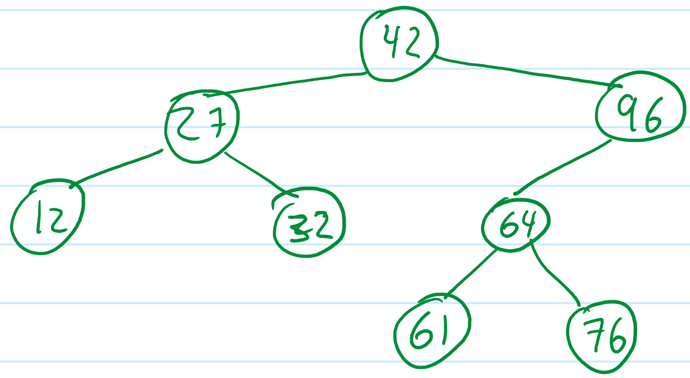
6.3 Decrease-by-a-Variable-Factor Algorithms

Tuesday, October 22, 2024 12:12 PM

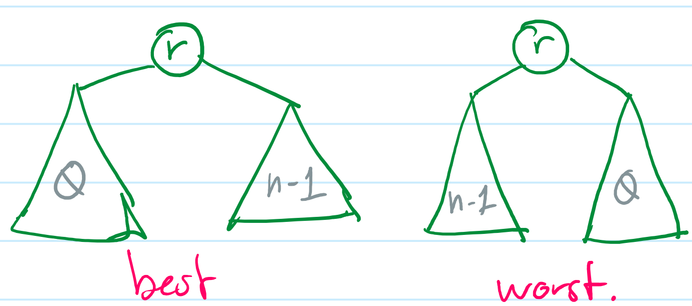
- Example Algorithm: Insertion and Search in a Binary Search Tree



Pseudo code

```
Search(T, x)
    r ← root of T
    if x < r then Search(left(T), x)
    if x > r then Search(right(T), x)
    else.
        x = r! you found x!!
```

- The problem size decreases, but you do not know by how much.
- $x < r$ size of tree $= n$



degenerate tree.

Analysis:

basic operation: comparison.

$C_{best}(n) = 1 \in \Theta(1)$

$C_{worst}(n) = n \in \Theta(n)$

Issue: Worst case analysis does not gives us a complete picture.
It is better to do average case complexity.

$C_{avg}(n) \in \Theta(\log n)$

- Problem: The Selection Problem (median problem)

- given an unsorted array $A[0 \dots n-1]$ find the K'th largest element.

- special case: if $k = \lceil \frac{n}{2} \rceil$ i.e. find the "median"

idea #1:

1:- Sort the array
2:- look at the $A[n-k-1]$ element.
⊖ you are sorting the whole array

idea #2:

1:- scan the array
2:- maintain the k biggest elements you have seen so far.
⊖ If K is large, the table of largest elements is difficult to maintain

idea #3:

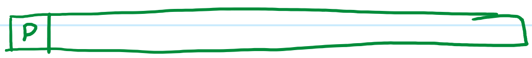
idea #3:

use a consept called "partitioning"

Partitioning.

Suppose you have a friend that

Given:



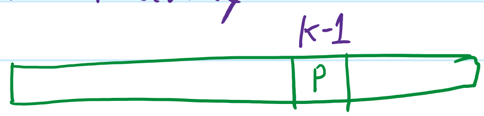
Gives you:

Note: • partitioning is **not** sorting.
• p is in it's "sorted" place.

Draft Algorithm: to find the k'th biggest element

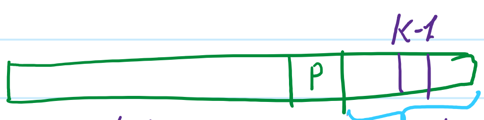
1: partition array

Case 1 :-



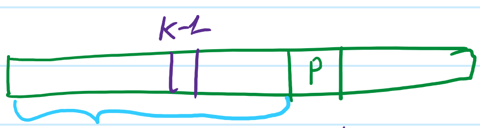
-return p.

Case 2 :-



-partition the right section.

Case 3 :-



-partition the left section

e.g. Trace.

[20, 31, 4, 25, 6, 3, 29, 7, 33]   n = 9   k = 3

p

⇓

[4, 6, 3, 7, 20, 31, 25, 29, 33]

p

⇓

[- - - - - - - - 25, 29, 31, 33]

p

[- - - - - - - - 25, 29, 31, 33]

p

[~~~~~~~ , 25, 29, ~~~~]

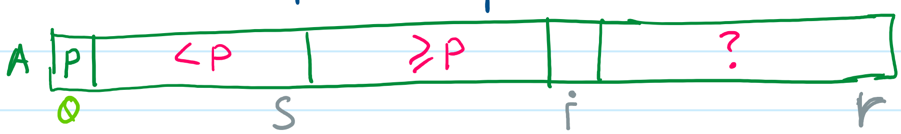
p

[~~~~~~~~~~~ , 29, ~~~~]

How to partition? Lomuto Partitioning.

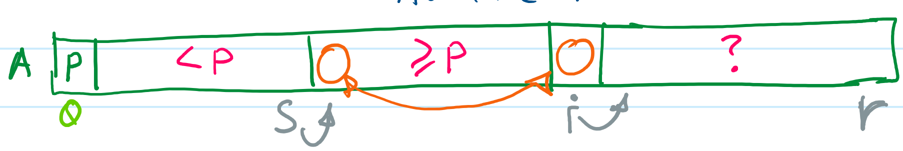
Intuition:

consider a partial partition

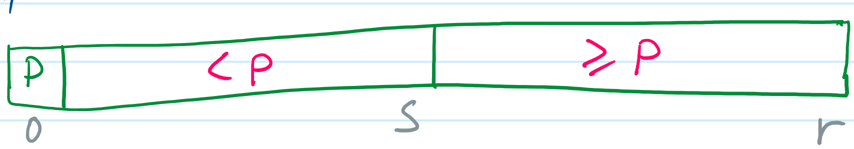


what to do with A[i]?

cases
- A[i] ≥ p advance i
- A[i] < p
  - Swap A[i] with A[s+1]
  - Advance s
  - Advance i



eventually



then: swap A[0] with A[s]

```
FUNCTION LomutoPartition( A[l..r] )
    p ← A[l]
    s ← l
    for i ← l+1 to r do
        if A[i] < p then
            swap( A[i], A[s+1] )
            s ← s+1
```

swap(A[i], A[s+1])
s ← s+1

swap (A[l], A[s])
return s

Let's use Lomuto Partition to find k'th largest element.

FUNCTION QuickSelect (A[l..r], k)
    s ← LomutoPartition (A[l..r])
    if s = k-1 then
        return A[s]
    else
        if s > l + k-1 then
            QuickSelect (A[l.. s-1], k)
        else
            QuickSelect (A[s+1, r], k)

Analysis:

basic operation:

$C_{best}(n) = n-1 \in \Theta(n)$

$C_{worst}(n) = (n-1) + (n-2) + (n-3) + .. + 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$

pivot ends on the extremes

$C_{Avg}(n) \in \Theta(n)$