Linear Sorting

August 6, 2014

1 Introduction

Although, we have been analyzing sorting based in comparing numbers (trichotomy law: $x < y$, $y < x$ or $x = y$), there are ways to improve the lower bound of $\Omega(n \log n)$.

2 Counting Sort

In this algorithm, we use the idea of accumulating the number of values before certain value. This is basically what we do when calculating the accumulative function in probability. For example, given the sequence 5,3,5,4,2,1,1, we can do the following.

1. Count the elements equal to i. In our case, we have 2 one's, 1 two's, 1 three's, 1 four's and 2 five's.
2. Then, we have 2 numbers less or equal to one, 3 numbers less or equal to two, etc
3. Then, we have, for example, 2 numbers less or equal to one, then you can put a one in the second position. Now, you have only 1 number less or equal to one...

The final algorithm is in the slides, and it has a complexity of $O(n + k)$ in the worst case scenario. If $k = O(n)$, then counting sort has the complexity $O(n)$.

Note

An important property of this algorithm is its stability: "numbers with the same value appear in the output array in the same order as they do in the input array."

3 Radix Sort

This is the sorting algorithm used by the 1900's IBM Tabulating machines. You can understand the algorithm if you think that any number has the following representation:

$$p = p_d 10^d + p_{d-1} 10^{d-1} + ... + p_1 10^1 + p_0 10^0 \tag{1}$$

Then, you can sort the n $d-$digit numbers if you start sorting the least siginificative digit of the n numbers to the most significative digit of the n numbers. The following lemma allows to prove the complexity of the algorithm.

Lemma 8.3

Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $\Theta(d(n+k))$ time if the stable sort it uses takes $\Theta(n+k)$ time.

Proof:

The proof follows from the induction starting at the least significative digit to the most significative digit. After all, the sorting based on the digit position will leave sorted numbers from less to more when you only take the numbers from the i digit position to the 1 digit position. Now, to calculate the complexity: When each digit is in the range 0 to $k-1$, and k is not too large, we use counting sort for the sorting based in the digits at position i. Therefore, when each sorting pass takes $\Theta(n+k)$ and there are d passes, we have that the complexity of radix sort is $\Theta(d(n+k))$.

Note What happens when $k = O(n)$?

Lemma 8.4

Given n b-bit numbers and any positive integer $r \leq b$, RADIX-SORT correctly sorts these numbers in $\Theta(\frac{b}{r}(n+2^r))$ time if the stable sort it uses takes $\Theta(n+k)$ time for inputs in the range 0 to k.

Proof:

For a value $r \leq b$, each key has $d = \left\lceil \frac{b}{r} \right\rceil$ digits of r bits. Therefore, each digit is in the range of 0 to $2^r - 1$, the classic trick is to use $k = 2^r - 1$. Then, each pass of the counting sort takes $O(n + 2^r)$, and there are d passes. Then, we have the following complexity $\Theta\left(d(n+2^r)\right) = \Theta\left(\frac{b}{r}(n+2^r)\right)$.

Note For given values of n and b, we want to find what values of r are convenient under the constraint $r \leq b$.

Case I if $b < \lfloor \log n \rfloor$, then if we choose $r = b$ then $\frac{b}{b}\left(n + 2^b\right) = \Theta(n)$.

Case II if $b \geq \lfloor \log n \rfloor$, then if we choose $r > \lfloor \log n \rfloor$, we have that the running time is $\Omega\left(\frac{b}{r}(n+2^r)\right)$. However, if we choose $r \leq \lfloor \log n \rfloor$, we have that the running time is $\Theta\left(\frac{b}{r}(n)\right)$. Thus, the perfect choice for this case is $r = \lfloor \log n \rfloor$, then we have a running time of $\Theta\left(\frac{bn}{\log n}\right)$.

4 Bucket Sort Analysis

Note: Even with numbers in a range between $[0, N)$, it is possible to use the following trick: Divide the numbers by N. This will send all the numbers to the interval $[0, 1)$ and to the base case of the algorithm.

To make the complexity analysis, we take into account the following random variable

- n_i denotes the number of elements placed in bucket $B[i]$.

Then, we have the following recursion:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \tag{2}$$

And again we use the expected value:

$$E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O(E([n_i^2]) \tag{3}$$

We need to prove that $E[n_i^2] = 2 - \frac{1}{n}$ for $i = 0, 1, 2, ..., n-1$ (Because the input comes from a uniform distribution). For this, we define:

- $X_{ij} = I\{A[j] \text{ falls in bucket } i\}$ for $i = 1, 2, ..., n-1$ and $j = 1, 2, ..., n$.

Then, $n_i = \sum_{j=1}^{n} X_{ij}$. Thus,

$$\begin{aligned} E[n_i^2] &= E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] \\ &= E\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\ &= E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\leq j\leq n}\ \sum_{1\leq k\leq n, k\neq j} X_{ij}X_{ik}\right] \\ &= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1\leq j\leq n}\ \sum_{1\leq k\leq n, k\neq j} E[X_{ij}X_{ik}] \end{aligned}$$

But, we know that X_{ij} has two values 1 or 0. Therefore, we have that

$$\begin{aligned} E[X_{ij}^2] &= 1^2 \times \frac{1}{n} + 0^2 \times \left(1 - \frac{1}{n}\right) = \frac{1}{n} \\ E[X_{ij}X_{ik}] &= E[X_{ij}]E[X_{ik}] = \frac{1}{n} \times \frac{1}{n} \end{aligned}$$

Which allows us to obtain the final equation:

$$\begin{aligned}
E[n_i^2] &= \sum_{j=1}^{n} \frac{1}{n} + \sum_{1 \le j \le n} \sum_{1 \le k \le n, k \neq j} \frac{1}{n^2} \\
&= n \times \frac{1}{n} + n(n-1)\frac{1}{n^2} \\
&= 1 + \frac{n-1}{n} \\
&= 2 - \frac{1}{n}
\end{aligned}$$

Then,

$$E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} \left[2 - \frac{1}{n}\right] = \Theta(n) + nO\left(2 - \frac{1}{n}\right) = \Theta\left(n\right) \tag{4}$$