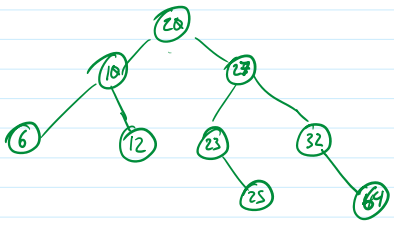
6.3 Variable-Size-Decrease Algorithms

Tuesday, October 24, 2023 11:35 AM

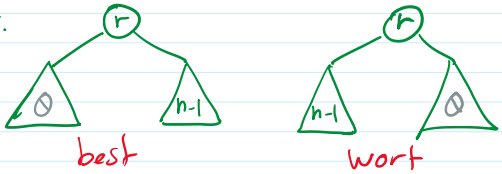
- Insertion and Search in a Binary Search Tree



pseudo.. insert(x)
given a tree t with root r.
if x > r
insert in the right subtree
else if x < r
insert in the left subtree
else do nothing.

- the problem decreases, but you do not know by how much.
for a tree of size n

x < r.



degenerate tree

basic operation comparison

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

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

Issue with Decrease-by-variable size algorithm is that worst case analysis may not give you a complete picture.
It is better to use average case complexity.

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

- The Selection Problem. Plus the median.
  - given an unsorted array $A[0 \ldots n-1]$, find the kth largest element.
  - Special case: if $k = \lceil \frac{n}{2} \rceil$ then we are finding the median of A

idea #1:
1.- sort the list.
2.- look at the $A[n-k-1]$ element.

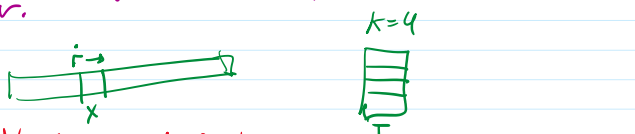
⊖ we don't to order every element.

- X

idea #2

1- Scan the array remember k largest elements you have seen so far.

e.g.



⊖ table T is difficult to maintain, and may be too large.

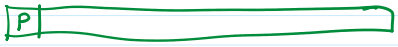
idea #3:

based on operation called "partitioning"

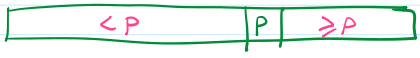
partitioning:

Suppose you have a friend that

Given array



Gives you:

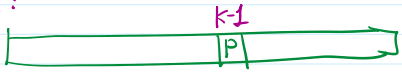


NOTE: • partitioning is not sorting.
• P is in its "sorted" place.

Draft. to find kth biggest element

1- partition array

Case 1:

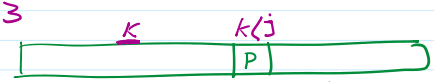


return p.

Case 2:

partition the right section

Case 3



partition the left section.

e.g. P

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

find the kth largest element. k=4

0 1 2 3 4 5 6 7 8

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

partition

0 1 2 3 4 5 6 7 8

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

unsorted ✓ unsorted?

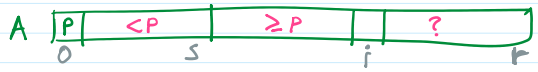
Back to partitioning: ¿How?

Intuition:

Back to partitioning: 2 more!

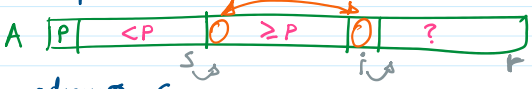
Intuition:

Suppose you have a partial partition.



what to do with A[i]?

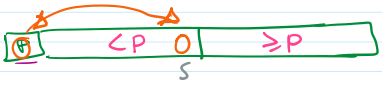
- A[i] ≥ p
  advance i
- A[i] < p
  - swap A[i] with A[s+1]



  - advance s
  - advance i

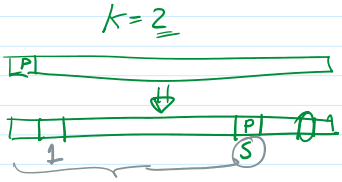
Algorithm: "Lomuto Partition"

```
FUNCTION LomutoPartition(A[l..r])
    p ← A[l]
    s ← l
    FOR i ← l+1 TO r DO
        IF A[i] < p THEN
            s ← s+1
            swap(A[s], A[i])
    swap(A[l], A[s])
    RETURN s
```



Now use this function to find the K'th 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-s-1)
```



Analysis.

basic-operation: Comparison

$C_{best}(n) = n-1 \in \Theta(n)$ the first pivot selected is kth element.

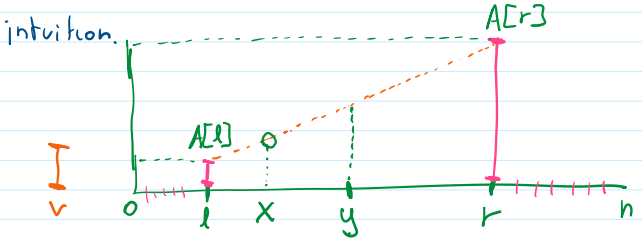
$C_{worst}(n)$ = pivot ends at the extremes

$= (n-1) + (n-2) + (n-3) + \dots + 1 = \frac{(n-1)\cdot n}{2} = \Theta(n^2)$

$C_{Avg}(n) = \Theta(n)$ ← specially with smarter ways of picking the pivot

- Interpolation Search.

Given a sorted Array: A
find an element. v



Suppose that the values in the array are linear.

$$x = l + \left\lfloor \frac{(v - A[l])(r-l)}{A[r] - A[l]} \right\rfloor$$

Algorithm:

```
FUNCTION InterpolationSearch(A[0..n-1], V)
    l ← 0
    r ← n-1
    WHILE l ≤ r DO
        x = l ⌊ (v-A[l])(r-l) / A[r]-A[l] ⌋
        IF V = A[x]
            return x
        ELSIF V < A[x]
            r ← x-1
        ELSE
            l ← x+1
    RETURN -1
```

Analysis:

basic_operation Comparison

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

$C_{avg}(n) \in \Theta(\log_2 \log_2 n + 1)$

—o— EOF