Sample Solution

Assignment 7

CSCI 3110 — Fall 2018

We use the following adaptation of Quick Sort. Let p be the pivot chosen to partition the input. We partition S into three sets: $L := \{x \in S \mid x < p\}$, $M := \{x \in S \mid x = p\}$, and $R := \{x \in S \mid x > p\}$. If H_S denotes the set of heavy hitters of S, then

$$H_S = \begin{cases} H_L \cup H_R & |M| < k \\ H_L \cup H_R \cup \{p\} & |M| \geq k \end{cases}.$$

We can find H_L and H_R by recursively calling the algorithm on L and R. The size of M can of course be determined in linear time. Moreover, if we choose the pivot p to be the median of S, which we can do in linear time using the linear-time selection algorithm, then $|L| \leq |S|/2$ and $|R| \leq |S|/2$. So the cost of the case when we do make recursive calls is $T(n) \leq 2T(n/2) + O(n)$. What's the base case? Well, if $|S| < k$, we can immediately report $H_S = \emptyset$ because there is no element in S that occurs at least k times. The cost of this is in $O(1)$. So this gives the following algorithm:

HeavyHitters(S)

```
if |S| < k
  then return ∅
  else p := FindMedian(S)
       (L, M, R) := Partition(S, p)
       H_L := HeavyHitters(L)
       H_R := HeavyHitters(R)
       if |M| < k
         then return H_L ∪ H_R
         else return H_L ∪ H_R ∪ {p}
```

FindMedian is the standard linear-time selection algorithm. Partition is a straightforward adaptation of the standard two-way partition algorithm, but let's present it here for completeness:

Partition(S, p)

$(L, M, R) := (\emptyset, \emptyset, \emptyset)$
for every $x \in S$
  do if $x < p$ **then** $L := L \cup \{x\}$
    else if $x = p$ **then** $M := M \cup \{x\}$
    else $R := R \cup \{x\}$
return (L, M, R)

As already observed above, FindMedian, Partition, and determining the size of $|M|$ take $O(n)$ time, and $|L| \le |S|/2$ and $|R| \le |S|/2$. So the running time of the algorihm is given by the recurrence

$$T(n) \le \begin{cases} 2T(n/2) + O(n) & n \ge k \\ O(1) & n < k \end{cases},$$

which can be rewritten as

$$T(n) \le \begin{cases} 2T(n/2) + dn & n \ge k \\ d & n < k \end{cases}$$

for an appropriate constant $d > 0$.

This is easily shown to be in $O(n \lg(n/k))$: We claim that $T(n) \le cn \lg(n/k)$ for some $c > 0$.

For $1 \le n < 4k$, we have $T(n) \le cn$, for a large enough constant c. Indeed, if $n < k$, $T(n) \le d \le cn$ for $c \ge d$. If $k \le n < 2k$, the algorithm makes two recursive calls on less than k elements each, so the cost is $T(n) \le dn + 2d \le 3dn \le cn$ for $c \ge 3d$. If $2k \le n < 4k$, the algorithm makes two recursive calls on less than $2k$ elements, so the cost is $T(n) \le dn + 2 \cdot (3dn/2) = 4dn \le cn$ for $c \ge 4d$. Since $\lg(n/k) \ge 1$, we have $cn \le cn \lg(n/k)$, so for $1 \le n < 4k$, $T(n) \le cn \lg(n/k)$.

For $n \ge 4k$, we have

$$\begin{aligned} T(n) &\le 2T\left(\frac{n}{2}\right) + dn \\ &\le 2c\left(\frac{n}{2}\right)\lg\left(\frac{n}{2k}\right) + dn && \text{(by the inductive hypothesis)} \\ &= cn\left(\lg\left(\frac{n}{k}\right) - 1\right) + dn && \left(\text{because } n \ge 4k, \text{ so } \lg\left(\frac{n}{k}\right) \ge 2 \text{ and } \lg\left(\frac{n}{2k}\right) = \lg\left(\frac{n}{k}\right) - 1\right) \\ &\le cn \lg\left(\frac{n}{k}\right) && \text{as long as } c \ge d. \end{aligned}$$