-7

CS 240 - Data Structures and Data Management **Spring 2017**

Lecture 5-7: May 15 - 25, 2017

Lecturer: Taylor Smith *Notes By: Harsh Mistry*

5.1 Sorting and Randomized Algorithms

Selection vs. Sorting

The selection problem is: Given an array A of n numbers, and 0 k ¡ n, find the element in position k of the sorted array.

Observation : the kth largest element is the element at position n k. Best heap-based algorithm had time cost $\theta(n_k \log n)$
For median selection, k = n , giving cost $\theta(n \log n)$. This is the same cost as our best sorting algorithms.

Crucial Subroutines

quick-select and the related algorithm quick-sort rely on two subroutines:

- choose-pivot(A): Choose an index i such that A[i] will make a good pivot (hopefully near the middle of the order).

```
choose-pivot1(A)
return 0
```

- partition(A, p): Using pivot A[p], rearrange A so that all items ≤ the pivot come first, followed by the pivot, followed by all items greater than the pivot.

```
partition(A, p)
A: array of size n, p: integer s.t. 0 <= p <= n
  swap(A[0], A[p])
  i <- 1, j < - n  1
   loop
    while i < n and A[i] <= A[0] do
      i <- 1 + 1
    while j >= and A[j] > A[0] do
      j <- j -1
    if j < i then break
    else swap(A[i], A[j])
  end loop
  swap(A[0], A[j])
    return j
```

Quick Select Algorithm

```
quick-select1(A, k )
A: array of size n, k: integers.t. 0 < = k< n
  p <-  choose-pivot1(A)
  i <- partition(A, p)
  if i=k then
    return A[i]
   else if i >k then
    return quick-select1(A[0, 1, . . . , i     1], k)
  else if i <k then
    return quick-select1(A[i + 1, i + 2, . . . , n -1 ], k - i -1)
```

Analysis

Worst-case analysis: Recursive call could always have size n-1

$$T(n) = cn + c(n-1) + c(n-2) + \ldots + c \cdot 2 + d \in \theta(n^2)$$

Best-case analysis: First chosen pivot could be the kth element. No recursive calls; total cost is $\theta(n)$

Average case analysis : Assume all n! permutations are likely. Average cost of sum for all permutations, divided by n!
Define T(n, k) as average cost for selecting kth item from size-n array :

$$T(n) = max_{0 \leq k \leq n} T(n, k)$$

The cost is determined by i, the position of the pivot A[0].
For more than half of the n! permutations, $\frac{n}{4} \leq i \leq \frac{3n}{4}$.
In this case, the recursive call will have length at most $\frac{3n}{4}$, for any k. The average cost is then given by:

$$T(n) \leq \begin{cases} cn + \frac{1}{2}(T(n) + T(3n/4)) & n \geq 2 \\ d, & n = 1 \end{cases}$$

Rearranging displays that $T(n) \in O(n)$

5.1.1 Randomized Algorithms

A randomized algorithm is one which relies on some random numbers in addition to the input. The cost will depend on the input and the random numbers used.

Expected Running Time

Define T(I,R) as the running time of the randomized algorithm for a particular input I and the sequence of random numbers R.
The expected running $T^{(exp)}(I)$ of a randomized algorithm for a particular input I is the expected value for T(I,R):

$$T^{(exp)}(I) = E[T(I,R)] = \sum_R T(I,R) \cdot Pr[R]$$

The worst case running time is then

$$T^{(exp)}(n) = \max_{size(i)=n} T^{(exp)}(I)$$

Randomized Quick Select

```
shuffle(A)
A: array of size n
  for i <- 0 to n     2 do
    swap ( A[i], A[i + random(n-i)] )
```

Expected cost becomes the same as the average cost, which is $\theta(n)$