Linear-Time Selection

Randomized Selection (Algorithm)

Design and Analysis of Algorithms I

Prerequisites

Watch this after:

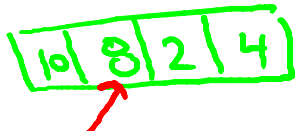
- QuickSort - Partitioning around a pivot
- QuickSort – Choosing a good pivot
- Probability Review, Part I

The Problem

For simplicity

Input : array A with n distinct numbers and a number

Output : i^{th} order statistic (i.e., i^{th} smallest element of input array)



3^{rd} order statistic

Example : median.

(i = (n+1)/2 for n odd,

i = n/2 for n even)

Reduction to Sorting

O(nlog(n)) algorithm

1) Apply MergeSort 2) return i^{th} element of sorted array

Fact : can't sort any faster [see optional video]

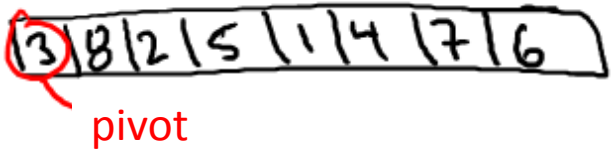
Next : O(n) time (randomized) by modifying Quick Sort.

Optional Video : O(n) time deterministic algorithm.
-- pivot = "median of medians" (warning : not practical)

Partitioning Around a Pivot

Key Idea : partition array around a pivot element.

-Pick element of array



-Rearrange array so that

-Left of pivot => less than pivot

-Right of pivot => greater than pivot

Note : puts pivot in its "rightful position".

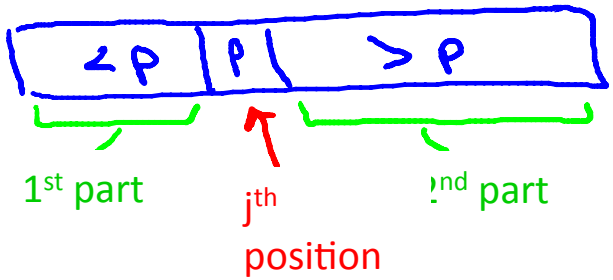
Suppose we are looking for the 5th order statistic in an input array of length 10. We partition the array, and the pivot winds up in the third position of the partitioned array. On which side of the pivot do we recurse, and what order statistic should we look for?

- ○ The 3rd order statistic on the left side of the pivot.
- ○ The 2nd order statistic on the right side of the pivot.
- ○ The 5th order statistic on the right side of the pivot.
- ○ Not enough information to answer question – we might need to recurse on the left or the right side of the pivot.

Randomized Selection

Rselect (array A, length n, order statistic i)

0) if n = 1 return A[1]
1) Choose pivot p from A uniformly at random
2) Partition A around p
   let j = new index of p
3) If j = i, return p
4) If j > i, return Rselect(1st part of A, j-1, i)
5) [if j<i] return Rselect (2nd part of A, n-j, i-j)



Properties of RSelect

Claim : Rselect is correct (guaranteed to output ith order statistic)

Proof : by induction. [like in optional QuickSort video]

Running Time ? : depends on "quality" of the chosen pivots.

Ter
ver

What is the running time of the RSelect algorithm if pivots are always chosen in the worst possible way?

- $\theta(n)$
- $\theta(n \log n)$
- $\theta(n^2)$
- $\theta(2^n)$

Example :

-- suppose i = n/2
-- suppose choose pivot = minimum every time
=> $\Omega(n)$ time in each of $\Omega(n)$ recursive calls

Running Time of RSelect?

Running Time ? : depends on which pivots get chosen.
(could be as bad as $\theta(n^2)$)

Key : find pivot giving "balanced" split.

Best pivot: the median ! (but this is circular)

⇒Would get recurrence $T(n) \leq T(n/2) + O(n)$
⇒ T(n) = O(n) [case 2 of Master Method]
Hope : random pivot is "pretty good" "often enough"

Running Time of RSelect

Rselect Theorem : for every input array of length n, the average running time of Rselect is O(n)

-- holds for every input [no assumptions on data]

-- "average" is over random pivot choices made by the algorithm