CS 240 Module 3

Graham Cooper

May 4th, 2015

Selection

Given an array A[$a...n-1$] and $0 \leq k \leq n-1$ return the kth largest element in A.

0.1 1) Selection-sort Idea:

Scan A k times, deleting max each time.
Cost: $\Theta(kn)$

0.2 2)

Sort A, return A[n-k]
Cost: $\Theta(nlogn)$

0.3 3)

Scan the array once, and keep k largest seen so far in the min-heap.
Cost: $\Theta(nlogk)$
Eg: [6,5,3,8,7,4], k =3
We put in 6, 5 then 3 into the min heap. After we look at the rest of the elements and keep the min heap the size of k and add new elements if an element in the array is larger than the root of the min-heap. Continue through the array and at the end pick the root of the min heap.

0.4 4)

Heapify(A) then call deleteMax k times.
Cost: $\Theta(n + klogn)$ For median selection (k = n/2) then it is the same as sorting so $\Theta(n)$

Partition Algorithm

Given an array A[0...n-1] and $0 \leq k \leq n-1$, find the element at position k of the sorted A.

<u>**Observation:**</u>
A = $[\overset{0}{7}, \overset{1}{3}, \overset{2}{2}, \overset{3}{4}, \overset{4}{6}, \overset{5}{1}]$
Sorted(A) = [1, 2, 3, 4, 6, 7]
What is the position of A[3](4) in the sorted A. the answer is the number of elements < A[3] in A[0..2] and A[4,5]

<u>**Idea:**</u> choose one element (pivot) and partition the data into: (items < pivot), pivot, (items > pivot). If position(pivot) == k, done, otherwise, continue either on the left or on the right, depending on the position of the pivot.

WHAT WE WANT TO DO:

Implicit A = $[\overset{0}{9}, \overset{1}{4}, \overset{2}{5}, \overset{3}{8}, \overset{4}{6}, \overset{5}{3}, \overset{6}{2}]$

Lets pick A[2] as the pivot, swap A[2] and A[0]
A = [5,4,9,8,6,3,2]

<u>**Idea:**</u> Find the outermost wrongly positioned pair and swap.

advance i, backup j.
A = [5, 4, 9, 8, 6, 3, 2]
$i < j$ so we should swap i
A = [5, 4, 2, 8, 6, 3, 9]
Advance i, backup j
A = [5, 4, 2, 8 ,6 ,3, 9]
$i < j$ swap i
A = [5, 4, 2, 3, 6 ,8 ,9]
advance i, backup j
A = [5, 4, 2, 3, 6, 8 ,9]
$j < i$ stop, swap, A[0] wiht A[j]
A = [3, 4, 2, 5, 6, 8, 9]
Return 3.

Quick Select(A,K)

P = choosePivot(A)
i = partition(P)
if i = k
return A[i]
if i > k:
return QuickSelect(A[0...i-1], k)
if i < k:
return QuickSelect(A[i+1...n-1], k-i-1)

0.4.1 Cost of Quick Select

Let T(n) e cost of QuickSelect
T(n) = $\Theta(n)+$
$\Theta(1)$, if n = k
T(i) if $i > k$
T(n-i-1), if $i < k$

Best Case: T(n) = $\Theta(n)$ if i = k
(first chosen pivot if the element at position k, no recursive calls)

Worst Case: i = 0 or i = n-1
Recursive call has size n - 1
(if we pick the first element as the pivot, then an array sorted in ascending or descending order will give the worst case runtime.)

T(n) =
d if n = 1
T(n-1) + cn if n $\geq$ 2
T(n) = cn + c(n-1) + c(n-2) + ... + c(2) + d
$= c)\frac{n(n+1)}{2} - c + d \in \Theta(n^2)$

What if hte partition is balanced
A[p] is always close to median

T(n) =
T($\frac{n}{2}$) + cn if n $\geq$ 2
d if n = 1

Assume n is a power of 2: 2^x
$T(2^x) = c \cdot 2^x + c \cdot 2^{x-1} + ... + c \cdot 2 + d$
$= c(2^{x+1} - 2) + d$
$= 2c(n-1) + d \in \Theta(n)$

Average-Case analysis: Average cost over all inputs of size n as function of n.
Observation: behaviour of QuickSelect depends on relative ordering, and not on actual values. [1,3,5,7] will yeild the same worst case behaviour as [4,5,6,7].

Assume all keys are unique, $x_1, x_2, ...x_n$ then there are n! possible orderings on these keys. and each ordering is equally likely

After we pick the pivot, what iwll the split look like?

L(num of items)	R(num of items)
0	n-1
1	n-2
...	...
k - 1	n - k
k	n - k - 1
k + 1	n - k - 2
...	...
n - 1	0

For each choce of pivot (n possible pivots) there are (n-1)! permutations of non-pivot elements, each of the splits is equally likely

After Partiction:
A = [0...x...]
Define T(n,k) an average cost for selecting kth item from a size n array.

$$T(n,k) = cn + \frac{1}{n}T(n-1, k-1)x + \frac{1}{n}(n-2, k-2) + ...$$

Put in summation notation.