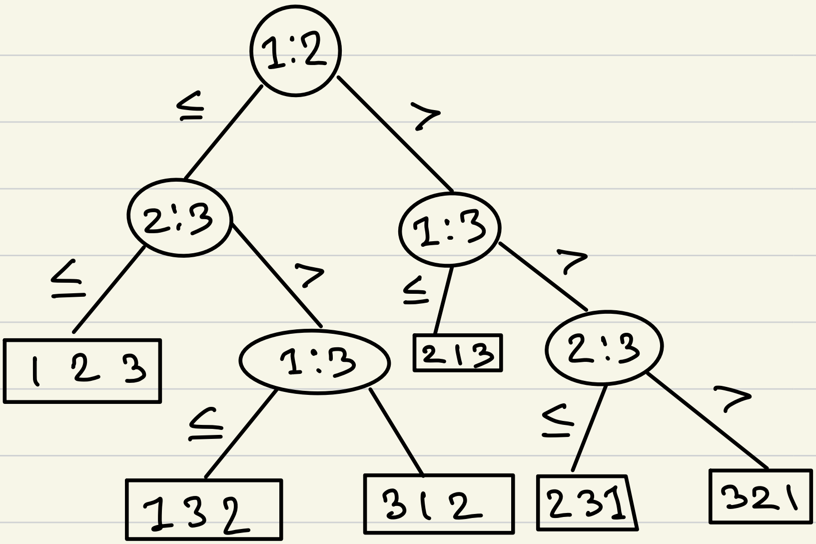
Lower Bounds

Sorting Lower Bound

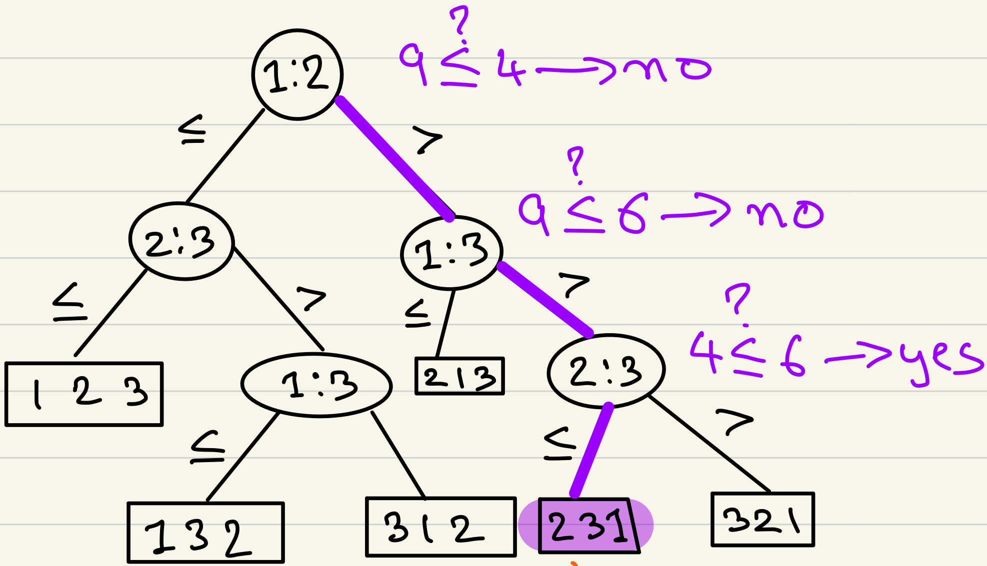
▷ Decision Tree Analysis



▷ i : j ⇒ if(A[i] ≤ A[j]); { yes → left, no → right }

▷ $\boxed{\text{i j k...}}$ ⟶ Leaf node

Use Case | A = < 9, 4, 6 >



Correct order is $A[2] \le A[3] \le A[1]$
or < 4, 6, 9 >

▷ Observations

- ▷ There can be multiple trees
- ▷ Runtime on A[...] = height of root → leaf path
- ▷ We're only counting comparisons, actual runtime could be worse.
- ▷ Worst case T(n) = height of the entire tree.

▷ Proof of $n \log n$ being the sorting lower bound

For $A[1..n]$

No. of leaf nodes (possible order) = $n!$

Since the tree is binary,

height of tree $(h) \geq \log(\text{no. of leaves}) = \log(n!)$

$\Rightarrow h \geq \log(n!) \longrightarrow (\Theta(n \log n))$

$$\Rightarrow \boxed{h = \Omega(n \log n)}$$

Since worst case $T(n)$ = height of tree

$$T(n) \geq n \log n$$

or

$$\boxed{T(n) = \Omega(n \log n)}$$

According to decision tree, no sorting algorithm can do better than $n \log n$.

But is the decision tree the best way to find answers to complex question?

Ans: No!

▷ ELEMENT UNIQUENESS (EU)

Given an an array $A[1..n]$, find if there are any duplicates in the array.

THEOREM: Any decision Tree for elemental uniqueness has height in $\Omega(n \log n)$

Any DT, T, for EU $\Rightarrow$ DT for sorting arrays with distinct elements

▷ Every leaf node encodes all operations required to sort.

▷ Proof | DT for EU ⇒ DT for sorting

$\forall$ T for EU

Assumptions:

▷ Input of form $A[1..n]$, with distinct elements for all permutations

$\therefore T : [\{1..n\}, \{1..n\} ..]$

$\Rightarrow \underbrace{A_\pi[i]}_{\text{Unique element}} \Rightarrow \underbrace{\pi(i)}_{\text{All perm. of } A_\pi[i]}$

▷ Use notation $\pi^{-1}(k) = i$ s.t. $\pi(i) = k$

Eg: $\pi = (3, 1, 2) \Rightarrow \pi(3) = 2, \pi^{-1}(2) = 3$

index of 2 in the perm.

Claim:

Let $\forall \pi$ ($\Rightarrow A_\pi$), the path with all "yes" outcome be P. (ends in a "yes" leaf node)

$\forall k \in [1...n]$, let $i = \pi^{-1}(k)$, $j = \pi^{-1}(k+1)$

$\Rightarrow$ $i : j$ $\in P$

comparison b/w $A[i]$ & $A[j]$

For eg: $\pi = (4,2,1,3)$, $K=3$

$\Rightarrow i = \pi^{-1}(3) = 4$, $j = \pi^{-1}(4) = 1$

$\therefore$ P contains 4:1

<u>Validation:</u>

$\forall K \in [1 \dots n]$, $i = \pi^{-1}(K)$, $j = \pi^{-1}(K+1)$

Let A' be $A[1 \dots n]$, except $A'[j] = K$

For eg: If $A = (4,2,1,3)$ $K=3 \Rightarrow A'(3,1,2,3)$

Now, $EU(A_\pi) = Yes$ & $EU(A'_\pi) = No$

$\Rightarrow T(A) = \{\dots (i:j) \to y\} \to EU(A) = Yes$ &

$T(A') = \{\dots (i:j) \to n\} \to EU(A') = No$

Basically, i:j is the only difference between T(A) & T(A').

Or, if i:j $\notin$ P, then $T(A) = T(A')$, since all other comparisons are same, $\Rightarrow T(A') = Yes$, which is wrong.

Hence, for the unique path P, with all "yes",

$$\boxed{i:j \in P}$$

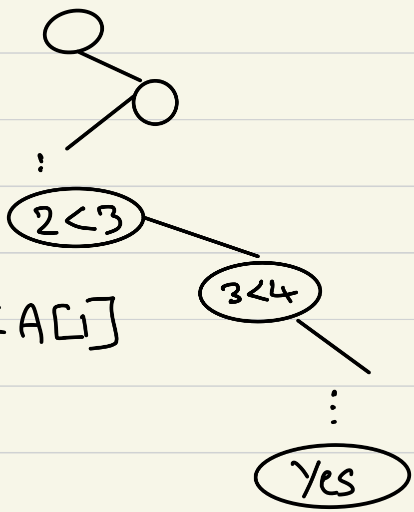
Corollary

$\forall \pi$, if P is induced path of $T(\pi)$, then P is actually comparing numbers (not indices).

$\therefore$ All the comparisons for $A[1..n]$, i.e., $\{(1 \overset{?}{\leq} 2), (2 \overset{?}{\leq} 3), (3 \overset{?}{\leq} 4) .., (n-1 \overset{?}{\leq} n)\}$ uniquely determine π, s.t. $EU(A_\pi) = Yes$

For eg: $A[4, 2, 1, 3]$

P $\rightarrow$



$1 < 2 < 3 < 4$

$\updownarrow$

$A[3] < A[2] < A[4] < A[1]$

Since all comparisons are included in P,

DT of EU $\Rightarrow$ DT of Sorting

$\Rightarrow$ Runtime of EU $\geq n \log n$

▷ Sorting In Linear Time | Counting Sort

But how do we sort in linear time?

Ans: No comparisons!

Intuition

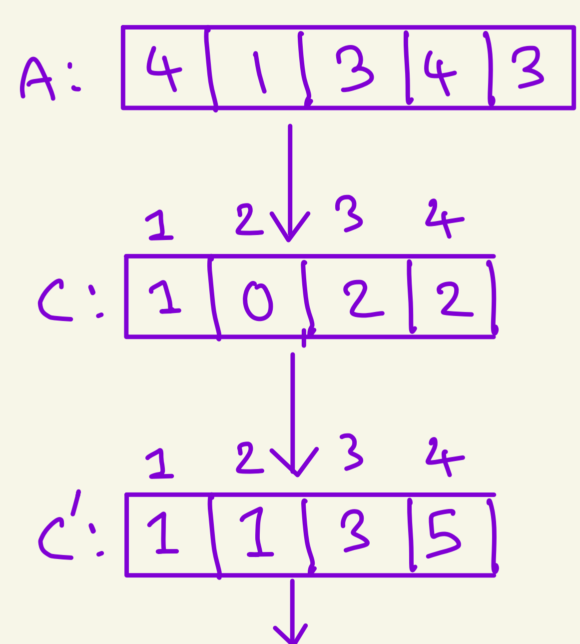
Input: $A[1 \dots n]$, where $A[j] \in \{1, 2, \dots k\}$

Output: $B[1 \dots n]$, sorted

Auxiliary storage: $C[1 \dots k]$

Imagine an input array $A[4, 1, 3, 4, 3]$

Broadly, we do the following,



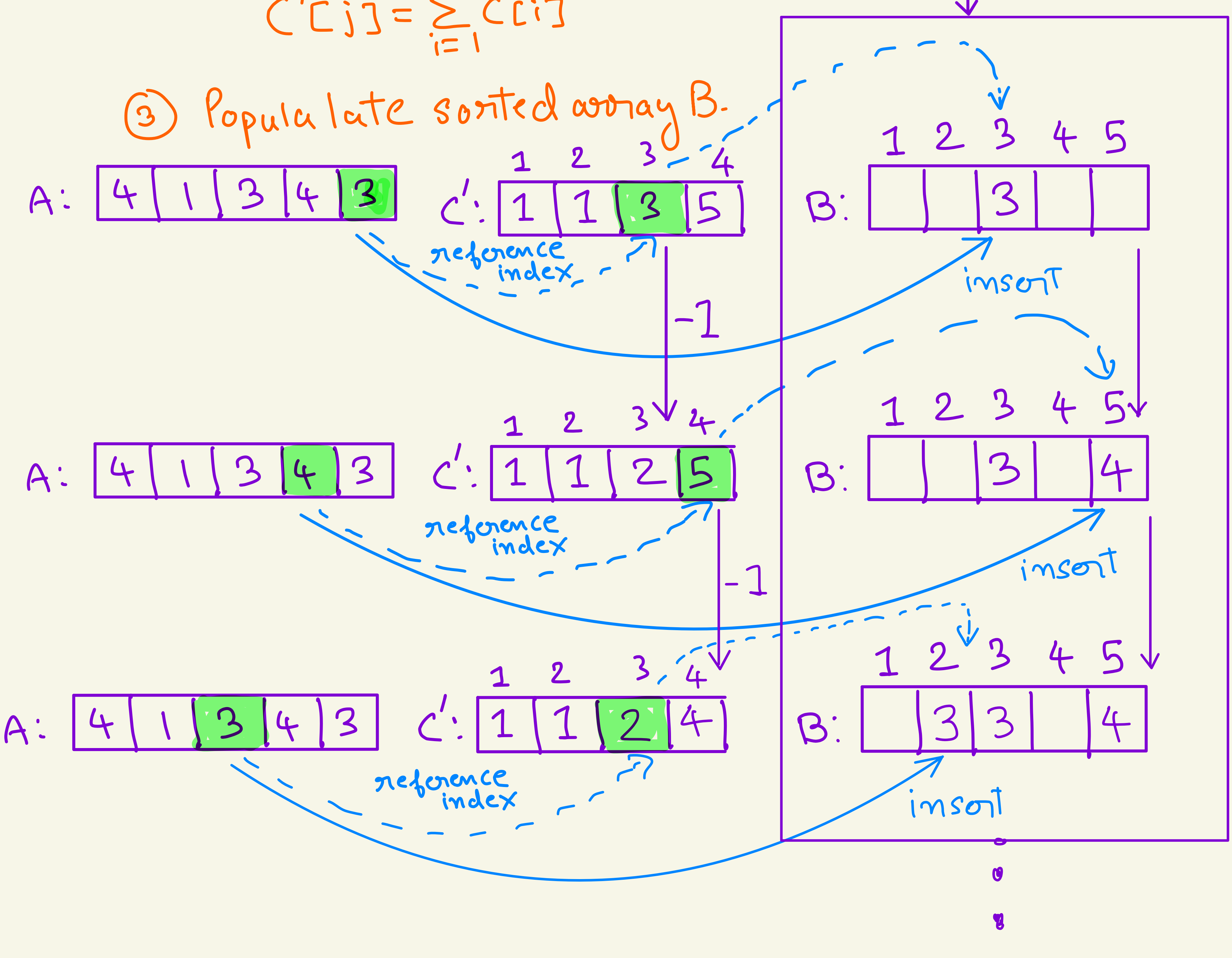
① Count occurrences

② Count values $\leq C[i]$

or

$$C'[j] = \sum_{i=1}^{j} C[i]$$

③ Populate sorted array B.



Sorted ⟶ B: 1 | 3 | 3 | 4 | 4

Analysis

$\Theta(k)$ — **for** $i \leftarrow 1$ **to** k
  do $C[i] \leftarrow 0$

$\Theta(n)$ — **for** $j \leftarrow 1$ **to** n
  do $C[A[j]] \leftarrow C[A[j]] + 1$

$\Theta(k)$ — **for** $i \leftarrow 2$ **to** k
  do $C[i] \leftarrow C[i] + C[i-1]$

$\Theta(n)$ — **for** $j \leftarrow n$ **downto** 1
  do $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

$\Theta(n + k)$

Running time

If $k = O(n)$, then counting sort takes $\Theta(n)$ time.

- But, sorting takes $\Omega(n \lg n)$ time!
- Why ?

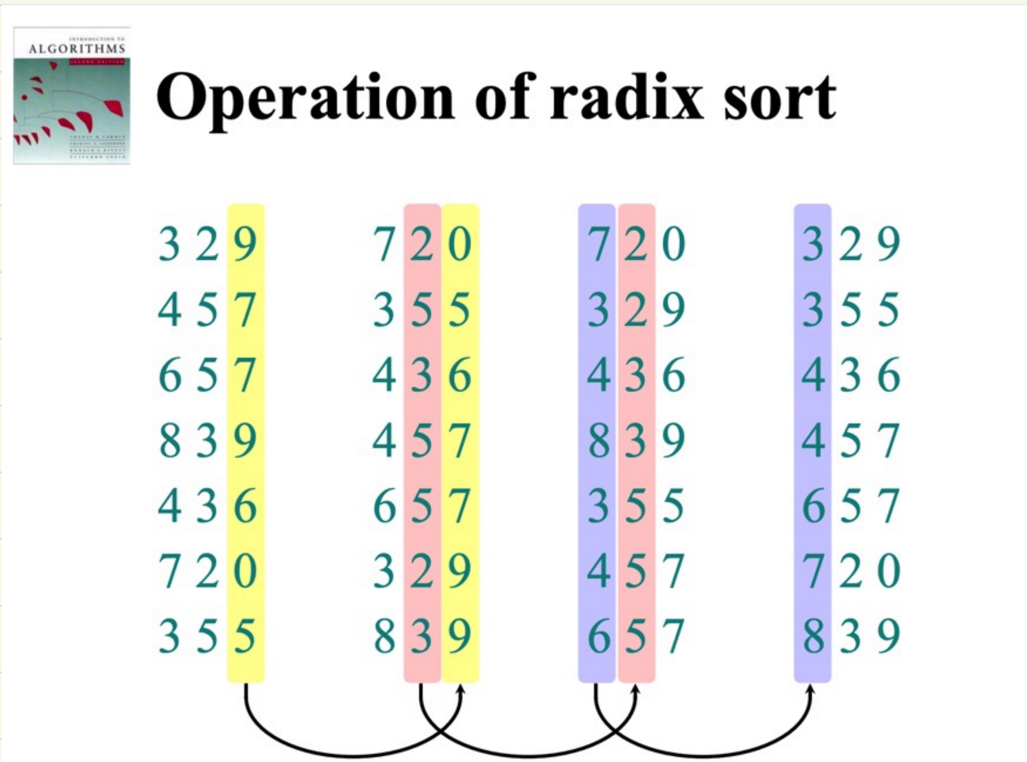
Answer:

- ***Comparison sorting*** takes $\Omega(n \lg n)$ time.
- Counting sort is not a ***comparison sort***.
- In fact, not a single comparison between elements occurs!

Stable Sorting: Counting sort is a stable sort as it preserves the input order among equal elements

▷ RADIX SORT

- Sort numbers digit-by-digit.



▷ Correctness of radix sort

For sorting on t^{th} digit, we assume that the numbers are sorted for $t-1$ digits.

▷ We sort on t^{th} digit

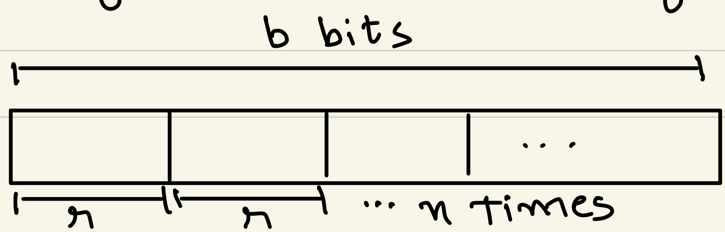
① Two numbers with different t^{th} digit are correctly sorted

② Two numbers with equal t^{th} digit are put in the same order as before.

∴ The numbers are sorted till the t^{th} digit

▷ Analysis of Radix Sort

For a give "b" bit number, we can split it into "n" digits, each "r" bit long.



Hence, we have n digits in base 2^r
where $n = b/r$

Recall Counting sort runtime $= \Theta(n+K)$ to sort
n numbers in range $[0, K-1]$

Use Case | $b = 32$, $m = 64$ numbers

① $r = 32 \Rightarrow 1$ 32-bit number

$$T(n) = 1 \times (2^{32} + 64) \approx 2^{32}$$

② $r = 16 \Rightarrow 2$ 16-bit numbers

$$T(n) = 2 \times (2^{16} + 2^{6}) \approx 2^{17}$$

③ $r = 4 \Rightarrow 8$ 4-bit numbers

$$\Rightarrow T(n) = 8(2^{4} + 64) = 640$$

④ $r = 1 \Rightarrow 32$ 1-bit numbers

$$\Rightarrow T(n) = 32(2 + 64)$$

$$\approx 2^{11}$$

$\therefore T(n \mid r=4) \ll T(n \mid r=1) \ll T(n \mid r=16) \ll T(n \mid r=32)$

Now,

$$T(n|r) = \frac{b}{r}\left(n + 2^r\right)$$

For optimal r,

① if $r << \log_2 n$, i.e., $2^r << n$

$$\Rightarrow T(n|r) = \frac{b}{r}\left(n + 2^r\right) \approx \frac{bn}{r}$$

$$\Rightarrow T(n|r) = O\left(\frac{bn}{\underbrace{r}}\right)$$

push toward $\log_2 n$ to decrease runtime

② if $r >> \log_2 n$, i.e., $2^r >> n$

$$\Rightarrow T(n|r) = O\left(\frac{b \cdot 2^r}{r}\right)$$

reduce to $\log_2 n$ to decrease runtime.

By minimizing $T(n|r)$, we get

$\boxed{r \approx \log_2 n}$ or base $2^r \approx n$

$$\boxed{T(n) = O\left(\frac{bn}{\log n}\right)}$$

Use Case | Radix sort numbers from $1 \ldots n^d$

$$r = \log_2 n \Rightarrow \text{base } n$$

$$\therefore\ b = d \log_2 n$$

$$\boxed{T(n) = O(dn)}$$

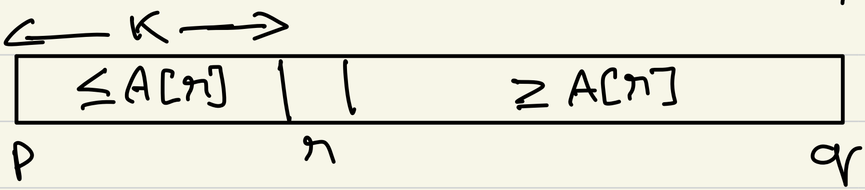
ORDER STATISTICS

Use Case | Find i^{th} smallest element in array

▷ <u>Randomized divide-and-conquer</u>

Essentially, we use the partition function of randomized quicksort.

```
Rand-Select(A, p, q, i)            --> i^th smallest in A[p..q]
    if p==q then return A[p]
    r <-- Rand-partition(A, p, q)
    k <-- r-p+1                    --> k = rank(A[r])
    if i==k then return A[r]
    if i<k
        then return Rand-Select(A, p, r-1, i)
    else return Rand-select(A, r+1, q, i-k)
```



Analysis

- Worst case running time
  - Unlucky: $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$
  - Lucky: $T(n) = T(9n/10) + \Theta(n) = \Theta(n)$

- Expected running time $= O(n)$

Conclusion

- Linear expected time
- Bad worst case

Can we do better in worst case?

Ans: Yes, just generate a good pivot.

Worst-case Linear-Time Order Statistics

$T(n)$	SELECT(i, n)
$\Theta(n)$	1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
$T(n/5)$	2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
$\Theta(n)$	3. Partition around the pivot x. Let $k = \text{rank}(x)$.
$T(7n/10)$	4. **if** $i = k$ **then return** x **elseif** $i < k$ **then** recursively SELECT the ith smallest element in the lower part **else** recursively SELECT the $(i-k)$th smallest element in the upper part

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n)$$

For a large enough constant "c".

$$T(n) \le \frac{1}{5}cn + \frac{7}{10}cn + \Theta(n)$$

$$T(n) = \frac{18}{20}cn + \Theta(n)$$

$$T(n) = cn - \left(\frac{2}{20}cn - \Theta(n)\right)$$

$$\boxed{T(n) \le \underbrace{c}_{}n}$$

However, in practice, "c" is very large. Not very useful.

▷ Randomized select is more practical.