Lecture 3: Lower Bounds for Sorting, Linear Time Sorting Algorithms

Instructor: Saravanan Thirumuruganathan

Outline

1. Lower bound for Comparison based Sorting Algorithms
   - Concept of Lower Bounds
   - Decision tree model of complexity
2. Linear Time Non-Comparison based Sorting Algorithms

In-Class Quizzes

- **URL:** `http://m.socrative.com/`
- **Room Name:** **4f2bb99e**

Sorting Problem

- **Input:** A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
- **Output:** A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$
- **Example:** $\langle 4, 2, 1, 3, 5 \rangle$ to $\langle 1, 2, 3, 4, 5 \rangle$
- Assume distinct values (doesn't affect correctness or analysis)

Sorting Algorithms - So Far

- We started with elementary algorithms (Bubble, Selection, Insertion) that were $O(n^2)$ in worst case
- Using some clever ideas (such as D&C), we have some algorithms (Merge and Quick) that are $O(n \lg n)$
- Question: Can we do better? Or have we hit some fundamental computational limit?

One Slide Summary

- Comparison based sorting algorithms require $\Omega(n \lg n)$
- In other words, for some input every comparison based sorting algorithms will require $\Omega(n \lg n)$
- Result does not apply for non-comparison based algorithms which can be more efficient under some circumstances

Lower Bound of an Algorithm[1]

- Algorithm A has a lower bound $\Omega(T(n))$ if there exists an input of size n on which A takes $\Omega(T(n))$ time
- Lower bound is not the same as best case!
- Insertion sort has lower bound $\Omega(n^2)$ (if the array is reverse sorted)
- Merge sort has lower bound $\Omega(n \lg n)$ (for reverse sorted array and many other inputs)
- Finding lower bound of an algorithm is relatively easy

[1] Slides from MCS/CS 401 slides by Roy M. Lowman

Lower Bound of a Problem

- Problem P has a lower bound $\Omega(T(n))$ if for **every** algorithm A that solves P, there exists an input of size n on which A takes $\Omega(T(n))$ time
- In general, very hard - has to hold for *all* algorithms (even those humans have not invented yet)
- Lower bounds known for very few problems!
- Typical Strategy: Restrict computation model

Lower Bound of a Problem

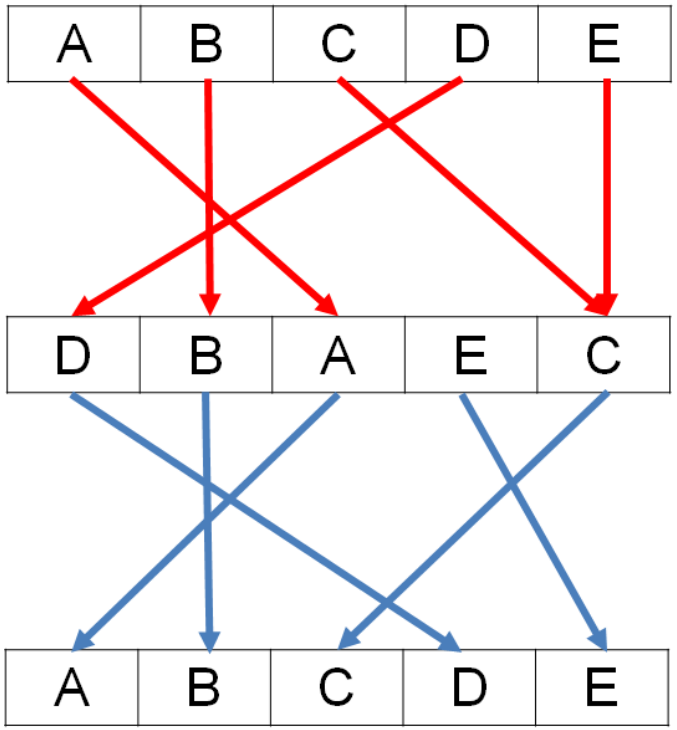
- $O(f(n))$ - upper bound
- $\Omega(g(n))$ - lower bound
- If $f(n)$ is not $g(n)$, then there is an "efficiency gap" - Opportunity for improvement!
- For example, $O(n^2)$ for insertion sort versus $\Omega(n \lg n)$ for sorting

Lower Bounds for Sorting

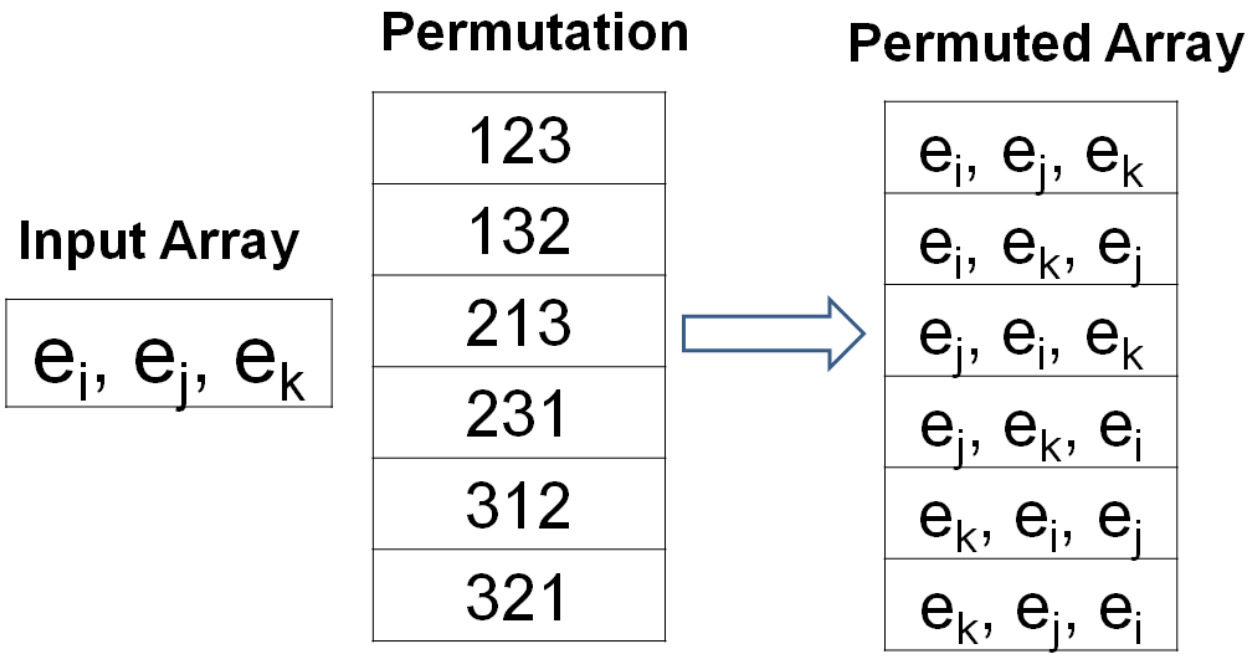
Theorem: Any Algorithm that sorts by **only comparing** elements must take $\Omega(n \lg n)$ time in the worst case.

Sorting As Permutation

- Sorting is equivalent to finding a permutation
- Find the **inverse** permutation (among all permutations) that will undo the original permutation



Sorting As Permutation



Quiz!

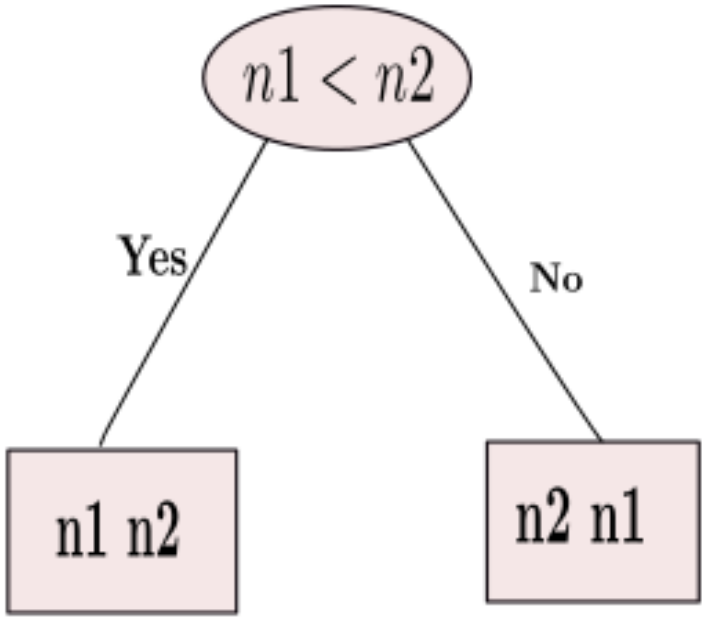
- Suppose you an array A with elements $\{1, 2, 3, 4, 5\}$. How many different arrays with distinct elements are possible?

Quiz!

Suppose you an array A with elements $\{1, 2, 3, 4, 5\}$. How many different arrays with distinct elements are possible?

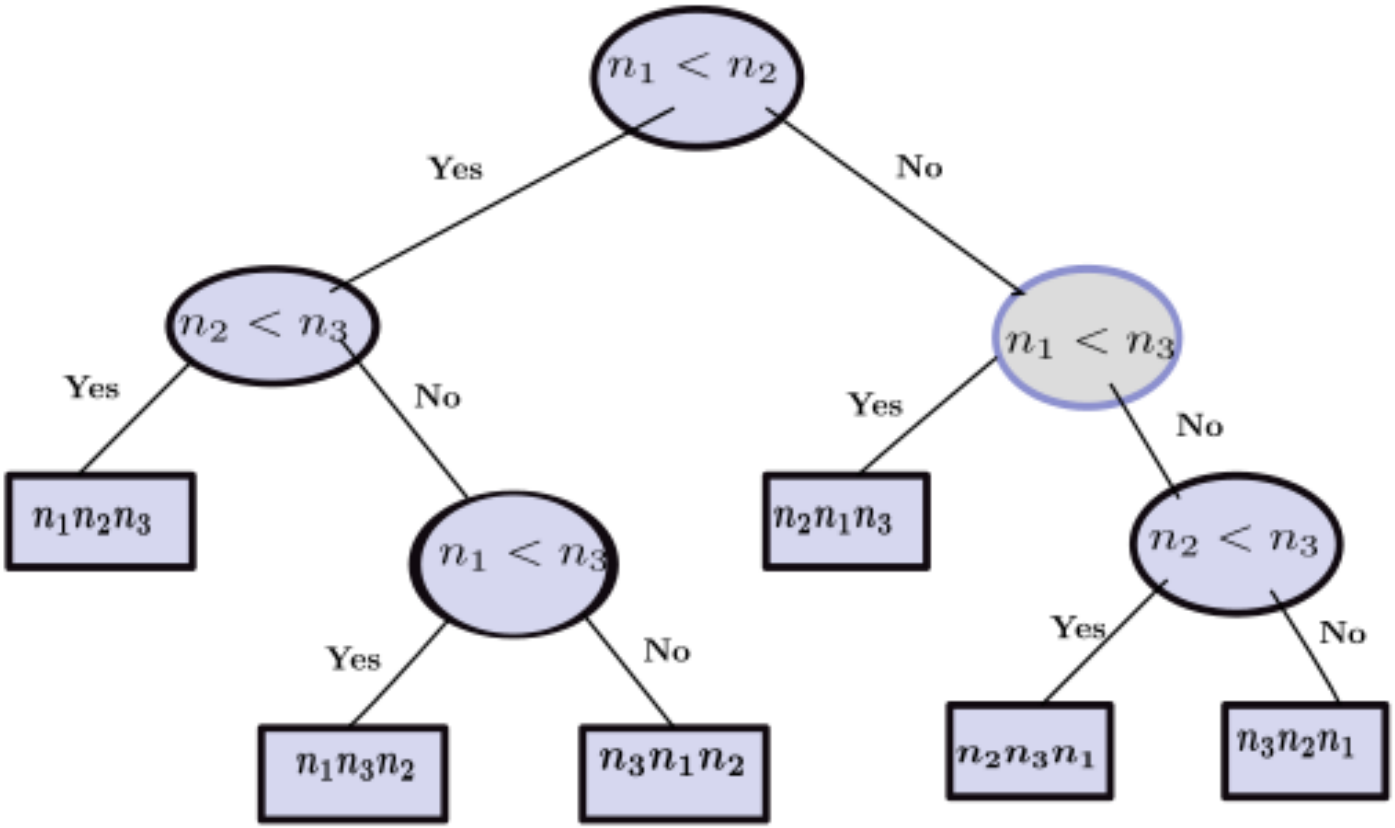
- You can choose any of 5 elements for first position
- You can choose any of 4 elements (except the one chosen previously) for second position
- And so on until you are left with no choice for last element
- There are $5 \times 4 \times 3 \times 2 \times 1 = 5! = 120$ different arrays

Decision Tree Model for Sorting[2]



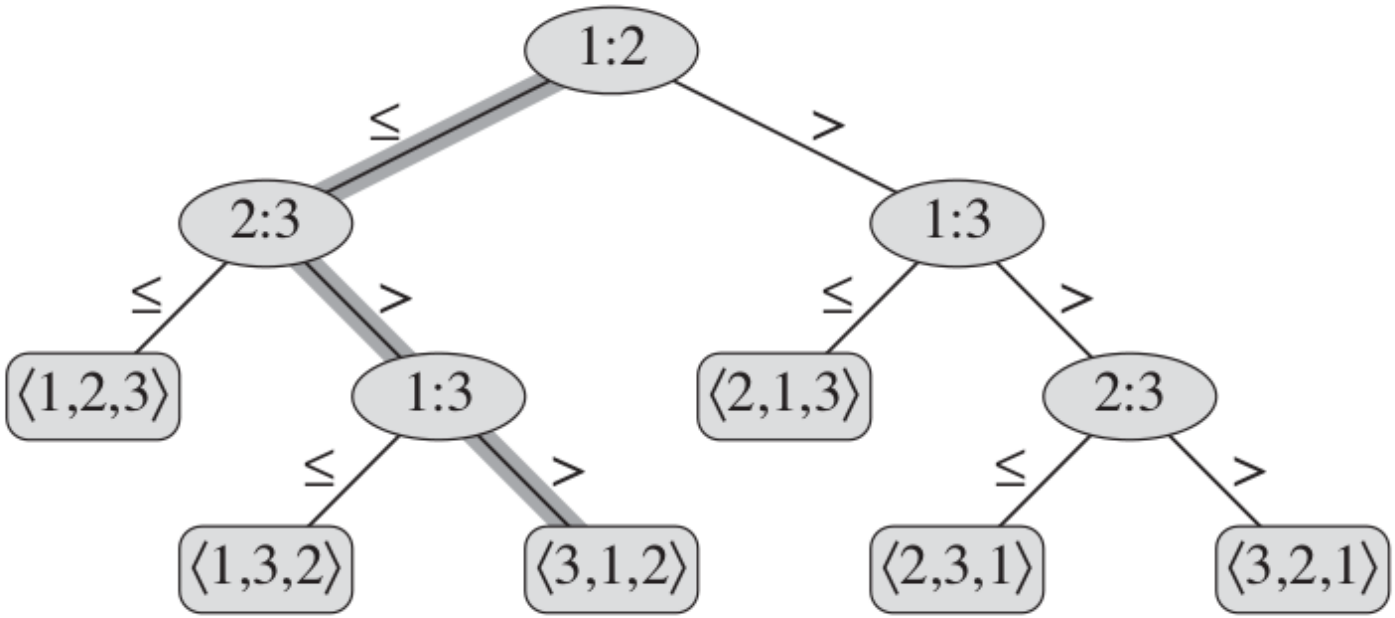
[2]Figures from MCS/CS 401 slides by Roy M. Lowman

Decision Tree Model for Sorting[3]



[3]Figures from MCS/CS 401 slides by Roy M. Lowman

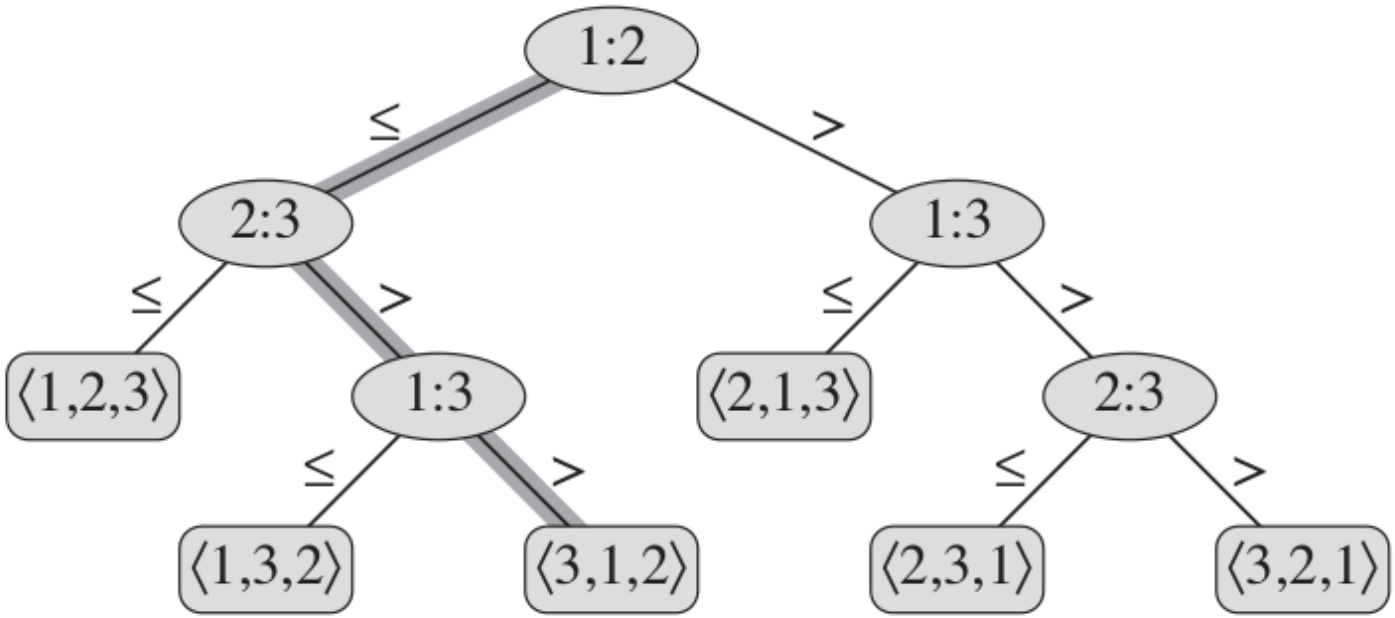
Decision Tree Model for Sorting[4]



[4]Figures from CLRS

Decision Tree Model for Sorting[5]

Sorting $\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$



[5] Figures from CLRS

Decision Tree Model for Sorting

- *Every* comparison based sort algorithms can be modelled as a decision tree
- Non-leaf (internal) nodes represent comparisons
- Leaf nodes provide the permutation that will result in the sorted array

Sorting Lower Bounds

- Number of leaves in the decision tree must be at least $n!$ (why?)
- Idea: Use $n!$ to get the lower bound
- Height of a leaf node represents number of **comparisons** made to sort a particular data represented by that node
- Interpretation of Longest path length and Average path length?

Sorting Lower Bounds

Theorem: Any Algorithm that sorts by **only comparing** elements must take $\Omega(n \lg n)$ time in the worst case.

Proof:

- The tree has at least $n!$ leaves
- A binary with height h has $\leq 2^h$ leaves

$$n! \leq 2^h$$

$$\lg n! \leq \lg 2^h = h \lg 2 = h$$

$$h \geq \lg(n!)$$

$$= \lg \prod_{i=1}^{n} i = \sum_{i=1}^{n} \lg i$$

$$= \Theta(n \lg n)$$

$$h = \Omega(n \lg n)$$

Sorting Lower Bounds

- Any **comparison** based sorting algorithms must take $\Omega(n \lg n)$ time in the worst case.
- Has major theoretical and practical implications

Linear Time Sorting Algorithms

- Non-comparison based sorting algorithms
- Side step the $\Omega(n \lg n)$ barrier
- Counting, Radix and Bucket sort

Counting Sort - Intuition

- Suppose you are given a binary string (say 01110001) and you want to sort it (to 00001111).
- Which algorithm would you use?
  - Bubble, Selection, Insertion sort
  - Merge and Quick sort

Counting Sort - Intuition

- What about decimal numbers?
- What about text with ASCII characters?
- Handling complex objects and stability

Counting Sort

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Counting Sort

- For every key j, $1 \leq j \leq k$, store its frequency in $C[j]$
- Then, change $C[j]$ to mean the number of keys $\leq j$
- Use a clever trick to move items to the right place by traversing the array right to left
- This trick is needed to ensure Stability - needed for Radix sort

Counting Sort - Visualization

- See URL `http://www.cs.miami.edu/~burt/learning/Csc517.101/workbook/countingsort.html`

Counting Sort - Analysis

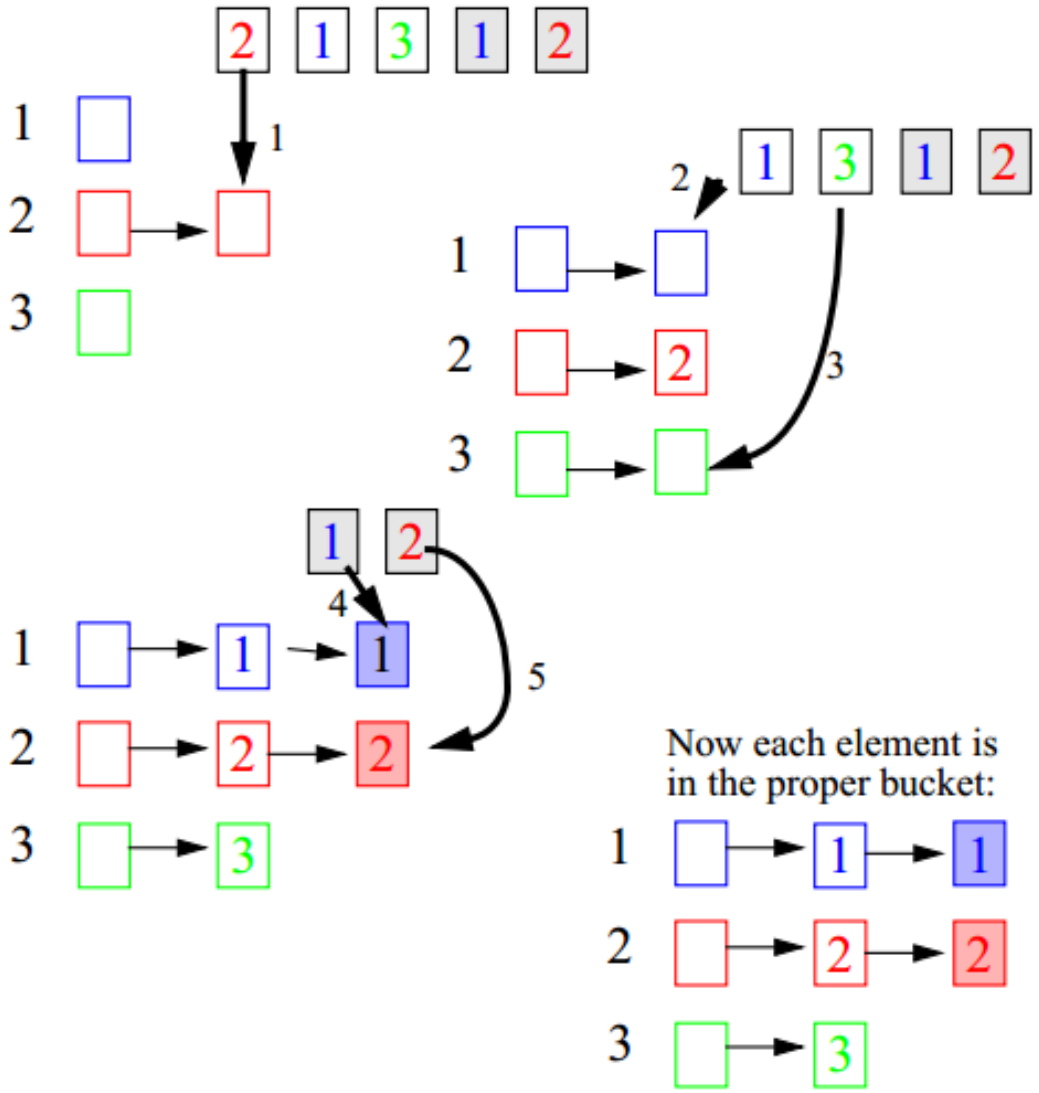
- Time Complexity : $O(n + k) = O(n)$ - Why?
- Count sort is Stable - Why?
- What prevents us from always using Counting sort?

Counting Sort - Results[6]

- Interesting Result 1:
  - Suppose you want to sort an array A with n elements. The keys are integers in the range of $1, \ldots, m$ where $m = O(n)$
  - You can sort A in $O(n)$ time
- Interesting Result 2:
  - For any *constant* k, we can sort n integers in the range $\{1, \ldots, n^k\}$ in $O(n)$ time
  - Hint: Combine Counting sort with Radix sort

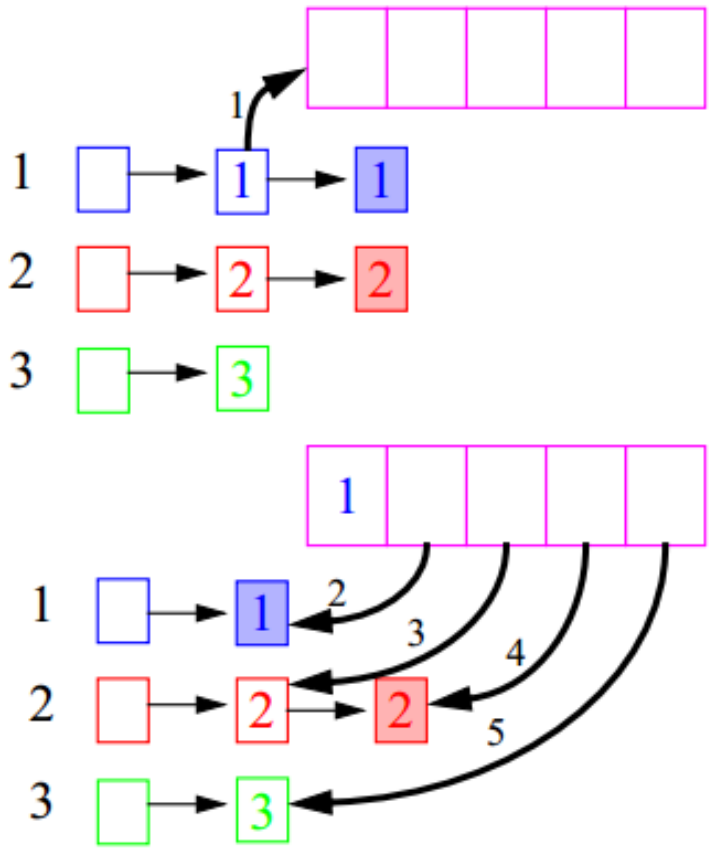
[6] `http://www.inf.ed.ac.uk/teaching/courses/ads/Lects/lecture8.pdf`

Bucket Sort[7]

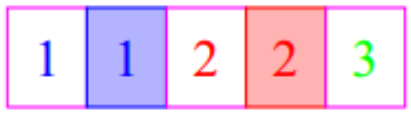


[7] http://www.dcs.gla.ac.uk/~pat/52233/slides/RadixSort1x1.pdf

Bucket Sort[8]



At last, the sorted array (sorted in a stable way):



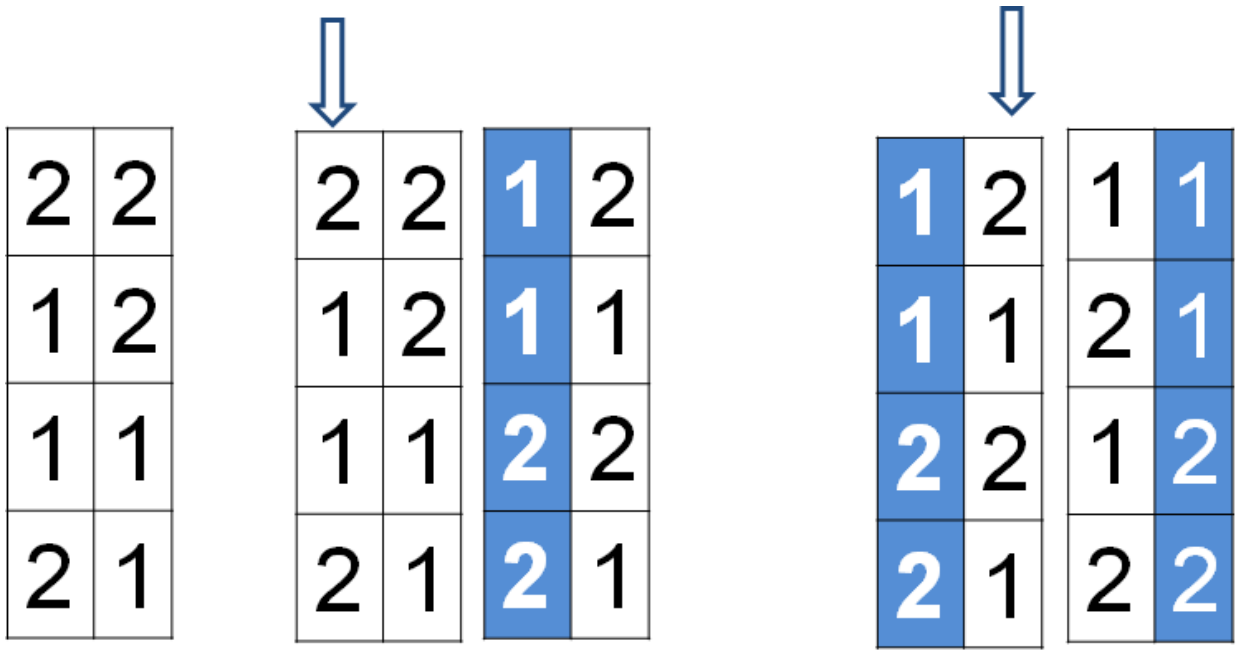
[8] http://www.dcs.gla.ac.uk/~pat/52233/slides/RadixSort1x1.pdf

Radix Sort[9]

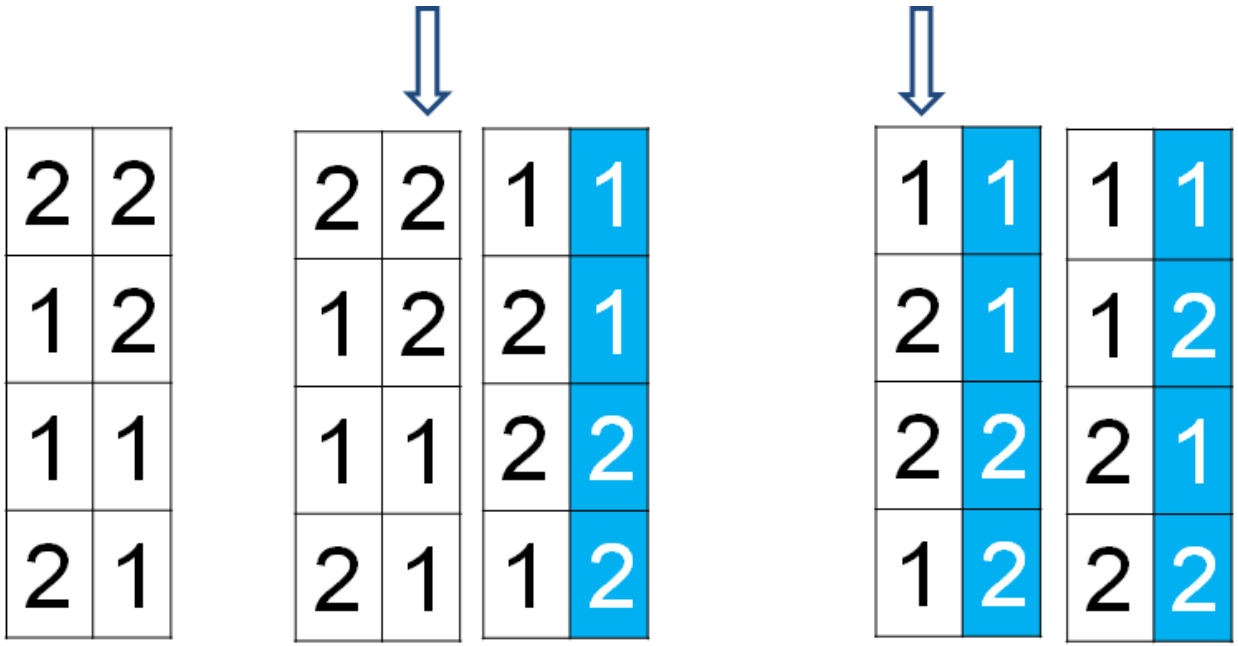
- Keys are sequences of *digits* in a fixed range $1, \ldots, k$, all of equal length d
- Applications:
  - Sorting phone numbers by Area Code
  - Sorting dates by Day, Month and Year (for e.g. Apache logs)
  - Sort courses by course name and number
  - Sorting social security number

[9] `http://www.inf.ed.ac.uk/teaching/courses/ads/Lects/lecture8.pdf`

Radix Sort - MSD Issues



Radix Sort - LSD Idea



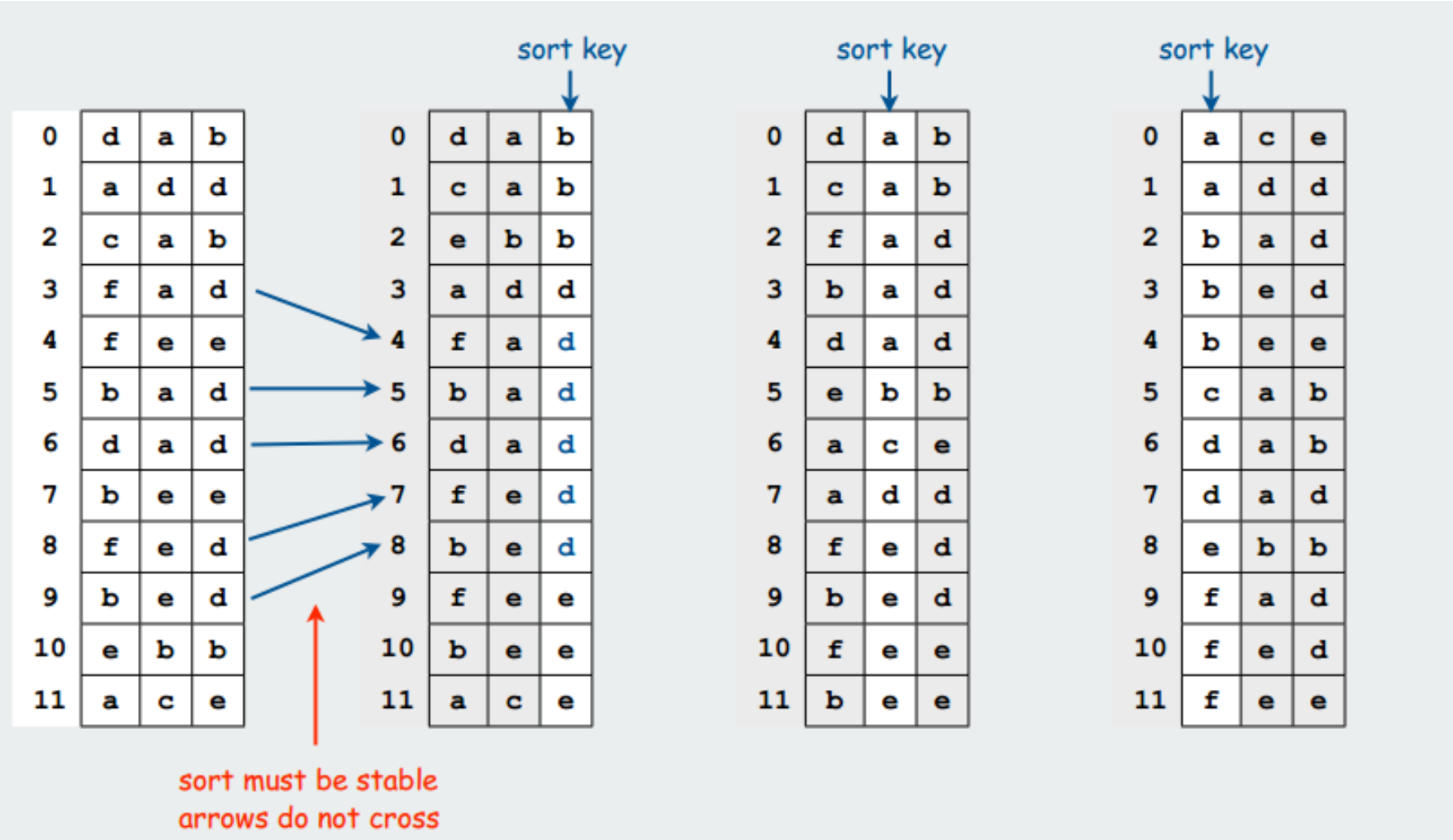
Radix Sort

- LSD Radix Sort
- MSD Radix Sort

LSD Radix Sort - PseudoCode

```
Radix-Sort(A, d)
    for i = 1 to d
        use a stable sort to sort array A on digit i
```

LSD Radix Sort - Example[10]



0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

0	d	a	b
1	c	a	b
2	e	b	b
3	a	d	d
4	f	a	d
5	b	a	d
6	d	a	d
7	f	e	d
8	b	e	d
9	f	e	e
10	b	e	e
11	a	c	e

0	d	a	b
1	c	a	b
2	f	a	d
3	b	a	d
4	d	a	d
5	e	b	b
6	a	c	e
7	a	d	d
8	f	e	d
9	b	e	d
10	f	e	e
11	b	e	e

0	a	c	e
1	a	d	d
2	b	a	d
3	b	e	d
4	b	e	e
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	d
11	f	e	e

[10] http://www.cs.princeton.edu/~rs/AlgsDS07/18RadixSort.pdf

LSD Radix Sort - Why it works[11]

- Suppose you sorted the array based on LSD (digit 1)
  - If two numbers **differ** on first digit, Counting sort puts them in correct relative order
  - If two numbers **agree** on first digit, stability keeps in correct relative order
- Consider the next step - sorting digit 2
  - If two numbers **differ** on second digit (or other digits), it doesn't matter what we do now
  - If two numbers **agree** on second digit (or other digits), stability ensures later passes wont cause any issues

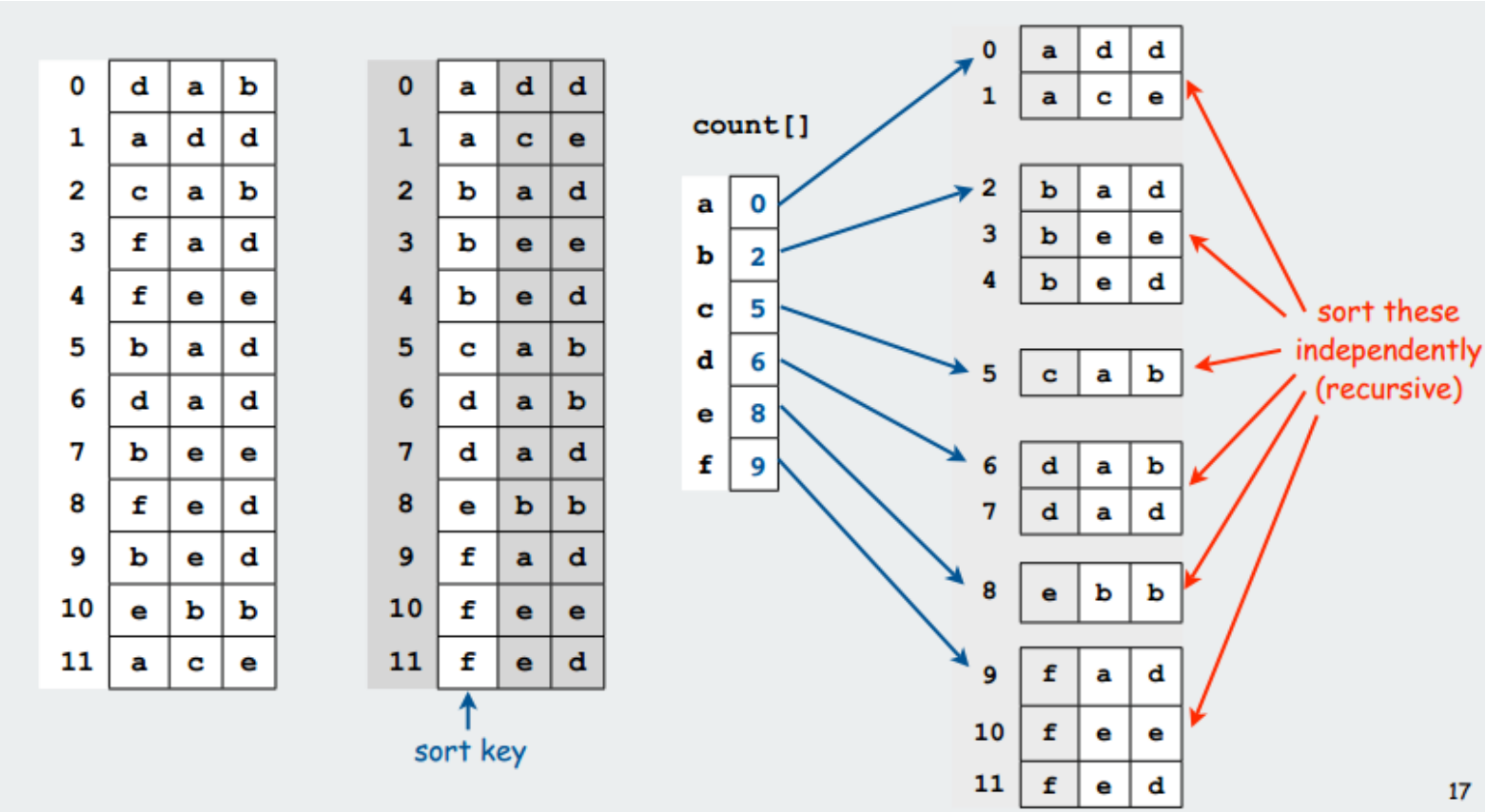
[11] `http://www.cs.princeton.edu/~rs/AlgsDS07/18RadixSort.pdf`

LSD Radix Sort - Bucket Sort Perspective

- See URL: `http://www.cs.umanitoba.ca/~chrisib/teaching/comp2140/notes/003e_radixSort.pdf`

MSD Radix Sort[12]

- Partition array to k pieces based on first digit
- **Recursively** sort array



[12] http://www.cs.princeton.edu/~rs/AlgsDS07/18RadixSort.pdf

LSD Radix Sort - Analysis

- Given n d-digit numbers in which each digit can take up to k possible values
- Radix-Sort correctly sorts them in $O(d(n+k))$ time
- **if** the stable sort takes $O(n+k)$ time.

Summary

Major Concepts:

- Concept of Lower bounds
- Lower bounds for Comparison based Sorting Algorithms
- Decision tree model for Complexity Analysis
- Linear Time Sorting Algorithms