Median Stats

- “find k-th element” ranked

$k=1 \rightarrow$ min
$k=n \rightarrow$ max
$k=n/2 \rightarrow$ median

- better approach, based on QuickSort
- ~~Median~~ QuickSelect(A,b,e,k) *//find k-th greatest in array A, sort between indices b=1 and e=n*

  $\boxed{k=7}$ task => left k=7
  q=12 $\boxed{k=17}$ task => right k=5

  - $\boxed{\text{q = Partition(A,b,e)}}$ *//returns pivot index q, b<=q<=e*
  - *//Partition also rearranges A so that if* `i<q then A[i]<=A[q]`
  - *and if* `i>q then A[i]>=A[q]`

  b ≤q q-1 A[q] q+1 ≥q e

  correct spot/rank

  - if(q==k) return A[q] *//found the k-th greatest*
  - if(q>k) ~~Median~~ QSel(A,b,q-1,k)
    - else ~~Median~~ QSel(A,q+1,e,~~q-k~~ k-q)
- Not like Quiksort, Median recursion goes only on one side, depending on the pivot
- why the second Median call has $k_{(new)}=q-k_{(old)}$?

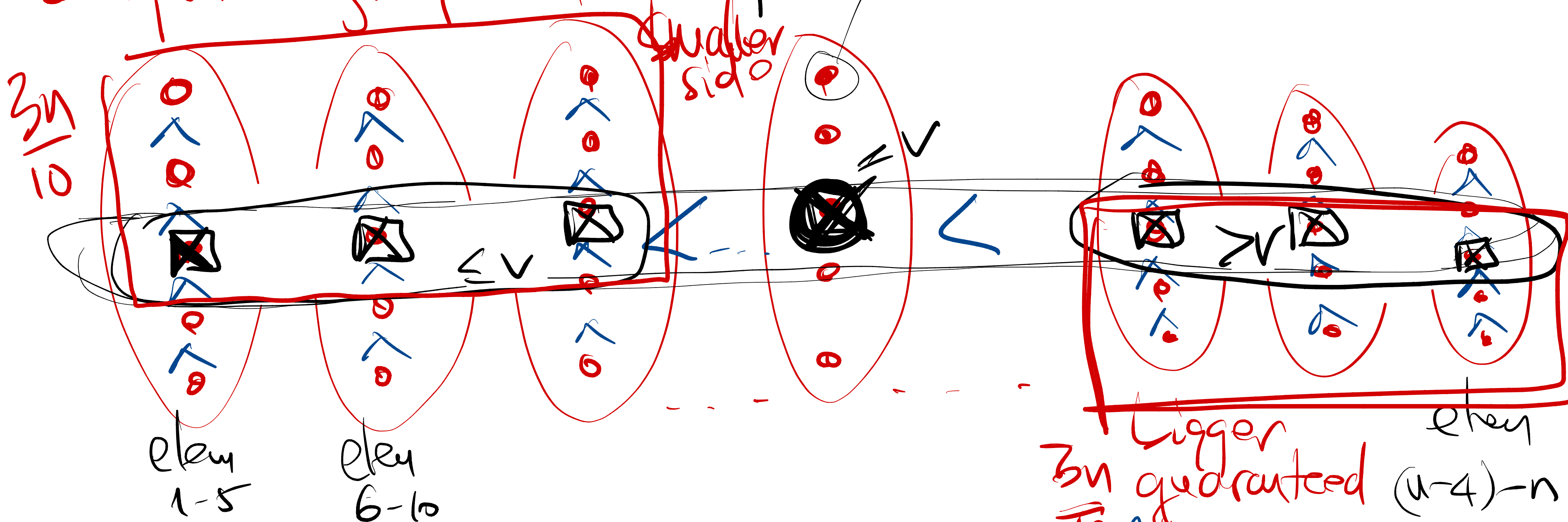
Median Stats

- Running Time of Median
- the recursive calls makes $T(n) = n + \max(T(q), T(n-q))$ worst case
  - "max" : assuming the recursion has to call the longer side
  - just like QuickSort, average case is when q is "balanced", i.e. $cn<q<(1-c)n$ for some constant $0<c<1$
  - $\boxed{\text{balanced case: } T(n) = n + T(cn)}$; Master Theorem gives linear time $\Theta(n)$ $T(n) = n + T\left(\frac{999}{1000}n\right)$
  - expected (average) case can be proven linear time (see book); worst case $\Theta(n^2)$ $T(n) = n + T(n-1) \Rightarrow \Theta(n^2)$
- worst case can run in linear time with a rather complicated choice of the pivot value before each partition call (see book)

QSel + Fix For Linear Time

value in array

- Split in groups of 5 : groups are not sorted across

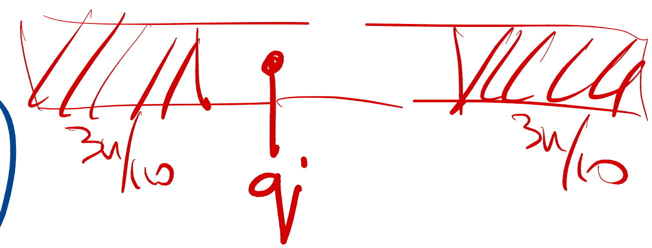


- sort each group (5 elem) const time × $\frac{n}{5}$
- consider ⊠ median-of-5 in each group ⟹ $\frac{n}{5}$ medians
- Find median ($\frac{n}{5}$ medians) ⊗ "center value"
- Use center value as pivot

Run Time (QSel + Linear Fix)

$$T_k(n) = \underbrace{\Theta(n)}_{\text{split}} + \underbrace{\Theta(n)}_{\text{sort groups}} + \overbrace{\boxed{T_{\text{median}}(n/5)}}^{\text{rec}} + \underbrace{\Theta(n)}_{\text{partition}} + \max \overbrace{\boxed{\begin{Bmatrix} T(q) \\ T(n-q) \end{Bmatrix}}}^{\text{rec}}$$

$$\frac{3n}{10} \le q \le n - \frac{3n}{10}$$



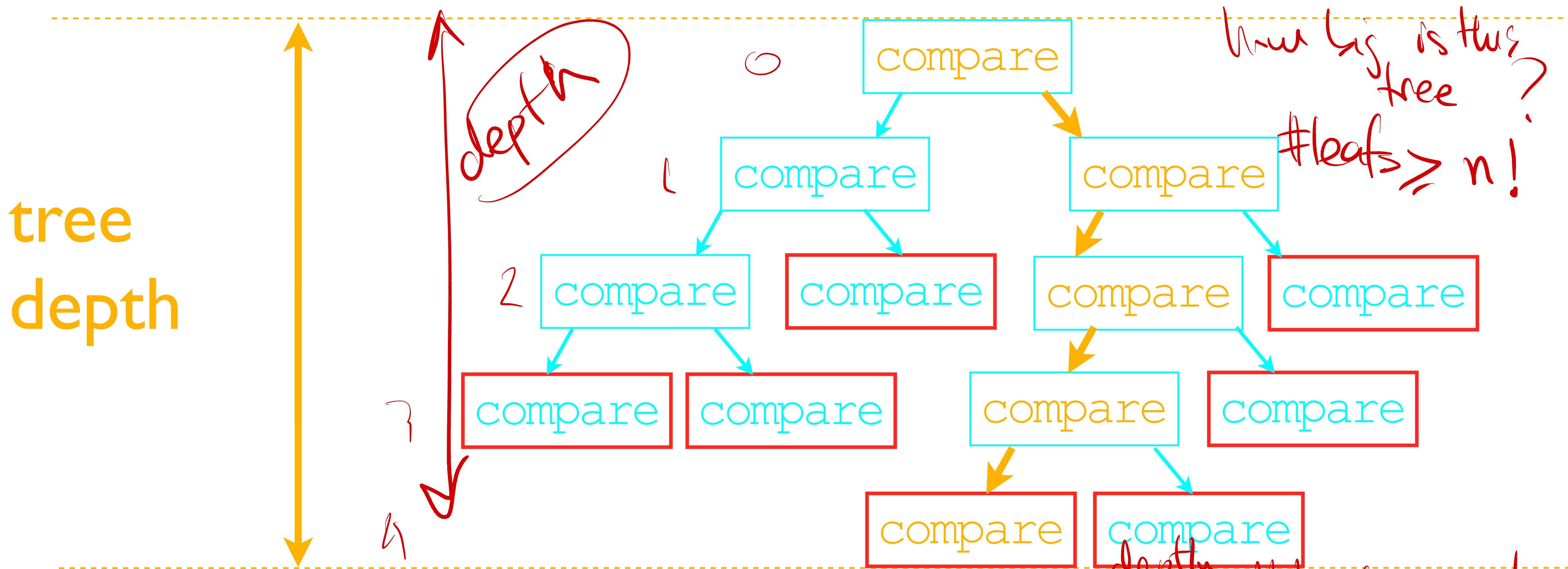
$$= \Theta(n) + T(n/5) + T\left(\frac{7n}{10}\right)$$

$$= \Theta(n)$$

<u>exer</u>: - pseudocode
- solve/argue recurrence
- trick also applies to QSort?

Sort by comparisons Alg $\Rightarrow$ R.T $= \Omega(n \log n$

Sorting : tree of comparisons



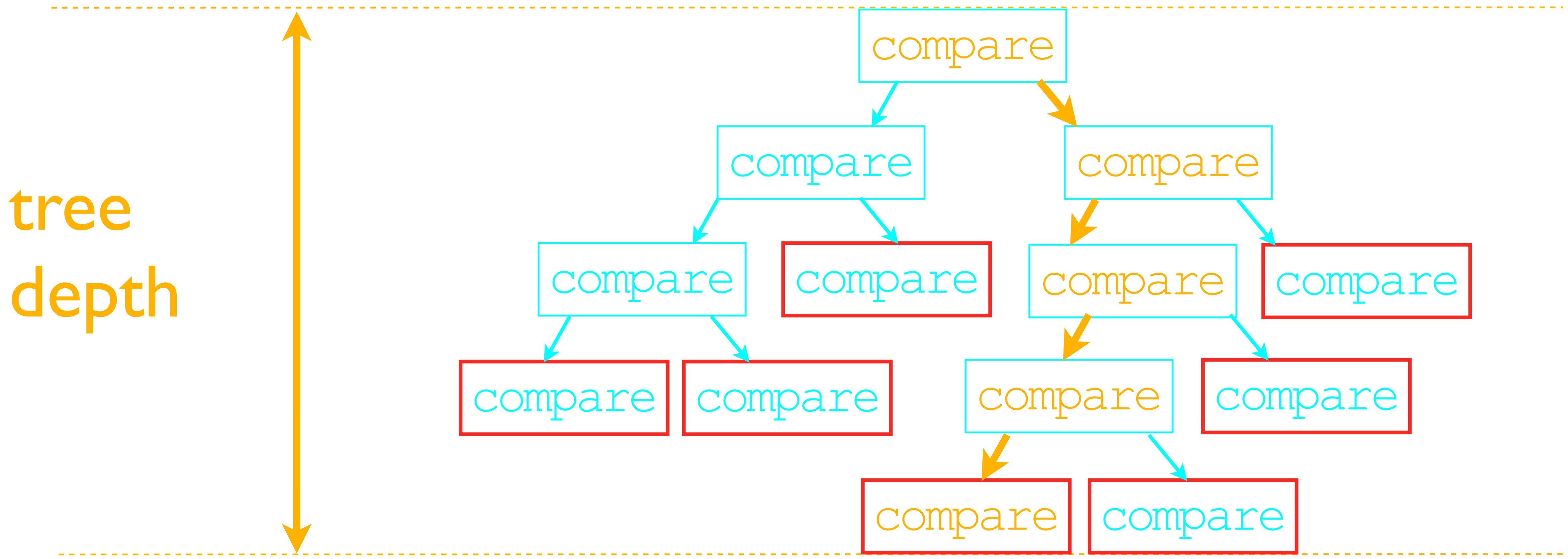
- tree of comparisons : essentially what the algorithm does

worst case = longest path = depth $\geq \log_2(n!)$

exercise $\Theta(n \log n)$

  - each program execution follows a certain path
  - red nodes are terminal / output
  - the algorithm has to have n! output nodes... why ?
  - if tree is balanced, longest path = tree depth = n log(n)

Sorting : tree of comparisons



- tree of comparisons : essentially what the algorithm does
  - each program execution follows a certain path
  - red nodes are terminal / output
  - the algorithm has to have n! output nodes... why ?
  - if tree is balanced, longest path = tree depth = n log(n)

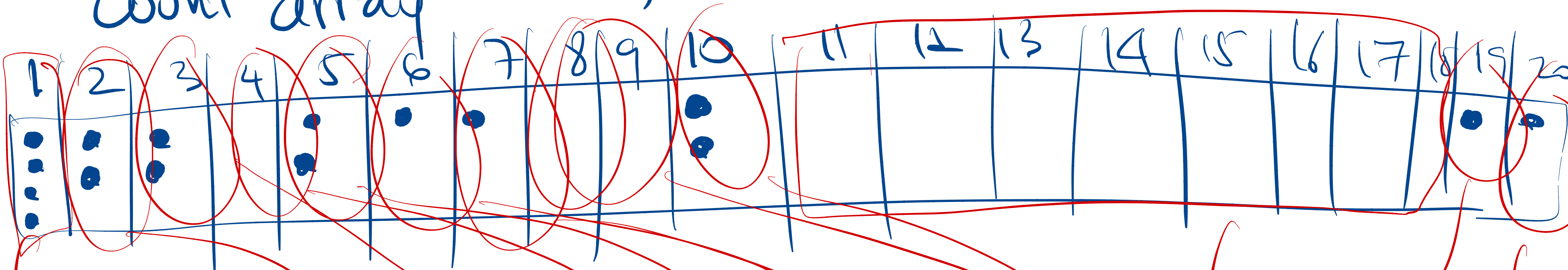
Array : A = [1, 3, 10, 2, 20, 5, 1, 6, 1,
3, 10, 19, 1, 5, 7, 2]

Counting Sort

1) count each value in RANGE [1 : 20] FIXED

$\Theta(n)$ $|RANGE| = K = 20$

count array



2) output values in order, freq = count

1 1 1 1, 2 2, 3 3, 5 5, 6, 7, 10, 10, 19 20

$\Theta(K)$ + $\Theta(n) = \Theta(n+K)$

going through each possib. value

linear time! BUT

- RANGE fixed, constant k, discrete
- constant k = reasonable in practice

"$k \leq n$" in practice

Linear-time Sorting: Counting Sort

- Counting Sort (A[]) : count values, NO comparisons
- STEP 1 : build array C that counts A values
  - `init C[]=0 ;`
  - `run index i through A`
    - `value = A[i]`
    - `C[value] ++;` *//counts each value occurrence*
- STEP 2: assign values to counted positions
  - `init position=0;`
  - `for value=0:RANGE`
    - `for i=1:C[value]`
      - `position = position+1;`
    - `OUTPUT[position]=value;`

Counting Sort

- n elements with values in k-range of $\{v_1, v_2, \ldots v_k\}$
  - for example: 100,000 people sorted by age: n=100,000; k = {1,2,3,...170} since 170 is maximum reasonable age in years.
- Linear Time $\Theta(n+k)$
  - Beats the bound? YES, linear $\Theta(n)$, not $\Theta(n*\log n)$, if k is a constant
  - Definitely appropriate when k is constant or increases very slowly
  - Not good when k can be large. Example: sort pictures by their size; n=10000 (typical picture collection), size range k can be any number from 200Bytes to 40MBytes.
- Stable (equal input elements preserve original order)

Radix Sort

- Counting sort on each digit

Radix Sort

- Counting sort on each digit

329
457
657
839
436
720
355

Radix Sort

- Counting sort on each digit

329	720
457	355
657	436
839	457
436	657
720	329
355	839

Radix Sort

- Counting sort on each digit

329	720	720
457	355	329
657	436	436
839	457	839
436	657	355
720	329	457
355	839	657

Radix Sort

- Counting sort on each digit

329	720	720
457	355	329
657	436	436
839	457	839
436	657	355
720	329	457
355	839	657

Still sorted (due to stability) if the
current sort column does not

Radix Sort

- Counting sort on each digit

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	653 657
720	329	457	720
355	839	657	839

Still sorted (due to stability) if the
current sort column does not

Radix Sort Analysis

- inside Radix sorting proc = counting sort)
  discrete range [0:9] k=10 << n $\Theta(n+k)$
- total run time $\boxed{\#digits \times \Theta(n+k)}$
  Range 32 bits / $[0 : 2^{32}-1]$

- critical that the digit-sorting procedure is stable
  - (329, 355) remain properly sorted when the third digit is used
- counting sort fits the bill: stable, also linear when the range is fixed, like base 10 digits {0-9}
- each digit-sort is linear. But how many digits ?
  - quick informal answer: log(n) digits with fixed range, so O(n*logn) total.

Radix Sort Analysis

- each digit-sort is linear. We can represent items with few/many bits by choosing representation base
- b bits per item, n items. (b,n fixed).
  - for example computers typically represent integers on b=32 bits and long integers on b=64 bits
  - limit to 2^b items total

$r=3 \Rightarrow$ base 8
$r=2 \Rightarrow$ base 4

- use r bits per digit -> number of digits d = b/r. (r,d up to us, variables)
  - each digit sort $\Theta(n+2^r)$, $d=b/r$ digits, so total $\Theta(b/r*(n+2^r))$
  - choosing $r \approx \log(n)$, total is $\Theta(b/\log(n)*(n+n)) = \Theta(bn/\log(n))$

Sorting : stable; in place

- stable: preserve relative order of elements with same value
- in place: dont use significant additional space (arrays)

	time	in-place	stable
Bubble	n^2	✔	✔
Insertion	n^2	✔	✔
Selection	n^2	✘	?
QuickSort	n*log(n)	✔	?
MergeSort	n*log(n)	✘	✔

Sorting : stable; in place

- stable: preserve relative order of elements with same value
- in place: dont use significant additional space (arrays)

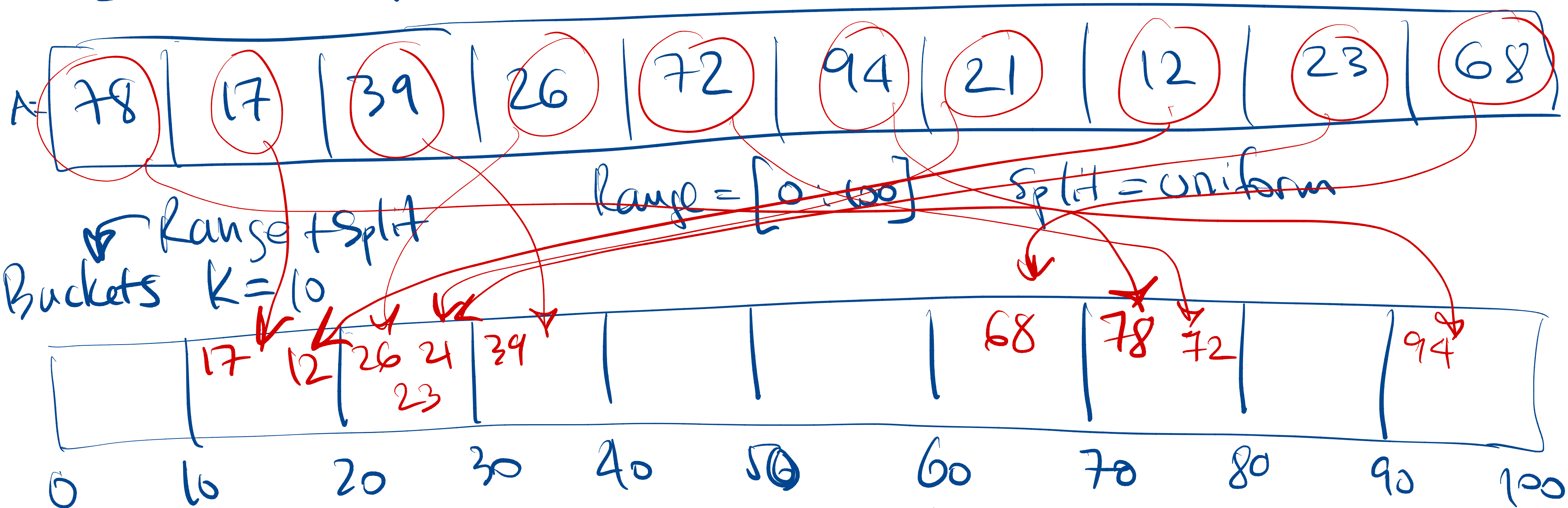
	time	in-place	stable
Bubble	n^2	✔	✔
Insertion	n^2	✔	✔
Selection	n^2	✘	?
QuickSort	n*log(n)	✔	?
MergeSort	n*log(n)	✘	✔

Sorting : stable; in place

- stable: preserve relative order of elements with same value
- in place: dont use significant additional space (arrays)

	time	in-place	stable
Bubble	n^2	✔	✔
Insertion	n^2	✔	✔
Selection	n^2	✘	?
QuickSort	n*log(n)	✔	?
MergeSort	n*log(n)	✘	✔

Bucket Sort



1) • place each value into correct bucket.

$\Theta(n+K)$

2) • for each bucket $\rightarrow \Theta(|\text{bucket}|^2)$
sort values in bucket, output them $+ \sum_{\text{buck}} \text{Sortin}(\text{buck})$

OBS: BS works for any K

$\boxed{\text{Choose } K=n}$

R.T. n elements total $a_1, a_2, \ldots a_n$

k buckets

unifrm assumpt

$n_1, n_2, \ldots n_k$ = size of the buckets $E[n_i] = 1$

[after the fact] random variables

$n_1 + n_2 + n_3 \ldots + n_k = n$

$\Theta(n+k)$ sort(bucket b)

R.T avg case $T(n) = \boxed{\Theta(n)} + \sum_{b=1}^{n} O(n_b^2)$

$E[T(n)] = \Theta(n) + E\left[\sum_{b=1}^{n} O(n_b^2)\right] =$

avg w.r.t. items to buckets

$= \Theta(n) + \sum_{b=1}^{n} E[n_b^2] \rightarrow \leq 2$

$\leq \Theta(n) + \sum_{b=1}^{n} 2 = \Theta(n)$

proof idea B. R.V. $X_{ij} = \begin{cases} 1 & \text{if item } i \to \text{bucket } j \\ 0 & \text{if not} \end{cases}$

n_j = # of items in bucket j = $\sum_{i = \text{item}} X_{ij}$

j = hash

$$n_j^2 = \left(\sum_i X_{ij}\right)^2$$

$$E\left(\sum X_{ij}\right)^2 = E\left[\sum X_{ij}^2 + \sum_{i \neq l} X_{ij} X_{lj}\right]$$

$(a+b+c)(a+b+c)$

a·a, b·b
prod-itself

a·b, a·c

$\simeq 1/n$? $\simeq 1/n^2$?