jeder Block hat $\log n$ Knoten, dem $i - ten$ Element weisen wir das Gewicht $(1-q)^i$ zu, mit $0 \leq i \leq \log n$ und $q = \frac{1}{\log \log n}$.
Das bedeutet, je weiter der Block verarbeitet ist, desto kleiner wird sein Gewicht. Das Gesamtgewicht eines Blockes ist

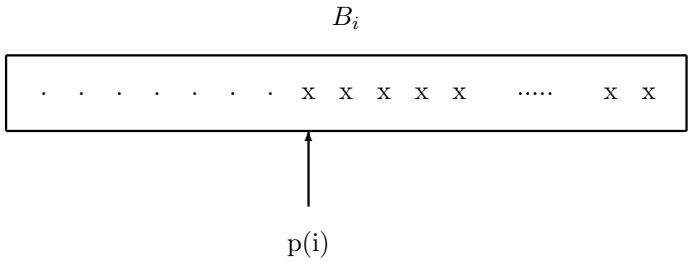
$$\sum_{i=0}^{\log n} (1-q)^i < \frac{1}{q}$$

und das Gesamtgewicht der Liste ist $\leq \frac{n}{q \cdot \log n}$.

Wir müssen zeigen, daß nach jeder Iteration das Gesamtgewicht der Liste um mehr als $1 - (\frac{q}{4})$ sinkt.

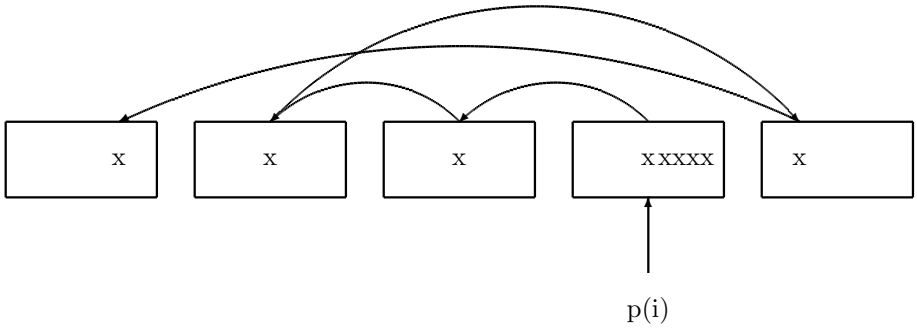
Betrachten wir zunächst das Gewicht der noch zu verarbeitenden Knoten pro Block:

- falls $N(p(i))$ ein *aktiver* Knoten ist, werden die verbeibenden Gewichte gezählt von p(i) bis zum Ende des Blocks.



- falls $N(p(i))$ ein *ruler* ist, dann ist das Gewicht gleich dem Gewicht der Elemente von $p(i)$ bis zum Ende des Blocks + dem Gewicht der Kette, die

zu $N(p(i))$ gehört.



$\rightarrow$ alle x werden zum Gewicht von B_i gezählt.

Satz: In jeder Iteration von Algorithmus 3.16 wird das Gesamtgewicht der Liste mindestens um den Faktor $(1 - \frac{q}{4})$ reduziert.

Der Algorithmus 3.16 ist in 3 Teile eingeteilt, für diese müssen wir im folgenden die Gültigkeit des Satzes einzeln nachweisen:

1. $N(p(i))$ **war** *aktiv* **und wurde als** *isolierter* **Knoten entfernt:**
Angenommen, es ist der $\alpha(i) - te$ Knoten im Block, dann sind alle vorherigen Knoten *subject* und werden damit zu einem anderen Block gezählt oder wurden bereits als *isolierte* Knoten entfernt.
Damit reduziert sich das verbleibende Gewicht des Blocks von

$$\sum_{j=\alpha(i)}^{\log n} (1-q)^j$$

zu

$$\sum_{j=\alpha(i)+1}^{\log n} (1-q)^j < (1-q) \cdot \sum_{j=\alpha(i)}^{\log n} (1-q)^j \leq (1-\frac{q}{4}) \cdot \sum_{j=\alpha(i)}^{\log n} (1-q)^j$$

2. ein *aktiver* **Knoten wird zum** *subject***:**
Damit wird er zur Kette und damit zum Block, dessen *aktiver* Knoten ein *ruler* ist gezählt. Die Kette bildet die Indexfolge $p(i_1), p(i_2), ..., p(i_k)$ und $N(p(i_1))$ ist der *ruler*. Wir nehmen dabei o.b.d.A. an, daß ausgehend vom *ruler* die Gewichte abnehmen. Vor der Zuordnung ist das Gesamtgewicht, der an der

Kette beteiligten Blöcke $B_{i_1}, ..., B_{i_k}$ durch

$$Q = \sum_{j=1}^{k} \sum_{l=\alpha(i_j)}^{\log n} (1-q)^l$$

gegeben. Wenn man die geometrische Reihe analysiert folgt:

$$\sum_{l=\alpha(i_j)}^{\log n} (1-q)^l < \frac{1}{q}(1-q)^{\alpha(i_j)}$$

Da $\alpha(i_1)$ nicht der minimale Index der Folge $\alpha(i_1), \alpha(i_2), ..., \alpha(i_k)$ ist (d.h. $\alpha(i_j)$ hat nicht das größte Gewicht der Kette), gilt

$$Q < \frac{1}{q}\sum_{j=1}^{k}(1-q)^{\alpha(i_j)} \leq \frac{2}{q}\sum_{j=2}^{k}(1-q)^{\alpha(i_j)}$$

und somit

$$\frac{q \cdot Q}{4} \leq \frac{1}{2}\sum_{j=2}^{k}(1-q)^{\alpha(i_j)}.$$

Für das Gewicht der Blöcke nach dem Schritt ist folgendes zu beachten:
Wir gewichten die *subject* Knoten so, daß die Summe dieser gegenüber der der ursprünglichen Knoten halbiert wird. Ein *subject* an Position $\alpha(i_j)$ trägt das Gewicht $\frac{(1-q)^{\alpha(i_j)}}{2}$ zur Summe bei, die verbleibende Hälfte wird verrechnet wenn der Knoten entfernt wird. Diese Veränderung des Gewichtes ist notwendig, da es vorkommen kann, daß in einigen Runden gar keine Knoten gelöscht werden. Das ist genau dann der Fall wenn alle $p(i)$ zu einer Teilliste gehören, in dem Schritt wird nur die Einteilung in *subject* und *ruler* vorgenommen. Damit ist das Gewicht der Blöcke $B_{i_1}, ..., B_{i_k}$ nach dem Schritt:

$$Q - \frac{1}{2}\sum_{j=2}^{k}(1-q)^{\alpha(i_j)} \leq Q - \frac{q \cdot Q}{4} = Q \cdot (1 - \frac{q}{4}).$$

3. der aktuelle Knoten ist ein *ruler* und löscht ein *subject*:
Angenommen der *ruler* hat das Gewicht $(1-q)^{\alpha(i_1)}$, die *subject* Knoten der Kette haben die Gewichte $(1-q)^{\alpha(i_2)}, (1-q)^{\alpha(i_3)}, ..., (1-q)^{\alpha(i_k)}$. Da wir annehmen, daß die Gewichte der *subject*–Knoten absteigend geordnet sind, wird in jedem Schritt das *subject* mit dem größten Gewicht entfernt.
Das Gesamtgewicht von Block und Kette vor dem Entfernen ist somit

$$Q = \sum_{l=\alpha(i_1)}^{\log n} (1-q)^l + \frac{1}{2}\sum_{j=2}^{k}(1-q)^{\alpha(i_j)}$$

Nach dem Entfernen erhalten wir:

$$Q' = Q - \frac{1}{2}(1-q)^{\alpha(i_2)}$$

Aus der Betrachtung der geometrischen Reihe folgt

$$\sum_{l=\alpha(i_1)}^{\log n} (1-q)^l < \frac{1}{q}(1-q)^{\alpha(i_1)} \leq \frac{1}{q}(1-q)^{\alpha(i_2)}$$

Da $(1-q)^{\alpha(i_2)}$ das größte Gewicht ist folgt:

$$\frac{1}{2}\sum_{j=2}^{k}(1-q)^{\alpha(i_j)} \leq \frac{k}{2}(1-q)^{\alpha(i_2)}$$

Desweiteren ist die Kette durch $\log\log n = \frac{1}{q}$ nach oben beschränkt, dadurch erhalten wir

$$\frac{1}{2}\sum_{j=2}^{k}(1-q)^{\alpha(i_j)} \leq \frac{k}{2}(1-q)^{\alpha(i_2)}$$

damit ist

$$Q \leq \frac{3}{2\cdot q}(1-q)^{\alpha(i_2)} = \frac{1}{q}(1-q)^{\alpha(i_2)} + \frac{1}{2\cdot q}(1-q)^{\alpha(i_2)}$$

Für das Gewicht nach der Operation ergibt sich:

$$Q - \frac{1}{2}(1-q)^{\alpha(i_2)} \leq Q - \frac{q\cdot Q}{3} \leq (1-\frac{q}{3})Q < Q(1-\frac{q}{4})$$

□

Satz: Nach $O(\log n)$ Iterationen von Algorithmus 3.16 sind noch $O(\frac{n}{\log n})$ Elemente in der Liste.
Beweis: Das Startgewicht ist durch $\frac{n}{q\log n}$ beschränkt. Nach $5\cdot\log n$ Iterationen ist das verbleibende Gewicht nach oben beschränkt durch

$$\frac{n}{q\log n}\cdot(1-\frac{q}{4})^{5\log n}.$$

Für große n gilt:

$$\frac{n}{q\log n}\cdot(1-\frac{q}{4})^{5\log n} < \frac{n}{\log n}(1-q)^{\log n}$$

Das kleinste Gewicht eines Knotens ist jedoch $\frac{(1-q)^{\log n-1}}{2}$ somit ist die verbleibende Anzahl der Elemente in der Liste durch

$$2\cdot\frac{n}{\log n}\cdot\frac{(1-q)^{\log n}}{(1-q)^{\log n-1}} = 2\cdot\frac{n}{\log n}\cdot(1-q) < 2\cdot\frac{n}{\log n}$$

beschränkt.
Nach dieser Listenverkürzung machen wir zum Bestimmen des Ranges bei Schritt 3 von Algorithmus 3.3 weiter.
Das Listranking kann so in Zeit $O(\log n)$ mit Hilfe von $O(n)$ Operationen auf einer EREW-PRAM ausgeführt werden.