Afg. 3

a) Zur Bestimmung der Minima muss zunächst abgeleitet werden. Statt wie im Skript vorgeschlagen Ketten- und Produktregel zu verwenden wurde an dieser Stelle die Quotientenregel zum ableiten genutzt.

$$f(x) = \frac{x * \ln(n)}{\ln(x)}$$

$$f'(x) = \frac{\ln(n) * \ln(x) - {}^{1}/_{x} * x * \ln(n)}{\ln(x)^{2}}$$

$$= \frac{\ln(n) * \ln(x) - \ln(n)}{\ln(x)^{2}}$$

$$= \frac{\ln(n) * \ln(x)}{\ln(x)^{2}} - \frac{\ln(n)}{\ln(x)^{2}}$$

$$= \frac{\ln(n)}{\ln(x)} - \frac{\ln(n)}{\ln(x)^{2}}$$

Zur Bestimmung der Minima muss die Ableitung nun gegen 0 aufgelöst werden:

$$f'(x) = \frac{\ln(n)}{\ln(x)} - \frac{\ln(n)}{\ln(x)^{2}} = 0 \qquad | * \ln(x)^{2}$$

$$\rightarrow \ln(n) * \ln(x) - \ln(n) = 0 \qquad | + \ln(n)$$

$$\rightarrow \ln(n) * \ln(x) = \ln(n) \qquad | / \ln(n)$$

$$\rightarrow \ln(x) = \frac{\ln(n)}{\ln(n)} = 1$$

$$\rightarrow x = e$$

Um zu beweisen, dass es sich tatsächlich um ein Minimum handelt muss nun der Vorzeichenwechsel überprüft werden:

für alle $x > e$: $\frac{\ln(n)}{\ln(x)} > \frac{\ln(n)}{\ln(x)^{2}}$ und $\frac{\ln(n)}{\ln(x)} - \frac{\ln(n)}{\ln(x)^{2}} > 0$

für alle $x < e$: $\frac{\ln(n)}{\ln(x)} < \frac{\ln(n)}{\ln(x)^{2}}$ und $\frac{\ln(n)}{\ln(x)} - \frac{\ln(n)}{\ln(x)^{2}} < 0$

Die Bedingungen für ein Minimum sind damit erfüllt. Das Minimum der Funktion $f(x) = x * log_{x}(n)$ liegt an der Stelle $(e, e * \ln(n))$.

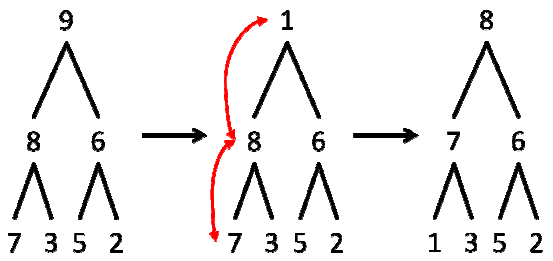
b) Theoretisch wäre e die beste Aufteilungsrate je Knoten. Da ein Knoten aber nur eine ganze Anzahl von Kindern haben kann, müsste die nächstgelegene ganze Zahl gewählt werden: 3. Die worst-case Laufzeit einer Heapify-Operation in einem solchen ternären Heap würde $O\left(3 * log_3\left(10^l\right)\right) \approx (6 * l)$ betragen. Zum Vergleich wäre die Laufzeit bei einem binären Heap: $O\left(2 * log_2\left(10^l\right)\right) \approx (6{,}6 * l)$.

c) In der Praxis wird dennoch hauptsächlich mit binären Bäumen gearbeitet. Ein Grund hierfür könnte z. B. die Tatsache sein, dass bei einer Heapify-Operation im Falle einer verletzten Heap-Eigenschaft und dem damit verbundenem Austausch zunächst das Maximum unter den Kindern des jeweiligen Knotens gefunden werden muss. Bei einem binären Heap wäre dies durch ein einfaches if-/ else-Statments implementierbar ($O(1)$). Sobald $k > 2$ benötigt man jedoch eine Schleife die alle k Elemente überprüft($O(k)$). Dies könnte den geringen Laufzeitvorteil der Heapify-Operation eines ternären Heaps gegenüber eines binären Heaps wieder aufheben.

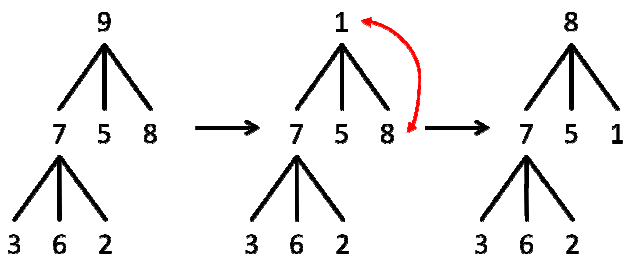
d) In einem beliebigen Max-Heap beträgt die Laufzeit zum Finden des Maximums immer $O(1)$. Grund hierfür ist die angenehme Tatsache, dass das Maximum immer an erster Stelle steht und daher nicht lange gesucht werden muss. Soll dieses Maximum jedoch gegen eine höhere Zahl ausgetauscht werden (durch eine InsertElement-Operation) so wird diese am nächsten freien Knoten angehängt und durch Heapify nach oben durchgereicht (Laufzeit: $O(ld(k))$).

Für die Laufzeit eines Heapifies auf den gesamten Heap würde dies zunächst bedeuten, dass die Suche nach dem jeweiligen Maximum unter den Kindern eines jeden Knotens auf einen Schritt begrenzt wäre (zum Vergleich: Bei einem k-nären Baum müsste jedes Element zunächst überprüft werden was k Schritte beansprucht). Der durch Heapify verursachte Austausch eines Elements hingegen würde extra Arbeit bedeuten, da es nicht nur an den Elternknoten angehängt sondern auch in den binären Heap einsortiert werden müsste (($ld(k)$ Schritte). Da aber $ld(k) < k$ wäre die Laufzeit des Heapifies in einer derartigen Konstruktion insgesamt kürzer.

e)



2 Vertauschungen



1 Vertauschung

f) Die worst-case-Laufzeit für eine Heapify-Operation an der Wurzel eines k-nären Heaps mit n Elementen beträgt $O(k * log_k(n))$. In der Aufgabenstellung wird behaupter, dass für ein beliebiges n gilt:

$\lceil 3 * log_3(n) \rceil \leq \lceil 2 * log_2(n) \rceil$

Induktionsanfang ($n = 1$):

$\lceil 3 * log_3(1) \rceil \leq \lceil 2 * log_2(1) \rceil = 0$

Behelfsformel:

$\lceil y * log_y(x) \rceil = \left\lceil \frac{y * ln(x)}{ln(y)} \right\rceil$

Induktionsschritt ($n = n + 1$):

$\lceil 3 * log_3(n+1) \rceil \leq \lceil 2 * log_2(n+1) \rceil$

$\triangleq \left\lceil \frac{3 * ln(n+1)}{ln(3)} \right\rceil \leq \left\lceil \frac{2 * ln(n+1)}{ln(2)} \right\rceil$

$\triangleq \left\lceil \ln(n+1) * \frac{3}{ln(3)} \right\rceil \leq \left\lceil \ln(n+1) * \frac{2}{ln(2)} \right\rceil = \left\lceil 2 * \frac{\ln(n+1)}{ln(2)} \right\rceil = \lceil 2 * log_2(n+1) \rceil$