Algorithm Design & Analysis (CSE222)

Lecture-4

Recap

- Testing Bipartiteness using BFS

- Counting Inversions

Outline

- Counting Inversions
- Select k Smallest Elements

Count Inversions in Sorted Arrays

Input: Two Sorted arrays

Output: Count Inversions

MERGE-COUNT(L, R)

Initialize $pt_\ell = 1; pt_r = 1; i = 1$ and $Count = 0$
Initialize an empty array A of size $|L| + |R|$.
while $(pt_\ell \leq |L|$ and $pt_r \leq |R|)$
    if $(L[pt_\ell] \leq R[pt_r]]$
        $A[i] = L[pt_\ell]$ and $pt_\ell = pt_\ell + 1$
    else
        $A[i] = R[pt_r]; pt_r = pt_r + 1$ and $Count = Count + |L| - pt_\ell + 1$
    $i = i + 1$
if $(pt_\ell > |L|)$
    Append remaining elements of R into A
if $(pt_r > |R|)$
    Append remaining elements of L into A
Return $(Count, A)$

Running Time?

Counting Inversion Using DC

Input: Unsorted array
Output: Count Inversions

COUNT-INVERSION(A)

if $|A| = 1$
else
  $m = \left\lceil \frac{|A|}{2} \right\rceil$
  $A_{Left} = A[1, \ldots, m]$ and $A_{Right} = A[m+1, \ldots, |A|]$
  $(C_{Left}, A_{Left}) =$ COUNT-INVERSION(A_{Left})
  $(C_{Right}, A_{Right}) =$ COUNT-INVERSION(A_{Right})
  $(C_{Split}, A) =$ MERGE-COUNT(A_{Left}, A_{Right})
  $Count = C_{Left} + C_{Right} + C_{Split}$
Return $(Count, A)$

Is it correct? Lets analyze!

Analysis of Count-Inversion(A)

1 3 4 | 2 5 7

Proof by induction
Base: k = 0. So, n = 2^k = 1. No inversion.
Induction Hypothesis: The algorithm will return correct count for all k = 1, 2, ..., i, i.e., until n = 2^i.

Let n = 2^{i+1}. It has two sorted arrays and calls Merge-Count($\boldsymbol{A_{Left}}, \boldsymbol{A_{Right}}$)
Why count is right?

(3,2), (4,2)

$|A_{Left}| - Pt_L + 1$

1

Count = $C_{Left} + C_{Right} + C_{split}$

Running Time

Recurrence: $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn$

L_1 $\quad 2T(n/2) \quad cn$

L_2 $\quad 2T(n/4) \quad \boxed{2 \cdot \frac{n}{2}}$

$\vdots$ $\quad 2T(n/4) \quad \boxed{2 \cdot (2 \cdot n/4)}$

$L_{\log n}$

n

$= cn\log n$

Outline

- Counting Inversions

- Select k Smallest Elements

k^{th} Smallest Element

Running time for the following problems

- Select smallest element in array of size n. (Can we get o(n) or $O(n^{1-c})$?)
- Select $n/2^{th}$ smallest element in an array of size n.
- Select k^{th} smallest element in an array of size n.

Input: Array of size n
Output: k^{th} smallest element.

1. Sort the input array in ascending order.
2. Return the value at k^{th} index.

Can we do better?

Not interested in the ranking of all the elements.

Only k^{th} smallest element!

QuickSort

Input: Array of size n
Output: Sorted Array

```
QUICKSORT(A, p, r)
if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q - 1)
        QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Partition Function

2	8	7	1	3	5	6	4

2	1	3	4	7	5	6	8

Running time?

k^{th} Smallest Element

Based on the ***pivot*** element, partition the array into two subarrays
Lesser (All elements smaller than the pivot).
Greater (All elements bigger than the pivot).

Let the ***pivot*** position returned by Partition() is *i*.

- If $i = k$ then return the pivot element.
- If $i > k$ then go to *Lesser* and look for k^{th} smallest element.
- Else, go to *Greater* and look for $(k - |Lesser| - 1)^{th}$ smallest element.

Analyze

Divide & Conquer algorithm:
Every Partition() takes O(n).

Worst case could be $O(n^2)$.

Best case could be O(n). When?

1	2	3	4	5	6	7	8

$$T(n) = T(n-1) + cn$$

$$T(n) = T(n/2) + cn$$

Randomized Algorithm

Input: Array A of size n, integer k.
Output: k^{th} smallest element in A.

RandQuickSelect(A,k)

1. Pick a random pivot element p from A.
2. Split A into subarrays *Lesser* & *Greater*. Set L = |*Lesser*|
3. If $L = k - 1$ then return p.
4. If $L > k - 1$ then return RandQuickSelect(*Lesser*, k).
5. If $L < k - 1$ then return RandQuickSelect(*Greater*, $k - L - 1$)

Claim: The expected running time of RandQuickSelect() is O(n).

Randomized Algorithm

Input: Array *A* of size n, integer *k*.
Output: k^{th} smallest element in *A*.

$\textsc{RandQuickSelect}(A, k)$

if $(|A| == 1)$
    Return A
$p = \textsc{ChoosePivot}(A)$
$Lesser = \{A[i] : A[i] < p\}$; $Greater = \{A[i] : A[i] > p\}$
if $(|Lesser| == k - 1)$
    Return p
if $(|Lesser| > k - 1)$
    Return $\textsc{RandQuickSelect}(Lesser, k)$
else
    Return $\textsc{RandQuickSelect}(Greater, k - |Lesser| - 1)$

Claim: The expected running time of RandQuickSelect() is O(n).

Analyze

$T(n) = cn + \frac{1}{n}\sum_{i=1}^{n-1} T(i) \le cn + \sum_{i=n/2}^{n-1} T(i)\cdot\frac{2}{n}$

$cn + \frac{2}{n}\cdot c\left(\frac{n}{2}\left(\frac{n}{2}+1\right)/2 + \frac{n}{2}\cdot\frac{n}{2}\right) = cn + c\frac{n}{4} + \frac{cn}{2}$

$= O(n)$

The expected running time of RandQuickSelect() is O(n).

The exact running time depends on partition sizes due to the random pivot.

Expected running time is the average of all possible partition sizes.
(0 & n-1) or (1 & n-2) or (2 & n-3) or … or (n-1 & 0).

The Partition() takes c·n time. So recurrence of expected running time is,

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

(0 & n-1) or (n-1 & 0)

$cn + \frac{2}{n}\left(c\frac{n}{2} + c\left(\frac{n}{2}+1\right) + \dots + c(n-1)\right)$

$\le cn + \frac{2}{n}c\left(\frac{n}{2}+0 + \frac{n}{2}+1 + \dots + n-1 + \frac{n}{2} + \frac{n}{2}\right)$

Analyze

$$
\begin{aligned}
T(n) &\leq c \cdot n + \frac{2}{n} \sum_{i=\lceil n/2 \rceil}^{n-1} T(i) \\
&\leq c \cdot n + \frac{2}{n} \left(T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + \ldots + T(n)\right) \\
&= c \cdot n + \frac{2c}{n} \left(\lceil n/2 \rceil + \lceil n/2 \rceil + 1 + \ldots + n\right) \\
&= c \cdot n + \frac{2c}{n} \left(n/2 \cdot n/2 + 1 + 2 + \ldots + n/2\right) \\
&= c \cdot n + \frac{2c}{n} \left(\frac{n^2}{4} + \frac{n/2(n/2+1)}{2}\right) \leq 3c \cdot n = O(n)
\end{aligned}
$$

Analyze

The expected running time of RandQuickSelect() is O(n).

The exact running time depends on partition sizes due to the random pivot.

Expected running time is the average of all possible partition sizes.
(0 & n-1) or (1 & n-2) or (2 & n-3) or … or (n-1 & 0).

The Partition() takes c·n time. So recurrence of expected running

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

Can we improve the worst case running time?

Reference

Slides

Algorithms Design by Kleinberg & Tardos - Chp 5.3