Review Lesson 1: Design and Analysis of Computer Algorithms

Yanhui Li

April 4, 2018

Problem 1

You are given a collection of n bolts of different widths and n corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger or smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut. Design an algorithm for this problem with average case efficiency in $\Theta(n \log n)$.

Solution

Let w_i and v_i be the widths of the i-th bolt and i-th nut. which have the same length

- Choose a bolts randomly. We assume that it is the x-th bolts with width w_x, and compare it with all n nuts (n comparisons). Now the n nuts are divided into three classes:
  - $N_< = \{i | v_i < w_x\}$ 1 bolts → n nuts
  - $N_= = \{i | v_i = w_x\}$

    Note that $|N_=| = 1$, and we assume it is the y-th nut with $v_y = w_x$.
  - $N_> = \{i | v_i > w_x\}$
- Compare the y-th nut with all other $n - 1$ bolts, and divided them into two classes ($n - 1$ comparisons):
  - $B_< = \{i | w_i < v_y\}$ y-th nut → (n-1) bolts

$N_< \mid N_= \mid N_>$

$B_< \mid B_= \mid B_>$

- $B_< = \{i | w_i > v_y\}$

- Note that for any nut in $N_<(N_>)$, its corresponding bolt is in $B_<$ $(B_>)$. $|N_>| = |B_>|, |N_<| = |B_<|, |N_>| + |N_<| = |B_>| + |B_<| = n - 1$.

- Now, after $2n - 1$ comparisons, we divide the total problem of size n into two sub-problems of smaller sizes $|N_>|$ and $|N_<|$ (Recurrence):
  - match each bolt in $B_<$ to its nut in $N_<$;
  - match each bolt in $B_>$ to its nut in $N_>$;

- We can solve this problem by above recursive steps. And the recursive function is given as follows:

$$T(n) = 2n - 1 + T(|N_>|) + T(|N_<|)$$

For the random choosing, $|N_>|$ could be $0, 1, \ldots, n - 1$, and $|N_<|$ $n - 1, n - 2, \ldots, 0$ correspondingly, with the probability $1/n$. Therefore we can get the recurrence function of $T(n)$:

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

From

$$\sum_{i=0}^{n-1} T(i) = \sum_{i=0}^{n-1} T(n - 1 - i)$$

we can easily get that

$$T(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + 2n - 1$$

By induction on n, we give the Inductive Proof of $A(n) \in O(n \ln n)$:

- Base case $(n = 1)$ is trivial.

- Inductive assumption: $A(i) \leq c \times i \ln i$ for $1 \leq i < n$

- Inductive proof:

$$\begin{aligned} T(n) &= \frac{2}{n}\sum_{i=0}^{n-1} T(i) + 2n - 1 \\ &\leq \frac{2}{n}\sum_{i=0}^{n-1} c \times i \ln i + 2n - 1 \end{aligned}$$

  Note that

$$\begin{aligned} \frac{2}{n}\sum_{i=0}^{n-1} c \times i \ln i &\leq \frac{2c}{n}\int_{1}^{n} x \ln x dx \\ &\approx \frac{2c}{n}\left(\frac{n^2 \ln n}{2} - \frac{n^2}{4}\right) \\ &= cn \ln n - \frac{cn}{2} \end{aligned}$$

?

Therefore,

$$T(n) \leq cn \ln n + n(2 - \frac{c}{2}) - 1$$

Let $c = 4$, we have $A(n) \leq 4n \ln n$.

The proof of $T(n) \in \Omega(n \ln n)$ is similar. Here we give the conclusion:

$$T(n) \in \Theta(n \ln n) = \Theta(n \log n)$$

Problem 2

Given a list E (unsorted) of n elements, for any element x, define $freq(x)$ as the number of occurrence of x in E. Design an algorithm to find the element t with $freq(t) > n/2$ in $O(n)$ time. (If not existing, output -1).

Solution The solution will be a single number or N/A

Given a list E (unsorted) of n elements, we construct the algorithm to find the element t with $freq(t) > n/2$ in $O(n)$ time by the following three steps:

- Search: find the $\lceil n/2 \rceil$-th largest number y in E (in $O(n)$ time).

- Analyze: if there exists an element x in E with $freq(t) > n/2$, x should be equal to y. pretty tricky

  Proof. If not, x is larger or less than y.

  In **larger** case, note that y is $\lceil n/2 \rceil$-th largest, so only $\lceil n/2 \rceil - 1$ elements are larger than y in E. Obviously, $freq(x) \leq \lceil n/2 \rceil - 1 < n/2$. A contradiction occurs.

  In **less** case, only $n - \lceil n/2 \rceil$ elements are less than y in E. And $freq(x) \leq n - \lceil n/2 \rceil \leq n/2$. A contradiction also occurs.

- Count: compute the number of occurrence of y in E (in $O(n)$ time).

$O(n) + O(n) = O(n)$

Problem 3

An array is bitonic if it is comprised of an increasing sequence of integers followed immediately by a decreasing sequence of integers. Write a program that, given a bitonic array of n distinct int values, determines whether a given integer c is in the array. Your program should use $O(\lg n)$ compares in the worst case. (20)

Solution

Use a version of binary search to find the maximum of this array in $\lg n$ compares; then use binary search to search in two pieces separately.

```
function INT BINARY-SEARCH-FINDMAX(Array A, int first, int last )
    if first < last then
        int middle = (first + last)/2;
        if A[middle] > A[middle + 1] then
            return BINARY-SEARCH-FINDMAX(A, first, middle);
        else
            return BINARY-SEARCH-FINDMAX(A, middle + 1, last);
        end if
    else
        return first
    end if
end function
```

Problem 4

Given an array of n elements in which each element is an integer between 1 and n, write an algorithm to determine if there are any duplicates in linear $O(n)$ time and $O(1)$ extra space. (20)

Solution

```
function BOOLEAN FIND-DUPLICATES(int Array A[1...n])
    for i = 1; i ≤ n; i++ do
        while A[i] ≠ i do
            if A[A[i]] = A[i] then
                return True;
            else
                int k;
                k = A[A[i]];  O(1) extra space
                A[A[i]] = A[i];
                A[i] = k;
            end if
        end while
    end for
    return False;
end function
```