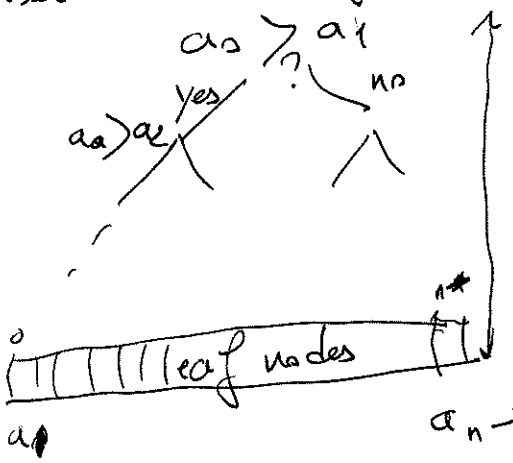
Lecture 9

Last time: Divide & Conquer Alg.
- Searching → for an element, or "selecting an element"
- Sorting.
  - Lower Bound on sorting problem.
    ⇓
    give a proof for the "problem" independent from any algorithm (which gives an upper bound).

Comparisons organized as a binary tree.



k 2^k leaf nodes.

there are $n!$ ways of ordering the indices

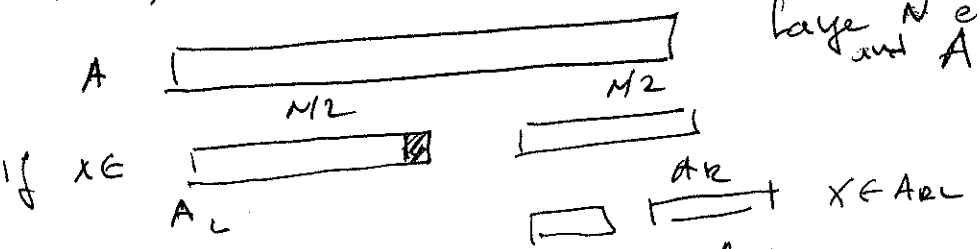
$n! = 2^k \Rightarrow$

$\log(n!) \geq cn \log n \Rightarrow$ $\Omega(n \log n)$

Today: let's start with binary search

given an array A of numbers, and another number X

Question: is X in A?



Large N elements and A is SORTED

$T(n) = a \cdot T\left(\left\lceil \frac{n}{b} \right\rceil\right) + d(n^d)$

$= 1 \; T\left(\left\lceil \frac{n}{2} \right\rceil\right) + d(1)$

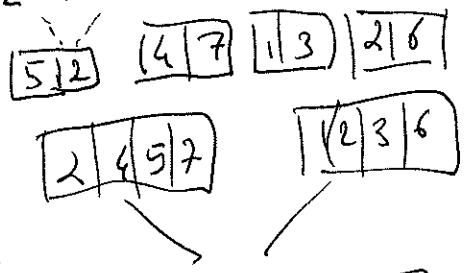
$= O(\log n)$ ☐

Merge sort ($a[1 \dots n]$)

outputs a sorted array

```
if n > 1
return MERGE(merge sort(a[1, ?, ..., n/2]),
             merge sort(a[n/2+1, ..., n]))
else return a
```

$a = [5, 2, 4, 7, 1, 3, 2, 6]$



$a = [1, 2, 2, 3, 4, 5, 6, 7]$

```
MERGE(X[1...k], y[1...l])
if   k=0   return y
if   l=0   return X
if   X[1] ≤ y[1]        concatenation operation
     return  X[1] || MERGE(X[2...k], y[1...l])
else return  y[1] || MERGE(X[1...k], y[2...l])
```

$$T(n) = 2\,T\left(\frac{n}{2}\right) + \underbrace{\Theta(k+l)}_{n} \Rightarrow \Theta(n \log n)$$

QuickSort: to sort the subarray $A[p \dots r]$

- Divide: Partition $A[p \dots r]$ into 2 (possibly empty) subarrays s.t.
  $A[p \dots q-1]$ and $A[q+1, \dots r]$
  (all the elements here $\leq A[q]$; $A[q]$; all these elements are $\geq A[q]$)
- conquer: Sort the subarray by recursive call to QuickSort
- Combine: no need for extra work since they are sorted in place.

```
QuickSort(A, p, r)
if p < r
   then q ← partition(A, p, r)
        QuickSort(A, p, q-1)
        QuickSort(A, q+1, r)
```

Initial call to QuickSort is (A, 1, n)

```
Partition(A, p, r)
  x ← A[r]
  i ← p-1
  for j: p to r-1
  Do if A[j] ≤ x
        then i ← i+1
             exchange A[i] ↔ A[j]
  exchange A[i+1] ↔ A[r]
  return i+1
```

x is our pivot element around the pivot we partition.

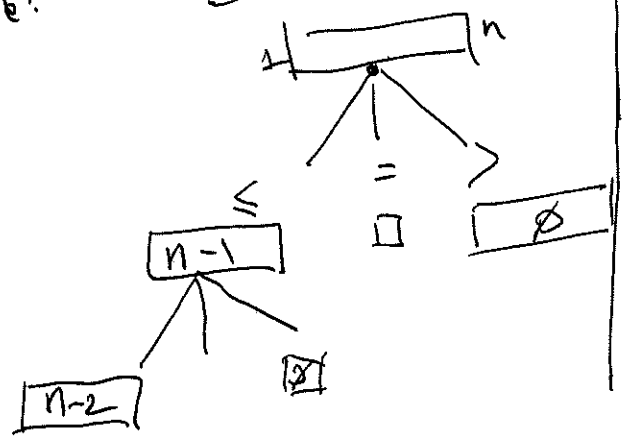
$A[p \dots i] \leq$ pivot

$A[i+1 \dots j-1] >$ pivot

$A[r]$ = pivot

example:

A = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ← x



the depth will be $\theta(n)$ i.e. linear in the input size n.

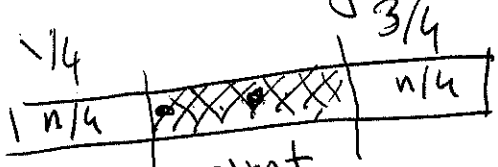
$\Rightarrow$ for this input we have the worst case time complexity for QuickSort

$$T(n) = T(n-1) + T(0) + \theta(n)$$

$$= \theta(n^2) \quad \square$$

Best case: $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n) \Rightarrow \theta(n \log n)$

Randomized Selection/Sorting / search to obtain average case performance.



Prob of hitting the shaded region is $p = \frac{1}{2}$

$T(n) = T\left(\frac{9}{10}n\right) + T\left(\frac{1}{10}n\right) + \dots$

$= \log_b n$

expected # of tries before the first success.

$E = 1 + \frac{1}{2}E$

1 success

2 tries will give you success!

$\Rightarrow$ geometric distribution mean

$E[X] = \frac{1}{p}$

$\Rightarrow$ on average after 2 recursive calls we can discard $\frac{1}{4}$ of the items

$T(n) = T\left(\frac{3}{4}n\right) + T\left(\frac{1}{4}n\right) + \Theta(n)$

suppose partition always produces 9 to 1 split

$T(n) \le T(9n/10) + T(n/10) + \Theta(n)$

$= \Theta(n \log n)$

Randomized QuickSort

Randomized-Partition (A, p, r)

i ← random(p, r)

exchange A[r] ↔ A[i]

return Partition (A, p, r)

- we assume all input permutations are equally likely (not always true)
- we add randomizations to QS.

idea 1: randomly permute the elements in the input.

idea 2: randomly choose the pivot: random sampling.