Homework — Algorithms and Data Structures

saved at 16:17, Friday 1st December, 2023

Worksheet 5

1. By induction on n, it can be proven that

$$\frac{n}{k} - \frac{k-1}{k} \leq \left\lfloor \frac{n}{k} \right\rfloor.$$

Proof. Base case.

$n \in [1 : k-1]$:

$$\frac{n-k+1}{k} \leq 0 = \left\lfloor \frac{n}{k} \right\rfloor.$$

$n = k$:

$$\frac{n-k+1}{k} = \frac{1}{k} \leq 1 = \left\lfloor \frac{n}{k} \right\rfloor.$$

Induction step.

$n \to n+k$:

$$\begin{aligned}
\frac{(n+k)}{k} - \frac{k-1}{k} \leq \left\lfloor \frac{(n+k)}{k} \right\rfloor &\implies \\
\frac{n}{k} + 1 - \frac{k-1}{k} \leq \left\lfloor \frac{n}{k} + 1 \right\rfloor &\implies \\
\frac{n}{k} - \frac{k-1}{k} + 1 \leq \left\lfloor \frac{n}{k} \right\rfloor + 1 &\implies \\
\frac{n}{k} - \frac{k-1}{k} \leq \left\lfloor \frac{n}{k} \right\rfloor &
\end{aligned}$$

□

2. (1)
```
heapify(A, i):

A(h) = min{ A(i), A(1), A(I) }
h = i: done
h != i: swap(A(i), A(h))
if h leaf { done } else
{ heapify(A, h) }
```

   (2) In exactly the same way as we sorted in ascending order using a max-heap.

   (3) There are a couple possible solutions:

   - Sort in descending order and then reverse which would add $n/2$ operations.
   - Build a reverse heap for which only the choice of indices will change (will introduce more arithmetic operations such as subtraction).

3. (1) Sorting an array of 7 elements can always be done in 14 or less comparissions (via mergesort). Partition will take $n-1$ comparisons. Since we need 14 comparisons for every 7 element, that means that the total comparisons needded just for sorting is $\lfloor n/7 \rfloor \cdot 14$. That combined with $n-1$ gives us $d = \frac{\lfloor n/7 \rfloor \cdot 14}{n} + \frac{n-1}{n}$, $d \to 3$ as $n \to \infty$.

(2)

$$\#U = \left\lfloor \frac{r-1}{2} \right\rfloor \cdot 4 + 3 \geq \frac{r-2}{2} \cdot 4 + 3 = r \cdot 2 - 1 = \left\lfloor \frac{n}{7} \right\rfloor \cdot 2 - 1 \geq \frac{2n-19}{7}$$

so now the formula looks like

$$t(n) = \begin{cases} t\left(\lfloor \frac{n}{7} \rfloor\right) + t\left(\lceil \frac{5n+19}{7} \rceil\right) + 3n & \text{if } n > 60 \\ n \cdot \lceil \log 60 \rceil & \text{if } n. \leq 60 \end{cases}$$

Proof. By induction on n.

Base case: $n \in [1:60]$ we can see that $c = 33$ is satisfactory.

Induction step:

$$\begin{aligned} t(n) &= t\left(\left\lfloor \frac{n}{7} \right\rfloor\right) + t\left(\left\lceil \frac{5n+19}{7} \right\rceil\right) + 3n \\ &\leq c\left\lfloor \frac{n}{7} \right\rfloor + c\left\lceil \frac{5n+19}{7} \right\rceil + 3n \\ &\leq nc \cdot \frac{6}{7} + 3c + 3n \\ &= n \cdot \left(\frac{6c}{7} + 3\right) + 3c \\ &\leq cn \\ \implies 3c &\leq n \cdot \left(\frac{c}{7} - 3\right) \\ \frac{3c}{n} &\leq \frac{c}{7} - 3 \\ \frac{21c}{n} &\leq c - 21 \\ \frac{21}{60}c &\leq c - 21 \quad (n > 60) \\ 21 &\leq \frac{39}{60}c \\ \frac{420}{13} \approx 32.3 &\leq c \end{aligned}$$

□

4. (1) Sorting an array of 3 elements can always be done in 3 or less comparissions (via mergesort). Partition will take $n-1$ comparisons. Since we need 3 comparisons for every 3 element, that means that the total comparisons needded just for sorting is $\lfloor n/3 \rfloor \cdot 3$. That combined with $n-1$ gives us $d = \frac{\lfloor n/3 \rfloor \cdot 3}{n} + \frac{n-1}{n}$, $d \to 2$ as $n \to \infty$.

(2)

$$\#U = \left\lfloor \frac{r-1}{2} \right\rfloor \cdot 2 + 1 \geq \frac{r-2}{2} \cdot 2 + 1 = r - 1 = \left\lfloor \frac{n}{3} \right\rfloor - 1 \geq \frac{n-5}{3}$$

so now the formula looks like

$$t(n) = \begin{cases} t\left(\lfloor \frac{n}{3} \rfloor\right) + t\left(\lceil \frac{2n+5}{3} \rceil\right) + 2n & \text{if } n > 60 \\ n \cdot \lceil \log 60 \rceil & \text{if } n. \leq 60 \end{cases}$$

Proof. By induction on n.

Base case: $n \in [1:60]$ we can see that $c = 6$ is satisfactory.

Induction step:

$$\begin{aligned} t(n) &= t\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + t\left(\left\lceil \frac{2n+5}{3} \right\rceil\right) + 2n \\ &\leq c\left\lfloor \frac{n}{3} \right\rfloor + c\left\lceil \frac{2n+5}{3} \right\rceil + 2n \\ &\leq nc + 2c + 2n \\ &\not\leq cn \end{aligned}$$

we can see that there would exist no c that could satisfy the inequality — it is no longer linear. □

5. (a) the number of leaves for a tree with n nodes is equal to $\lceil \frac{n}{2} \rceil$. This is made clear by the fact that appending a leaf to a heap with odd size never increases the count of leaves, whereas if the size of the heap is even, the count of leaves will increase by one. The base case is the heap of size 0 with 0 leaves.

   (b) the only subtrees with height 0 are the leaves.

   (c) deleting every leaf of a tree decreases it's depth by one.

   (d) if T is a heap, let T' be the same tree but with the leaves removed. The number of subtrees of T with height h is equal to the number of subtrees of T' with height $h-1$.

Now we can write a recursive formula for the number of subtrees with height h

$$s_n(h) = \begin{cases} s_{\lfloor n/2 \rfloor}(h-1) & \text{if } h > 1 \\ \lceil \frac{n}{2} \rceil & \text{if } h = 0 \end{cases}$$

Proof. Hypothesis – a heap with n nodes has at most $\lceil n/2^{h+1} \rceil$ subtrees of height h

By induction on h. Base case. $h = 0$

$$s_n(0) = \left\lceil \frac{n}{2} \right\rceil \leq \left\lceil \frac{n}{2^{0+1}} \right\rceil$$

Induction step. $h - 1 \to h$

$$\begin{aligned} s_n(h) &= s_{\lfloor n/2 \rfloor}(h-1) \\ &\leq \left\lceil \frac{\lfloor n/2 \rfloor}{2^{(h-1)+1}} \right\rceil \\ &\leq \left\lceil \frac{n/2}{2^{(h-1)+1}} \right\rceil \\ &= \left\lceil \frac{n}{2^{h+1}} \right\rceil \end{aligned}$$

□