Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 14.

Selected Topics in Sorting&Selection

with

Rank/Select and Wavelet Tree Data Structures

M. Oguzhan Kulekci

- Brief Review of Sorting
- Selection with Qsort
- Rank/Select Dictionaries
- Wavelet Tree Data Structure

Sorting

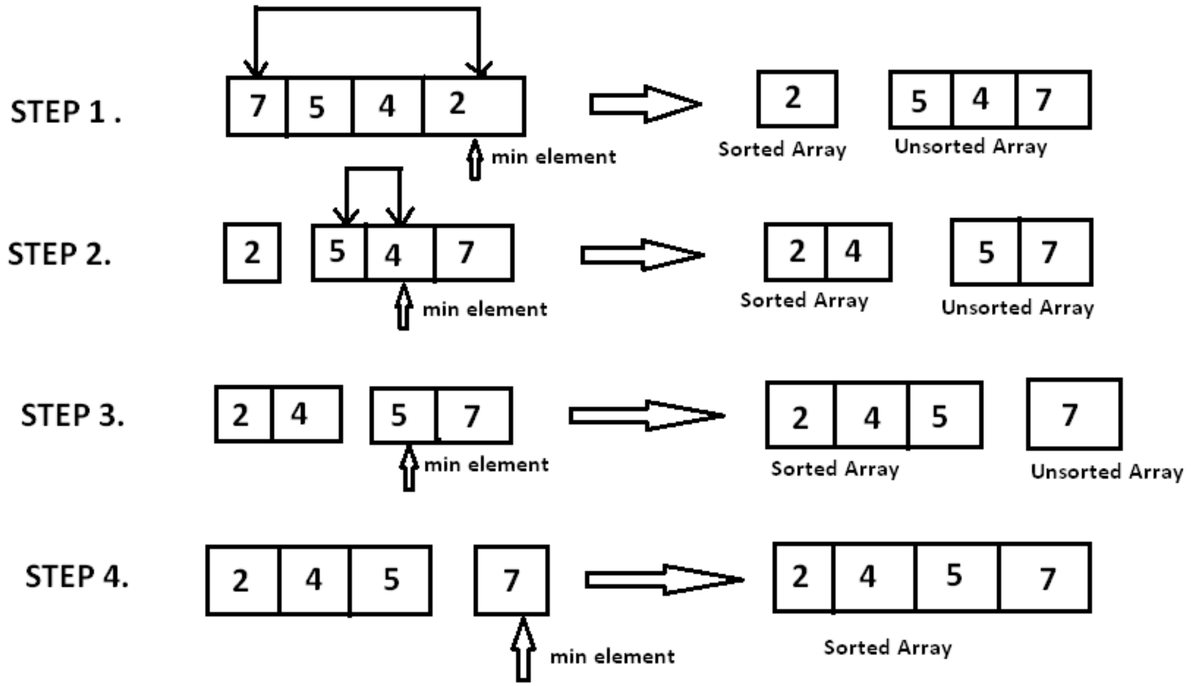
- One of the fundamental topics in computing science
- Significant amount of total computing time is devoted to sorting, why ?
- Basic building block in many algorithms
- We will briefly review them in three classes
  - $O(n^2)$ - time
  - $O(n \log n)$ - time (**optimal**)
  - $O(n)$ - time (…*Wait !!! Can this be true ?*)

Sorting - Some Terminogy

- In-place sorting
- Stable sorting
- Parallel sorting
- Sorting benchmarks, what are the metrics to evaluate a sort algorithm ?

Sorting - 1: $O(n^2)$-time Solutions

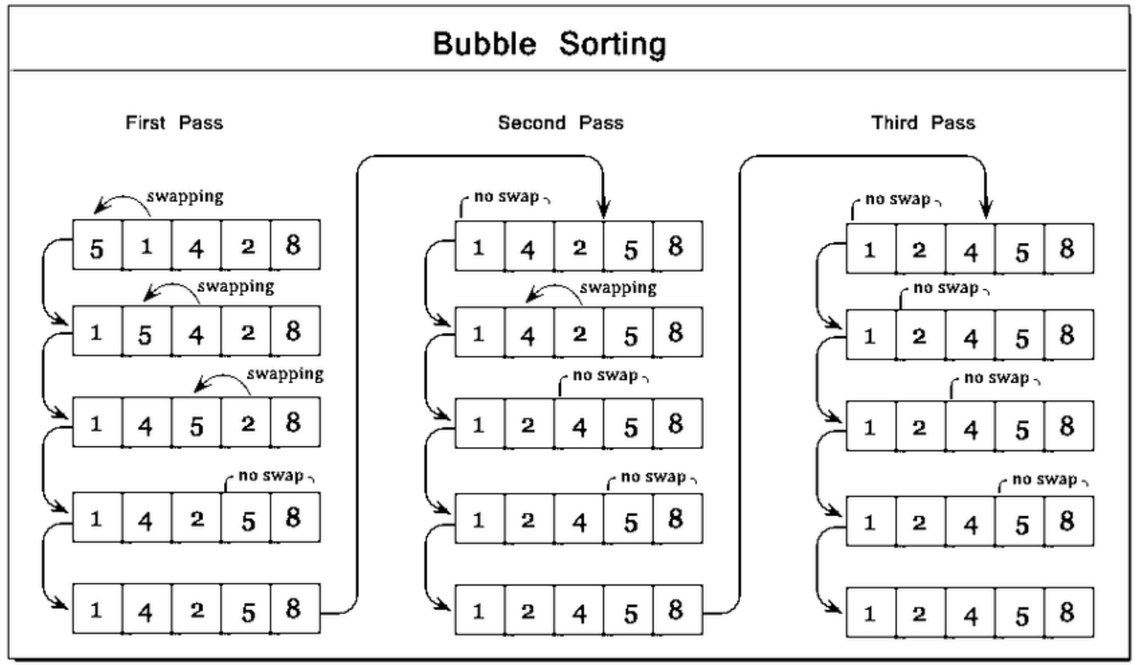
- **Selection Sort:** Find minimum, swap with the first position



Remember that we can find the maximum and minimum of a sequence with $3n/2$ comparisons !
Can we think of using this fact here ?

Sorting - 1: $O(n^2)$-time Solutions

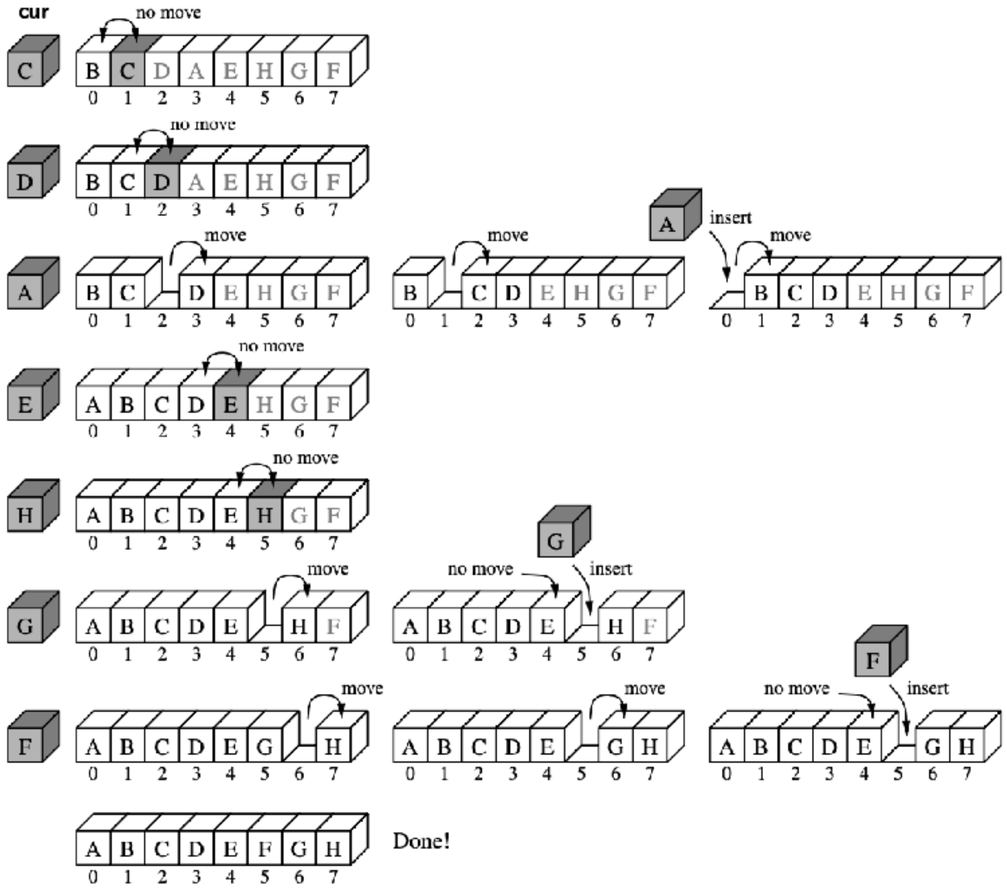
- **Bubble Sort:** Move max to the right most position via *bubbling*



Which one you prefer, bubble or selection sort? Why?

Sorting - 1: $O(n^2)$-time Solutions

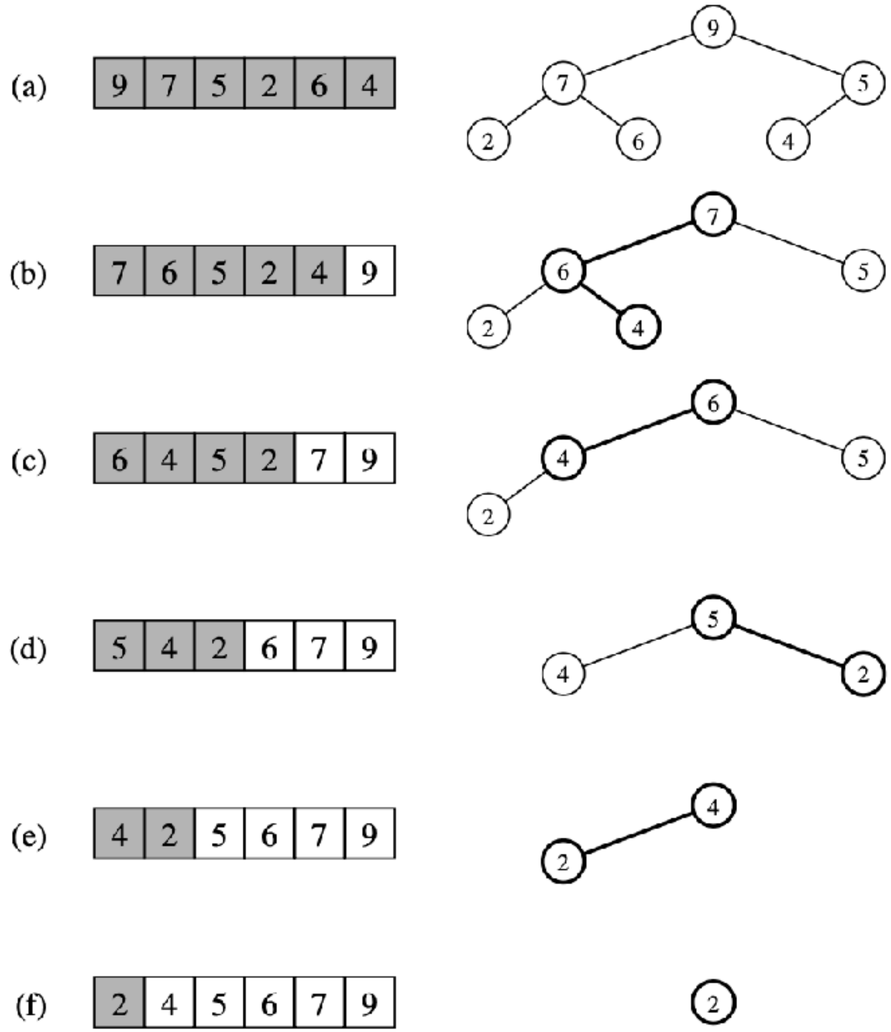
- **Insertion Sort:** Insert A[k+1] into correct position assuming A[1..k] is sorted.



Highly preferred in hybrid (?) sorting solutions.

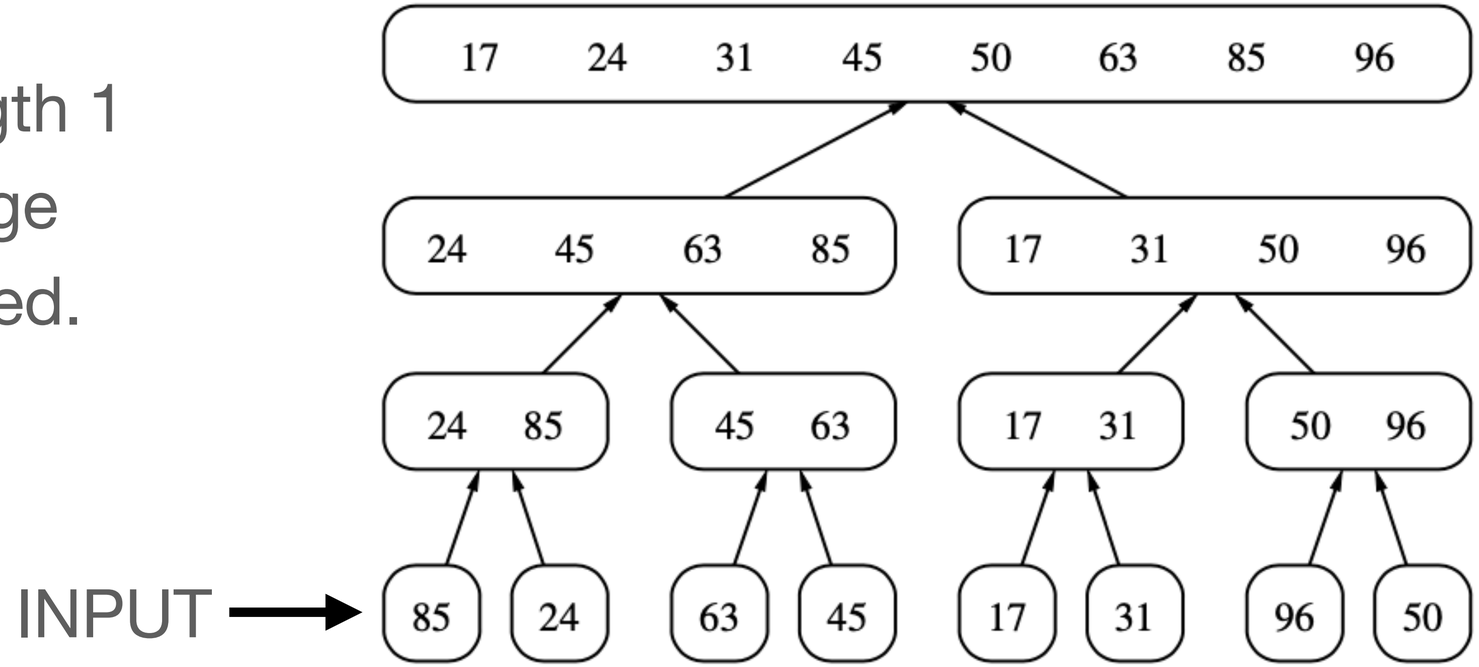
Sorting - 2: $O(n \log n)$-time Solutions

- **Heap Sort:** Mainly, the selection sort via a heap data structure.

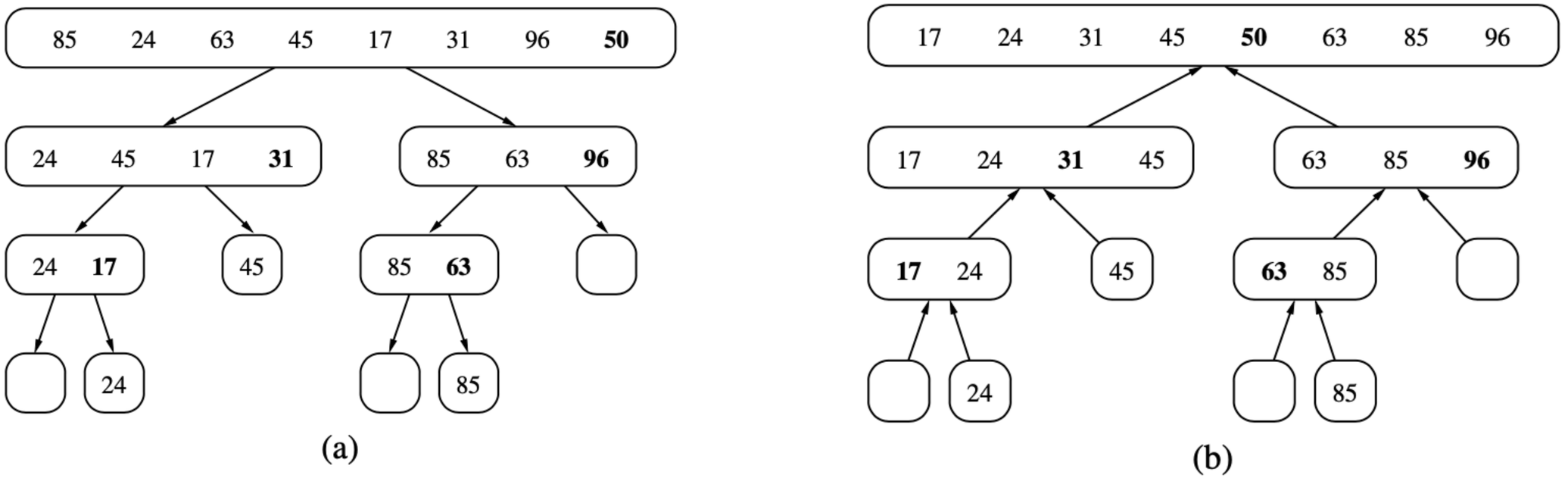


Sorting - 2: $O(n \log n)$-time Solutions

- **Merge Sort:** Start with length 1 lists, and successively merge sorted lists until all are sorted.

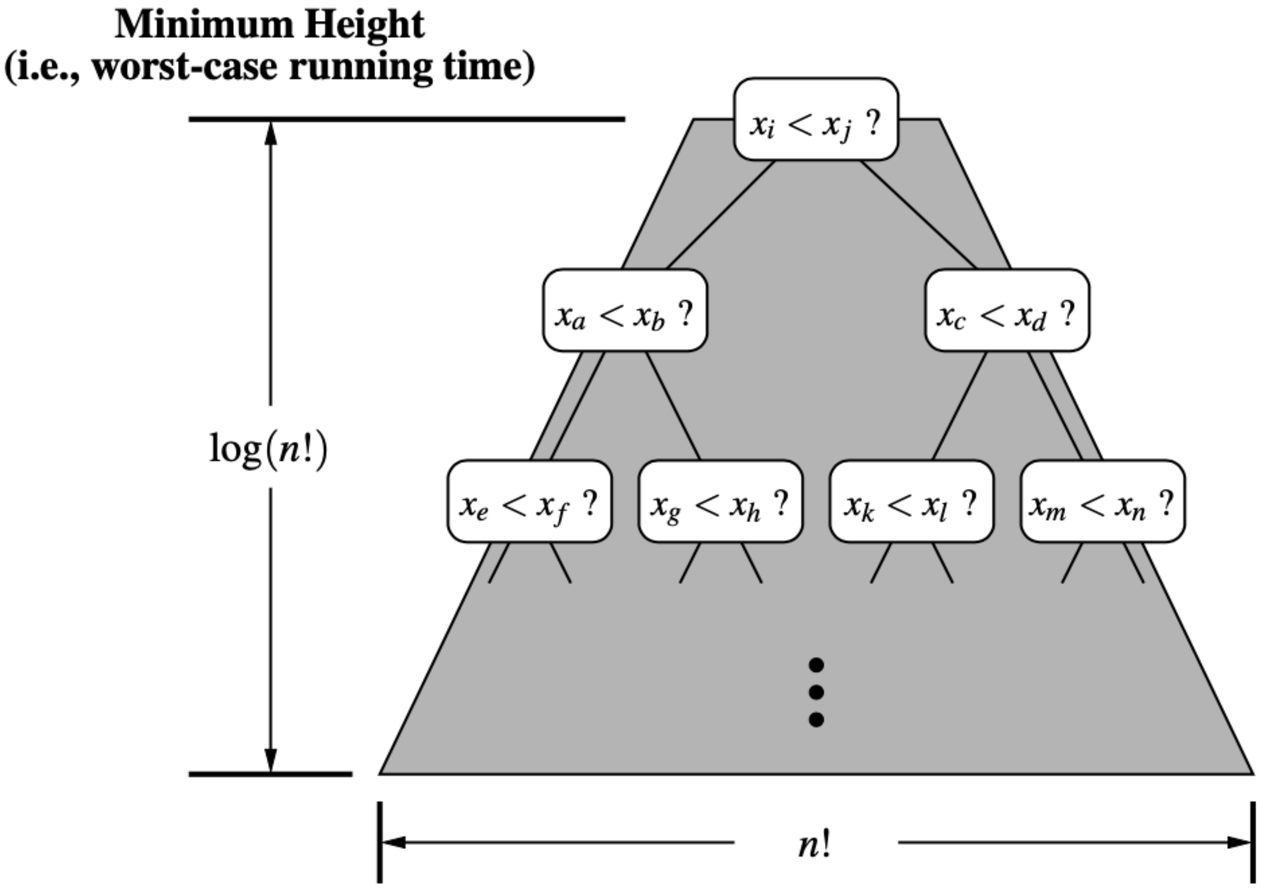


Sorting - 2: $O(n \log n)$-time Solutions



- **Quick Sort:** Choose the last element as the pivot, and divide the list into two according to the pivot. Repeat until left with single elements.
- Worst case is $O(n^2)$, best case $O(n \log n)$
- There is randomized quick sort to make it $O(n \log n)$ **average** expected time.

$O(n \log n)$-time is optimal !



There are n! different permutations of the input sequence each of which maps to a single sorted sequence.

Assume a decision tree such that each inner node defines a comparison with yes/no answer.

Each path from the root to a leaf defines a possible sorting.

Such a path is of length $\log n!$, which i $O(n \log n)$.

Sorting - 3: $O(n)$-time Solutions

- **These are not comparison-based sort algorithms, so not limited with the $O(n \log n)$ bound, but…**
- **Bucket Sort:** Once we know the all possibilities of items in the input, e.g., integers from 5 to 20, then a list is constructed including space for all possibilities. Pass over the input and put the items into corresponding buckets (akin to a frequency table)

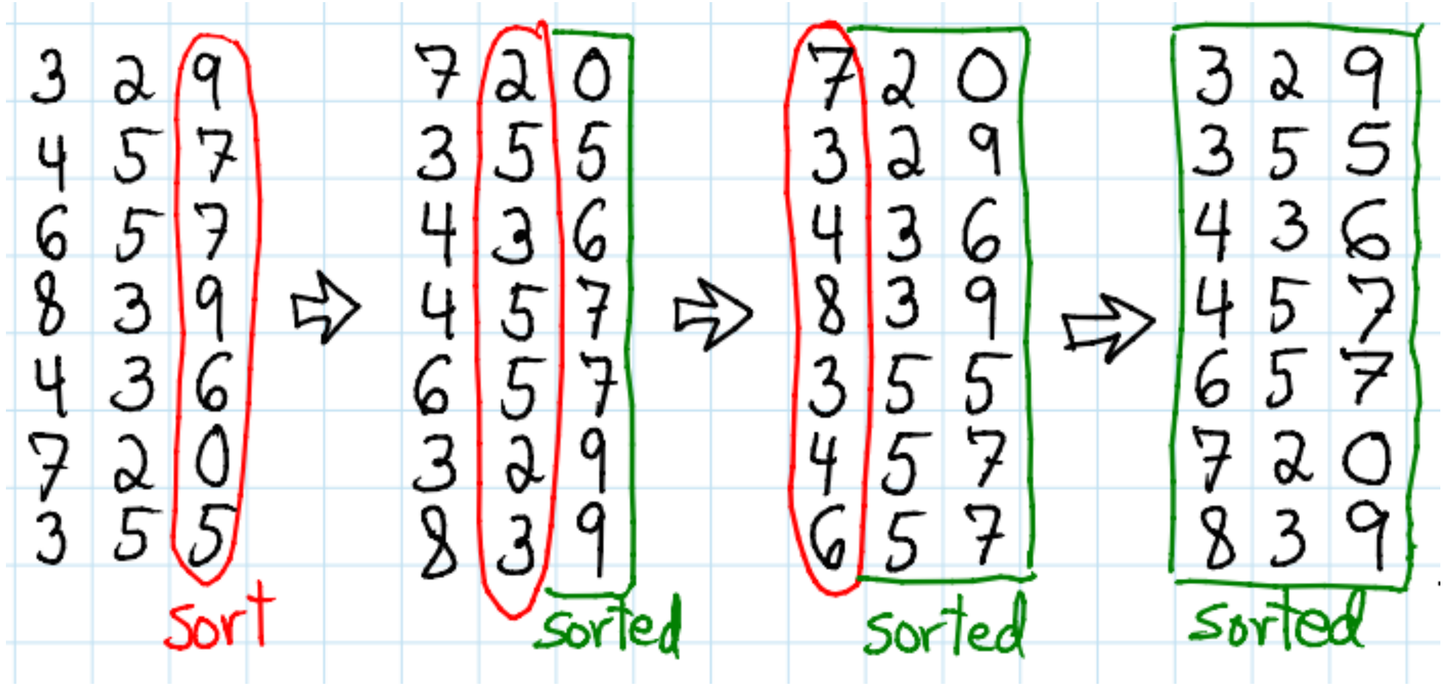
6	9	19	17	14	13	12	10

5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	X			X	X		X	X	X			X		X	

Sorting - 3: $O(n)$-time Solutions

- **These are not comparison-based sort algorithms, so not limited with the $O(n \log n)$ bound, but…**

- **Radix Sort:** Sort according from the least significant position to most frequent.



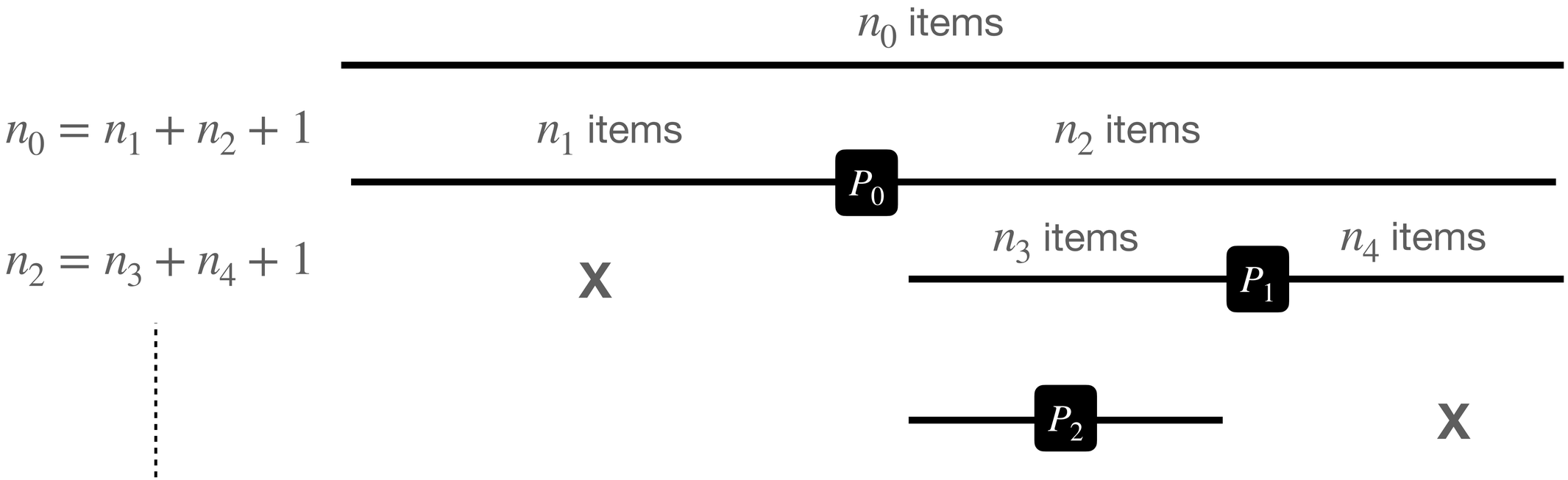
- We may think each sorting of the digit position as a bucket sort.
- The complexity is $O(n \cdot k)$ where k is the number of digits.
- Assuming k is constant, it becomes $O(n)$

Selection

Find the kth smallest or largest item on a queried range of a given list (range quantile queries)

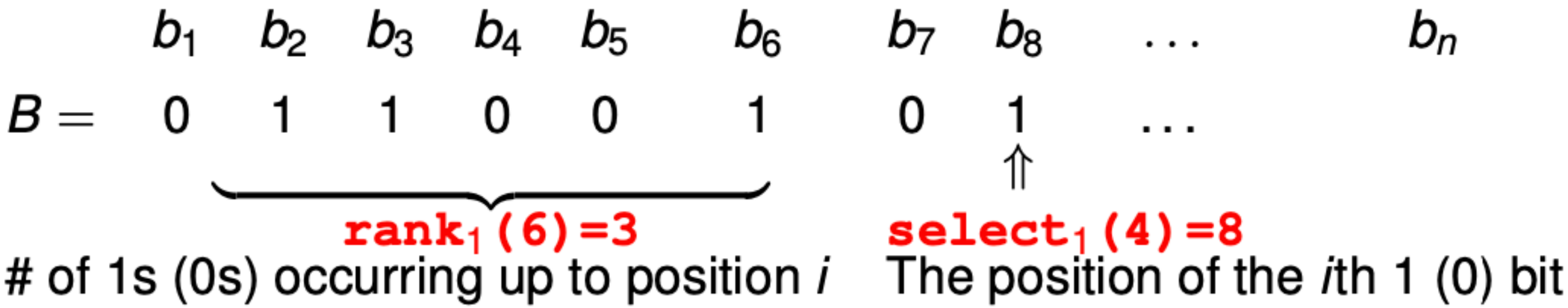
- Similar to top-k queries, frequently used in practical applications.
- Just sort everything and return the desired item, $O(n \log n)$-time
- A better solution is via Quick Select in $O(n)$-time (*average*)
- And yet another elegant solution via Rank/Select dictionaries and Wavelet trees in $O(\log \sigma)$-time on a list of n items each of which is represented by $\log \sigma$ bits.

Quick Select



- Like quick-sort, with one difference that we recurse only on one the left or right of pivot.
- Assume $n_0 = 100$, $n_1 = 45$, $n_2 = 54$, $n_3 = 15$, $n_4 = 38$, and we are searching for the 57th smallest element. The story is…
- Overall, number of **expected** comparisons is $n + \frac{n}{2} + \frac{n}{4} + \ldots \approx 2n \in O(n)$

Rank & Select Queries

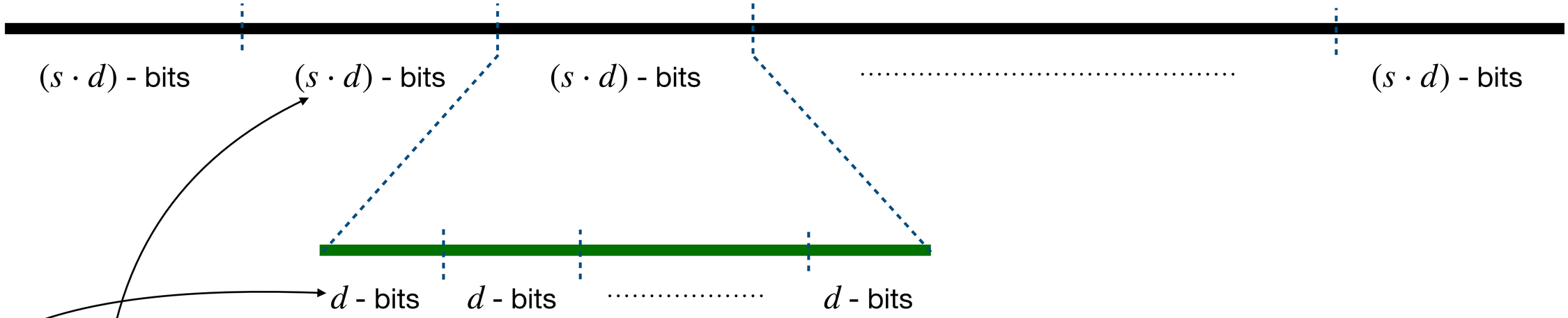


- The fundamental building block in compressed data structures.
- Deeply studied for more than 30 years

  (have a look to http://www.stringology.org/event/2009/psc09p01_presentation.pdf)

Rank&Select Dictionaries

Maintain a dictionary of rank values for some positions and use it to answer queries efficiently.



$n/(s \,.\, d)$ superblocks, each with $\log n$ bits **absolute** rank value

n/d blocks, each with $\log(s \cdot d)$ bits **relative** (to its superblock) rank value

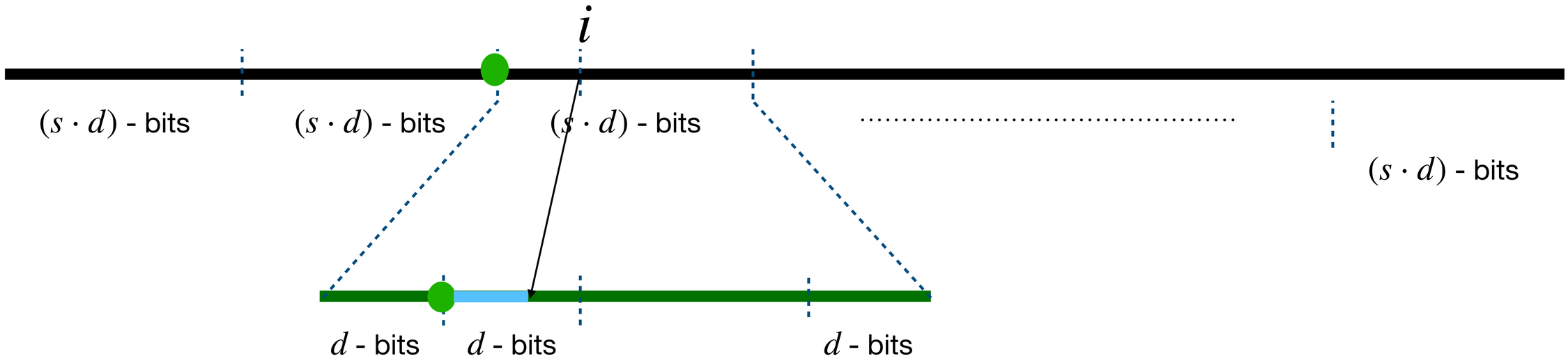
Total overhead

$$n \cdot \left[\frac{\log(s \cdot d)}{d} + \frac{\log n}{s \cdot d}\right] = o(n)$$

,which becomes $o(n)$ with proper selection of s and d

$d = \frac{\log n}{2}$ and $s = c \cdot \log n$

Rank&Select Dictionaries



`Rank(i):`

- Sum the corresponding superblock and block rank values ● from the maintained dictionary
- Add the number of set bits detected inside the inner-block up to the queried position ▬
    - **SIMD instructions are used to compute this value fast in constant time.**

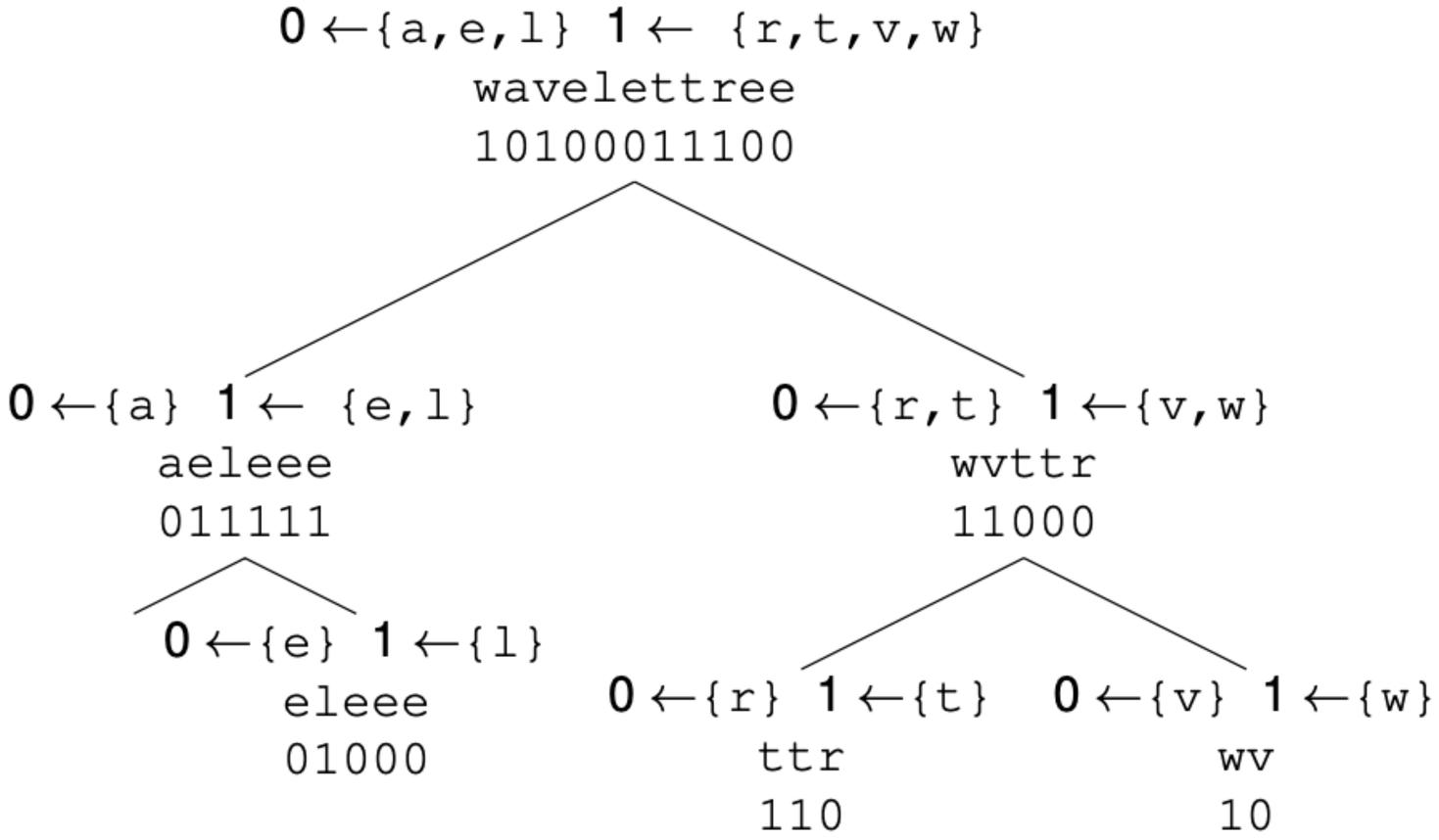
Overall, $O(1)$-time solution for rank with $o(n)$ overhead.

`Select(i):`

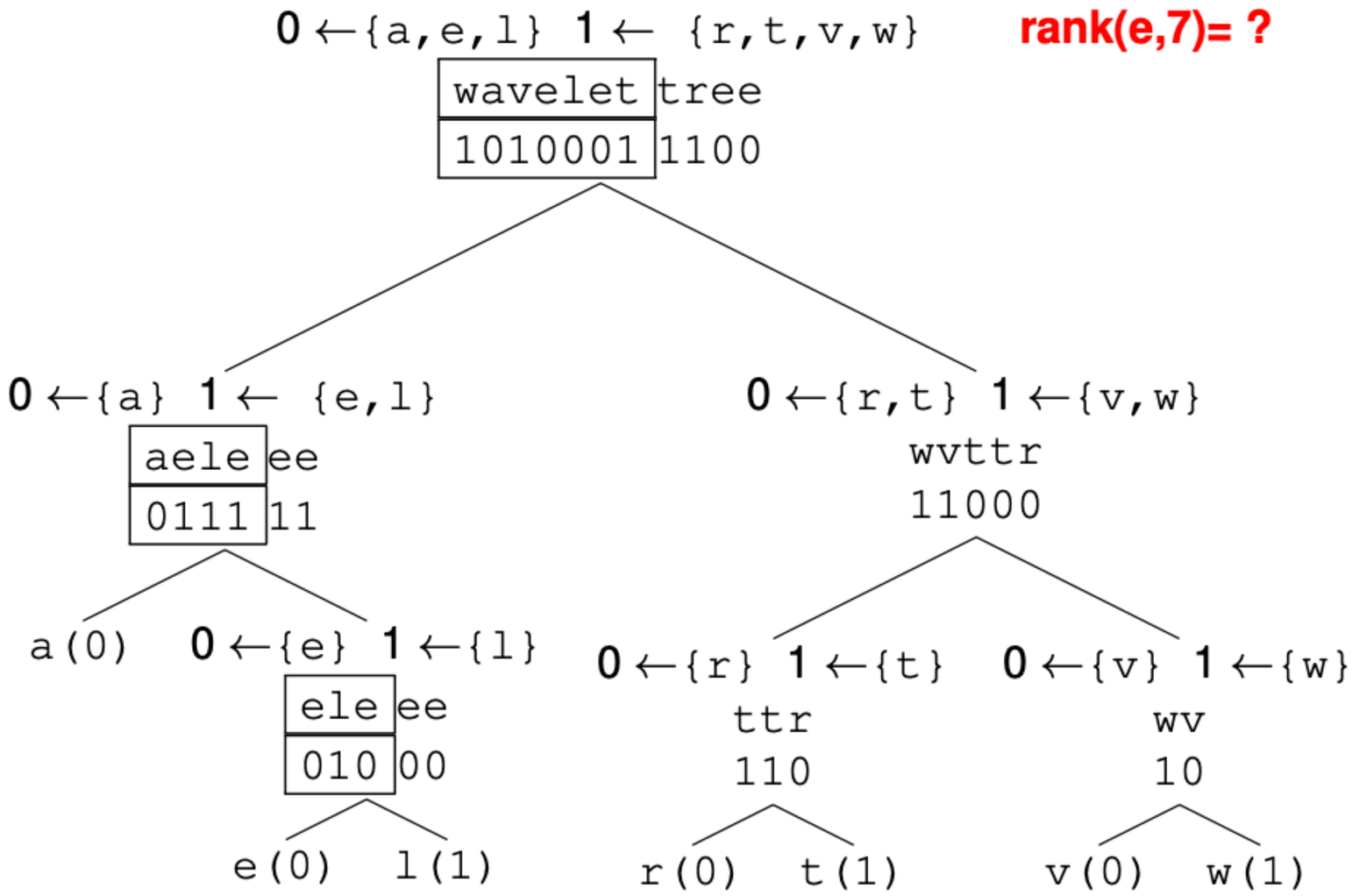
- Yes, it needs a different data structure with variable size blocks to answer in constant time.

Wavelet Trees

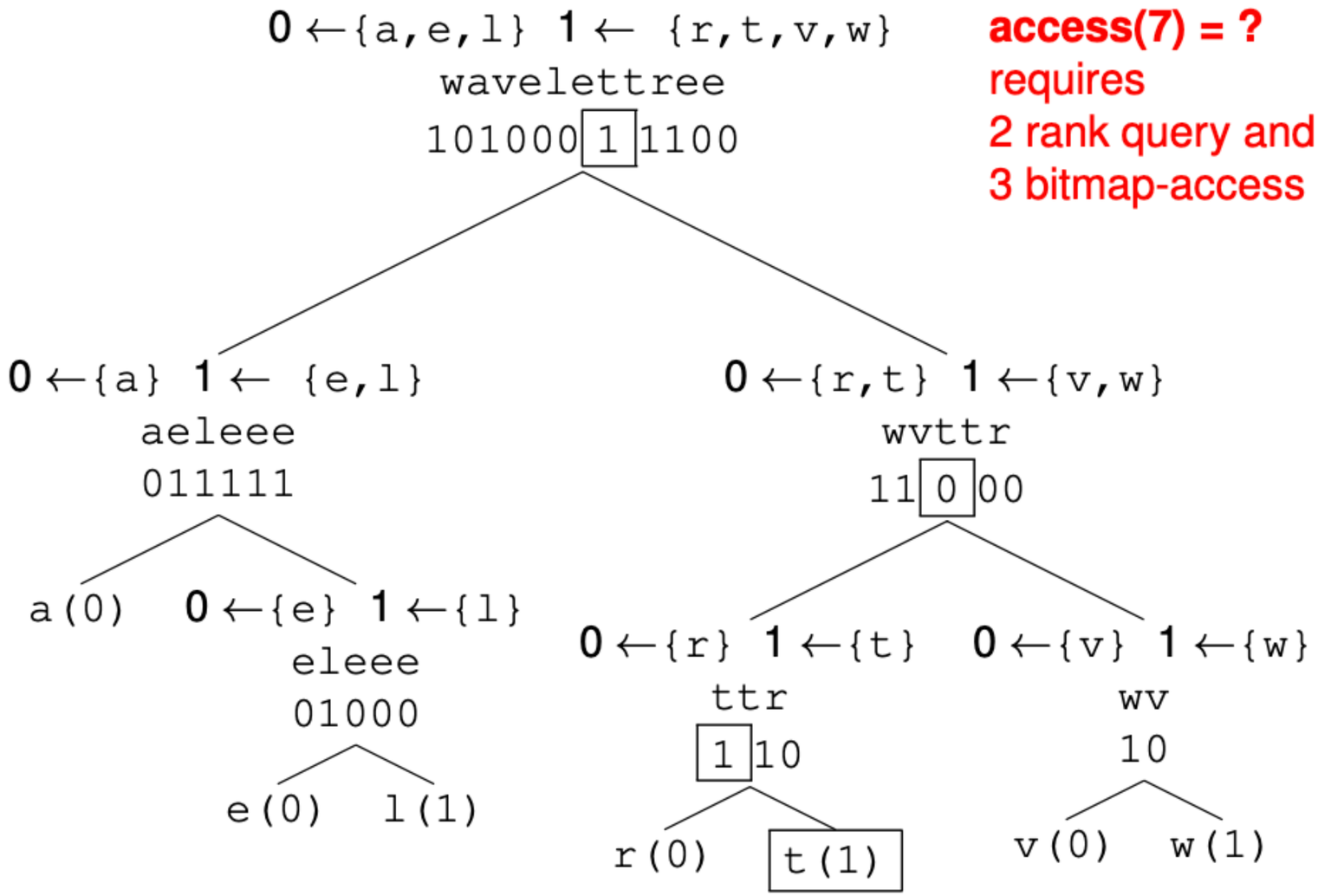
We can answer rank quarries on binary alphabet, but how about on larger alphabets



Wavelet Trees



Wavelet Trees



access(7) = ?
requires
2 rank query and
3 bitmap-access

R/S Dictionaries + Wavelet Trees

Amazing results in the last 20 years …

Highly practical and used in applications.

Overall we have :

Constant time Rank support on a bitmap with $o(n)$ extra space

Constant time Select support on a bitmap with $o(n)$ extra space

Wavelet trees support R/S within $O(\log n)$ time

Next lecture we will see range quantile queries with R/S+Wavelet Trees

Predecessor/Successor Query support

Some cute integer representations allowing random access while keeping space usage small

Reading assignment

- Read the sorting chapters from the books
- For the R/S dictionaries and wavelet trees you can refer to resources at http://www.stringology.org/event/2009/psc09p01_presentation.pdf and https://www.sciencedirect.com/science/article/pii/S1570866713000610
- For the implementations of such succinct data structures library check http://algo2.iti.kit.edu/gog/docs/html/index.html , if you are interested