Fundação Getulio Vargas

Estruturas de Dados e Algoritmos **Homework 3**

Prof. Jorge Poco **Due:**: April 7, 2019

Homework 3

Lucas Machado Moschen

1 Red-Black Trees

The code is attached in C++ 11.

2 Radix Sort

The code is attached in Python 3. The code is in a notebook, as requested. I use the bucket sort to help the radix sort.

3 Sorting in Place in Linear Time

(2pts) Suppose that we have an array of n data records to sort and that the key of each record has the value 0 or 1. An algorithm for sorting such a set of records might possess some subset of the following three desirable characteristics:

1. The algorithm runs in $O(n)$ time.
2. The algorithm is stable.
3. The algorithm sorts in place, using no more than a constant amount of storage space in addition to the original array.

(a) Give an algorithm that satisfies criteria 1 and 2 above.

Answer: The algorithm can be the counting sort. The running time $T(n)$ is $O(n + k)$, so it's a linear sorting algorithm. The relative order is preserved, so, it's stable. To make it stable, we must turn the counting array in a cumulative counting array, and after, for each number, we will have the right position, and we decrease the counting to put all the elements.

(b) Give an algorithm that satisfies criteria 1 and 3 above.

Answer: The original partition of the counting sort is Hoare Partition. It takes to indices in the list and exchange values in wrong positions. So we can chance the quick sort in this way to give us an in place algorithm (it only creates a temp variable to exchange two values in partition) and it takes a expected running time $O(n)$. if we partition with the indices p and r, we can verify that the indice that returns is $p \leq i < r$.

(c) Give an algorithm that satisfies criteria 2 and 3 above.

Answer: The insertion sort is a stable sort and it creates only a constant amount of storage in addition to the list. But, the running time is $\Theta(n^2)$.

(d) Can any of your sorting algorithms from parts(a)–(c) be used to sort n records with b-bit keys using radix sort in $O(bn)$ time? Explain how or why not.

Answer: The third one is stable, but the running time is $0(bn^2)$, so, we can't use it. The second one is not stable. Therefore, we can only use the first algorithm, because it sorts the keys stable and it runs in $O(n)$ time. The radix sort runs in $O(b*(n+d)) = O(bn)$.

(e) *Answer:* I will do by this way:

Algorithm 1: CountingSortInPlace(A,k)

```
let $C[0..k]$ the array to count with zeros. **for** $j = 0$ ***to*** $A.length$ **do**
    $C[A[i]] = C[A[i]] + 1$
**end**
temp = 0
**for** $i$ = *0* ***to*** $k$ **do**
    **for** $j = 1$ ***to*** $C[i]$ **do**
        $A[temp] = i$
        $temp+ = 1$
    **end**
**end**
```

This algorithm sorts in $O(n+k)$, because we need to count the n elements and for each key we are putting the number in the correct position. However, It's not guaranteed the stability, because we are not taking the relative postion of each number.

4 Alternative Quicksort Analysis

(2pts) An alternative analysis of the running time of randomized quicksort focuses on the expected running time of each individual recursive call to QUICKSORT, rather than on the number of comparisons performed.

(a) Argue that, given an array of size n, the probability that any particular element is chosen as the pivot is $1/n$. Use this to define indicator random variables $X_i = I\{i\text{th smallest element is chosen as the pivot}\}$. What is $E[X_i]$?

Answer: The output pivot is equally likely in a random choose. So, there is a symmetry among the values in the list. That indicates that the probability of each number be chosen as pivot in a list of n elements is $\frac{1}{n}$. So, if

$$X_i = \begin{cases} 1, & if\ the\ i^{th}\ element\ is\ chosen\ as\ pivot \\ 0, & if\ it\ is\ not \end{cases}$$

We have $E[X_i] = P(X_i) = \frac{1}{n}$, as argued before.

(b) Let $T(n)$ be a random variable denoting the running time of quicksort on an array of size n. Argue that

$$E[T(n)] = E\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right] \tag{1}$$

Answer: Given that we pick a pivot, randomly, suppose we picked the q^{th} smallest element in the list. So $X_i = 1$, if $i = q$, and $X_i = 0$, if $i \neq q$. After, we partition the list in two parts, the left sub list, we will have all the elements smaller then $A[q](A$ is the list). In the right one, we'll have the greater elements. So, there is $q-1$ elements in the left and $n-q$ in the right. Besides that, to take a partition, we pass through all the list(size n), so we take $\Theta(n)$ running time to do this job. So, $T(n) = X_q(T(q-1)+T(n-q)+\Theta(n))$. However, we don't know which q we will pick. To take a "mean", we use the expected value. Because of that, $E[T(n)] = E\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right]$.

(c) Show that equation 1 simplifies to

$$E[T(n)] = \frac{2}{n}\sum_{q=0}^{n-1} E[T(q)] + \Theta(n) \tag{2}$$

Answer:

$$\begin{aligned}
E[T(n)] &= E\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right] \\
&= \sum_{q=1}^{n} E[X_q(T(q-1) + T(n-q) + \Theta(n)]) \\
&= \sum_{q=1}^{n} E[X_q] \cdot E[T(q-1) + T(n-q) + \Theta(n)] \qquad [\textit{independence of the variables}] \\
&= \sum_{q=1}^{n} \frac{1}{n} \cdot (E[T(q-1)] + E[T(n-q)] + E[\Theta(n)]) \qquad [E[\Theta(n)] = \Theta(n),\ \textit{because } n \textit{ is constant}] \\
&= \frac{1}{n}\sum_{q=1}^{n} E[T(q-1)] + \frac{1}{n}\sum_{q=1}^{n} E[T(n-q)] + \frac{1}{n}\sum_{q=1}^{n} \Theta(n) \\
&= \frac{1}{n}\sum_{q-1=0}^{n-1} E[T(q-1)] + \frac{1}{n}\sum_{n-q=0}^{n-1} E[T(n-q)] + \frac{1}{n}\cdot n \cdot \Theta(n) \qquad [\textit{change the variables}] \\
&= \frac{1}{n}\sum_{i=0}^{n-1} E[T(i)] + \frac{1}{n}\sum_{i=0}^{n-1} E[T(i)] + \Theta(n) \qquad [\textit{let's take } q \textit{ instead of } i]
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{n}\sum_{q=0}^{n-1} E[T(q)] + \frac{1}{n}\sum_{q=0}^{n-1} E[T(q)] + \Theta(n) \\
&= \frac{2}{n}\sum_{q=0}^{n-1} E[T(q)] + \Theta(n)
\end{aligned}$$

(d) Show that

$$\sum_{k=1}^{n-1} k \log k \le \frac{1}{2}n^2 \log n - \frac{1}{8}n^2 \tag{3}$$

Answer:

$$\begin{aligned}
\sum_{k=1}^{n-1} k\log k &= \sum_{k=1}^{\lfloor n/2\rfloor} k\log k + \sum_{k=\lfloor n/2\rfloor+1}^{n-1} k\log k \\
&\le \sum_{k=1}^{n/2} k\log k + \sum_{k=n/2+1}^{n-1} k\log k \\
&\le \log(n/2)\sum_{k=1}^{n/2} k + log(n-1)\sum_{k=n/2+1}^{n-1} k \\
&= \log(n/2)\frac{(n/2+1)(n/2)}{2} + \log(n-1)\frac{(n+n/2)(n-n/2-1)}{2} \\
&= \log(n/2)\frac{n^2/4+n/2}{2} + log(n-1)\frac{3n^2/4-3n/2}{2} \\
&= \log(n)\frac{n^2/4+n/2}{2} - \log 2\frac{n^2/4+n/2}{2} + log(n-1)\frac{3n^2/4-3n/2}{2} \\
&\le \log(n)\frac{n^2-n}{2} - \frac{n^2/4+n/2}{2} \\
&= \frac{1}{2}n^2\log(n) - \frac{1}{8}n^2 - \frac{n}{2}(\log(n)+\frac{1}{2}) \\
&\le \frac{1}{2}n^2\log(n) - \frac{1}{8}n^2
\end{aligned}$$

(e) Using the bound from equation 3, show that the recurrence in equation 2 has the solution $E[T(n)] = \Theta(n\lg n)$. (Hint: Show, by substitution, that $E[T(n)] \le an\log n - bn$ for some positive constants a and b.)

Answer: Let's remember that $E[T(n)] = \frac{2}{n}\sum_{q=0}^{n-1} E[T(q)] + \Theta(n)$. Suppose, for the substitution method that: $E[T(n)] \le \begin{cases} an_0\log n_0 - bn_0 & if\ n = n_0 \\ an\log n - bn & if\ n > n_0 \end{cases}$. I'll prove it by induction.

Base case: $n_0 = 2$, $E[T(2)] = \frac{2}{2}\sum_{q=0}^{1} E[T(q)] + \Theta(2) = 0 \le 2a - 2b = 2(a-b)$

Inductive Step: Suppose it works for $n_0 < n < k$, let's prove it works for $n = k$.

$$E[T(k)] = \frac{2}{k}\sum_{q=0}^{k-1} E[T(q)] + \Theta(k) \le \frac{2}{k}\sum_{q=1}^{k-1}(aq\log q - bq) + \Theta(k) \le \frac{2a}{k}(\frac{1}{2}k^2\log k - \frac{1}{8}k^2) - \frac{2b}{k}(\frac{(k-1)(k)}{2}) + \Theta(k)$$

$$\le ak\log k - \frac{a}{4}k - b(k-1) + \Theta(k) = ak\log k - (\frac{a}{4}+b)k + b + ck$$

$$\le ak\log k - bk, \quad \textit{since } a > b \textit{ and } b > c$$

So, it's proved by induction that $E[T(n)] \le an\log n - bn$. It implies that $E[T(n)] = O(n\log n)$ To prove $E[T(n)] = \Omega(n\log n)$, take $c > \frac{a}{4} + b$. So, $E[T(n)] = \Theta(n\log n)$.