MCAC 302: Design and Analysis of Algorithms

Neelima Gupta

ngupta@cs.du.ac.in

September 7, 2020

Comparison Sorts

In comparison based sorting, values of the elements do not matter; it is only their relative ordering that matters. Thus an algorithm takes roughly the same amount of time on $< 2, 3, 1, 9, 8 >$ and on $< 12, 112, 1112, 11112, 20000 >$.

Comparison Sorts

In comparison based sorting, values of the elements do not matter; it is only their relative ordering that matters. Thus an algorithm takes roughly the same amount of time on $< 2, 3, 1, 9, 8 >$ and on $< 12, 112, 1112, 11112, 20000 >$.

- What is the best that any Comparison Sort can do in the worst case?

Comparison Sorts

In comparison based sorting, values of the elements do not matter; it is only their relative ordering that matters. Thus an algorithm takes roughly the same amount of time on $< 2, 3, 1, 9, 8 >$ and on $< 12, 112, 1112, 11112, 20000 >$.

- What is the best that any Comparison Sort can do in the worst case?
- We will show that we can not do better than $(n \log n)$. That is, it is $\Omega(n \log n)$.

Comparison Sorts

In comparison based sorting, values of the elements do not matter; it is only their relative ordering that matters. Thus an algorithm takes roughly the same amount of time on $< 2, 3, 1, 9, 8 >$ and on $< 12, 112, 1112, 11112, 20000 >$.

- What is the best that any Comparison Sort can do in the worst case?
- We will show that we can not do better than $(n \log n)$. That is, it is $\Omega(n \log n)$.
- What is the best that any Comparison Sort can do in the best case?

Comparison Sorts

In comparison based sorting, values of the elements do not matter; it is only their relative ordering that matters. Thus an algorithm takes roughly the same amount of time on $< 2, 3, 1, 9, 8 >$ and on $< 12, 112, 1112, 11112, 20000 >$.

- What is the best that any Comparison Sort can do in the worst case?
- We will show that we can not do better than $(n \log n)$. That is, it is $\Omega(n \log n)$.
- What is the best that any Comparison Sort can do in the best case?
- It is trivial to see that it is $\Omega(n)$.

Decision Trees

- Decision trees provide an abstraction of comparison sorts

[1] Figure from CSLR

Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.

[1] Figure from CSLR

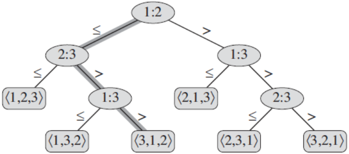
Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.
  - Let's draw the decision tree for Insertion Sort on an input of size 3 i.e. $\{x_1, x_2, x_3\}$

[1] Figure from CSLR

Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.
  - Let's draw the decision tree for Insertion Sort on an input of size 3 i.e. $\{x_1, x_2, x_3\}$

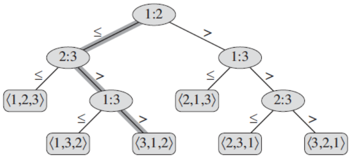


[1]

[1] Figure from CSLR

Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.
  - Let's draw the decision tree for Insertion Sort on an input of size 3 i.e. $\{x_1, x_2, x_3\}$



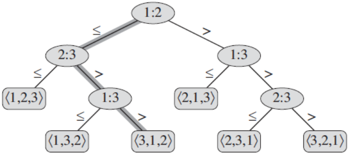
[1]

- Comparison tree is typically a binary tree.

[1]Figure from CSLR

Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.
  - Let's draw the decision tree for Insertion Sort on an input of size 3 i.e. $\{x_1, x_2, x_3\}$



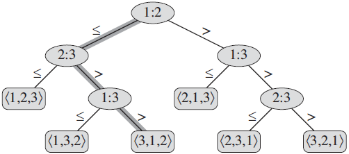
[1]

- Comparison tree is typically a binary tree.
- What do the leaves represent?

[1] Figure from CSLR

Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.
  - Let's draw the decision tree for Insertion Sort on an input of size 3 i.e. $\{x_1, x_2, x_3\}$



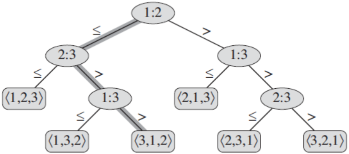
[1]

- Comparison tree is typically a binary tree.
- What do the leaves represent? All possible arrangements of the input elements.

[1] Figure from CSLR

Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.
  - Let's draw the decision tree for Insertion Sort on an input of size 3 i.e. $\{x_1, x_2, x_3\}$



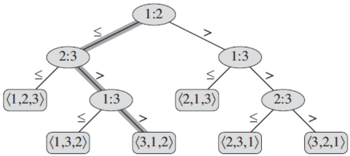
[1]

- Comparison tree is typically a binary tree.
- What do the leaves represent? All possible arrangements of the input elements.
- At least how many leaves must be there?

[1] Figure from CSLR

Decision Trees

- Decision trees provide an abstraction of comparison sorts
  - A decision tree represents the comparisons made by the algorithm. Every thing else ignored.
  - Let's draw the decision tree for Insertion Sort on an input of size 3 i.e. $\{x_1, x_2, x_3\}$



[1]

- Comparison tree is typically a binary tree.
- What do the leaves represent? All possible arrangements of the input elements.
- At least how many leaves must be there? at least $n!$.

[1] Figure from CSLR

Decision Trees contd..

- For a given algorithm:
  - One decision tree for each n

Decision Trees contd..

- For a given algorithm:
  - One decision tree for each n
  - Tree paths are all possible execution traces.

Decision Trees contd..

- For a given algorithm:
  - One decision tree for each n
  - Tree paths are all possible execution traces.
  - What represents the number of comparisons done by the algorithm in the worst case?

Decision Trees contd..

- For a given algorithm:
  - One decision tree for each n
  - Tree paths are all possible execution traces.
  - What represents the number of comparisons done by the algorithm in the worst case? The length of the longest path from the root to a leaf - the height.
- For n elements, what's the length of the longest path in a decision tree for insertion sort?

Decision Trees contd..

- For a given algorithm:
  - One decision tree for each n
  - Tree paths are all possible execution traces.
  - What represents the number of comparisons done by the algorithm in the worst case? The length of the longest path from the root to a leaf - the height.
- For n elements, what's the length of the longest path in a decision tree for insertion sort? For merge sort?

Decision Trees contd..

- For a given algorithm:
  - One decision tree for each n
  - Tree paths are all possible execution traces.
  - What represents the number of comparisons done by the algorithm in the worst case? The length of the longest path from the root to a leaf - the height.
- For n elements, what's the length of the longest path in a decision tree for insertion sort? For merge sort?
- What is the asymptotic height of any decision tree for sorting n elements?

Decision Trees contd..

- For a given algorithm:
  - One decision tree for each n
  - Tree paths are all possible execution traces.
  - What represents the number of comparisons done by the algorithm in the worst case? The length of the longest path from the root to a leaf - the height.
- For n elements, what's the length of the longest path in a decision tree for insertion sort? For merge sort?
- What is the asymptotic height of any decision tree for sorting n elements?
- Answer: We will prove that it is at least $(nlgn)$.

Lower Bound For Comparison Sorting

- Thm: Any decision tree that sorts n elements has height $\Omega(nlgn)$.
  - What's the minimum # of leaves?

Lower Bound For Comparison Sorting

- Thm: Any decision tree that sorts n elements has height $\Omega(n lg n)$.
  - What's the minimum # of leaves? Answer: $n!$. That is, if ℓ is the number of leaves then $\ell \geq n!$.

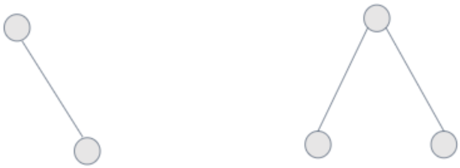
Lower Bound For Comparison Sorting

- Thm: Any decision tree that sorts n elements has height $\Omega(n lg n)$.
  - What's the minimum # of leaves? Answer: $n!$. That is, if ℓ is the number of leaves then $\ell \geq n!$.
  - What's the maximum # of leaves of a binary tree of height h?

Lower Bound For Comparison Sorting

- Thm: Any decision tree that sorts n elements has height $\Omega(nlgn)$.
  - What's the minimum # of leaves? Answer: $n!$. That is, if ℓ is the number of leaves then $\ell \geq n!$.
  - What's the maximum # of leaves of a binary tree of height h?

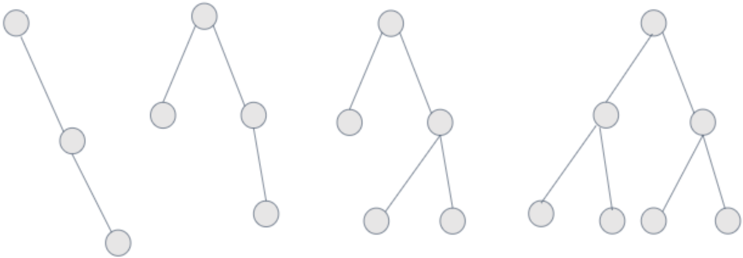
For $h = 1, \ell \leq 2^1$,



Lower Bound For Comparison Sorting

- Thm: Any decision tree that sorts n elements has height $\Omega(n lg n)$.
  - What's the minimum # of leaves?
    Answer: $n!$. That is, if ℓ is the number of leaves then $\ell \geq n!$.
  - What's the maximum # of leaves of a binary tree of height h?

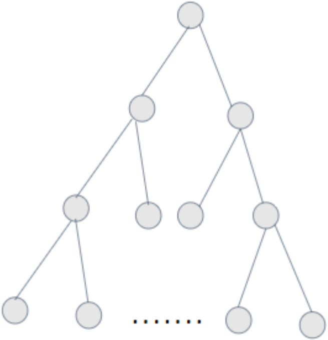
For $h = 2, \ell \leq 2^2$,



Lower Bound For Comparison Sorting

- Thm: Any decision tree that sorts n elements has height $\Omega(n lgn)$.
  - What's the minimum # of leaves?
    Answer: $n!$. That is, if ℓ is the number of leaves then $\ell \geq n!$.
  - What's the maximum # of leaves of a binary tree of height h?

For general $h, \ell \leq 2^h$ (can be proved by induction on h),



Lower Bound For Comparison Sorting

- Thus, $n! \leq \ell \leq 2^h$

Lower Bound For Comparison Sorting

- Thus, $n! \leq \ell \leq 2^h$
- Taking logarithms: $lg(n!) \leq h$

Lower Bound For Comparison Sorting

- Thus, $n! \leq \ell \leq 2^h$
- Taking logarithms: $lg(n!) \leq h$
- By Stirling's approximation: $n! > (\frac{n}{e})^n$
  $\implies \log n! > n \log n - n \log e = \theta(n \log n)$

Lower Bound For Comparison Sorting

- Thus, $n! \leq \ell \leq 2^h$
- Taking logarithms: $lg(n!) \leq h$
- By Stirling's approximation: $n! > (\frac{n}{e})^n$
  $\implies \log n! > n \log n - n \log e = \theta(n \log n)$
- Thus, $\log n! = \Omega(nlgn)$ and hence $h = \Omega(nlgn)$.

Lower Bound For Comparison Sorting

- Thus, $n! \leq \ell \leq 2^h$
- Taking logarithms: $lg(n!) \leq h$
- By Stirling's approximation: $n! > (\frac{n}{e})^n$
  $\implies \log n! > n \log n - n \log e = \theta(n \log n)$
- Thus, $\log n! = \Omega(nlgn)$ and hence $h = \Omega(nlgn)$.
- Thus the minimum height of a decision tree is $\Omega(nlgn)$

Lower Bound For Comparison Sorting

- Thus, $n! \leq \ell \leq 2^h$
- Taking logarithms: $lg(n!) \leq h$
- By Stirling's approximation: $n! > (\frac{n}{e})^n$
  $\implies \log n! > n \log n - n \log e = \theta(n \log n)$
- Thus, $\log n! = \Omega(nlgn)$ and hence $h = \Omega(nlgn)$.
- Thus the minimum height of a decision tree is $\Omega(nlgn)$
- Thus the time to sort n elements based only on comparisons without using any extra information, like values, in the worst case,is $\Omega(nlgn)$.

Lower Bound For Comparison Sorting

- Thus, $n! \leq \ell \leq 2^h$
- Taking logarithms: $lg(n!) \leq h$
- By Stirling's approximation: $n! > (\frac{n}{e})^n$
  $\implies \log n! > n \log n - n \log e = \theta(n \log n)$
- Thus, $\log n! = \Omega(nlgn)$ and hence $h = \Omega(nlgn)$.
- Thus the minimum height of a decision tree is $\Omega(nlgn)$
- Thus the time to sort n elements based only on comparisons without using any extra information, like values, in the worst case,is $\Omega(nlgn)$.
- Corollary: Mergesort is asymptotically optimal in the category of comparison sort algorithms.

Can we do better by using some extra information?

Suppose we are given n integers in the range $1 \dots m$. Can we sort faster?

Can we do better by using some extra information?

Suppose we are given n integers in the range $1 \dots m$. Can we sort faster?

Answer is Yes. We can sort them in $O(n)$ time.

Let us see with the help of an example.

Example 1

4	5	2	0	4	2	0	2

Example 1

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Ouput

							5

Output

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Ouput

						4	**5**

Output

Index	**0**	**1**	**2**	**3**	**4**	**5**
Count	2	0	3	0	2	1

Frequency Table

Ouput

					4	**4**	**5**

Output

Index	**0**	**1**	**2**	**3**	**4**	**5**
Count	2	0	3	0	2	1

Frequency Table

Ouput

				2	**4**	**4**	**5**

Output

Index	**0**	**1**	**2**	**3**	**4**	**5**
Count	2	0	3	0	2	1

Frequency Table

Ouput

			2	2	4	4	5

Output

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Ouput

		2	**2**	**2**	**4**	**4**	**5**

Output

Index	**0**	**1**	**2**	**3**	**4**	**5**
Count	2	0	3	0	2	1

Frequency Table

Ouput

	0	**2**	**2**	**2**	**4**	**4**	**5**

Output

Index	**0**	**1**	**2**	**3**	**4**	**5**
Count	2	0	3	0	2	1

Frequency Table

Ouput

0	**0**	**2**	**2**	**2**	**4**	**4**	**5**

Output

Index	**0**	**1**	**2**	**3**	**4**	**5**
Count	2	0	3	0	2	1

Frequency Table

Ouput

0	**0**	**2**	**2**	**2**	**4**	**4**	**5**

Output

Index	**0**	**1**	**2**	**3**	**4**	**5**
Count	2	0	3	0	2	1

Frequency Table

The algorithm is called Count Sort since we sort by counting.

Ouput

0	0	2	2	2	4	4	5

Output

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

The algorithm is called Count Sort since we sort by counting.
We will no more write pseudo-codes and we will talk about the algorithms at abstract level.

Example 2 : Count Sort with Satellite Data

Sort the following pincodes on their last digit.

110014	110005	110002	110020	110004	110022	110010	110012

Example 2

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Example 2 : Count Sort with Satellite Data

Sort the following pincodes on their last digit.

110014	110005	110002	110020	110004	110022	110010	110012

Example 2

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Index	0	1	2	3	4	5
Count	2	2	5	5	7	8

Cumulative Frequency Table

Example 2 : Count Sort with Satellite Data

Sort the following pincodes on their last digit.

110014	110005	110002	110020	110004	110022	110010	110012

Example 2

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Index	0	1	2	3	4	5
Count	2	2	5	5	7	8

Cumulative Frequency Table

$CF[i]$ gives us the number of elements with last digit $\leq i$. Thus, this gives us the last location where such elements should be stored. Let us first understand this concept on Example 1.

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Example 1

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Example 1

Index	0	1	2	3	4	5
Count	2	0	3	0	2	1

Frequency Table

Index	0	1	2	3	4	5
Count	2	2	5	5	7	8

Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	2	2	5	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data					2			

Output

Index	0	1	2	3	4	5
Count	2	2	4	5	7	8

Updated Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	2	2	4	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data		0			2			

Output

Index	0	1	2	3	4	5
Count	1	2	4	5	7	8

Updated Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	1	2	4	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data		0		2	2			

Output

Index	0	1	2	3	4	5
Count	1	2	3	5	7	8

Updated Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	1	2	3	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data		0		2	2		4	

Output

Index	0	1	2	3	4	5
Count	1	2	3	5	6	8

Updated Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	1	2	3	5	6	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	0	0		2	2		4	

Output

Index	0	1	2	3	4	5
Count	0	2	3	5	6	8

Updated Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	0	2	3	5	6	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	0	0	2	2	2		4	

Output

Index	0	1	2	3	4	5
Count	0	2	2	5	6	8

Updated Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	0	2	2	5	6	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	0	0	2	2	2		4	5

Output

Index	0	1	2	3	4	5
Count	0	2	2	5	6	7

Updated Cumulative Frequency Table

Example 1 with cumulative Frequency

4	5	2	0	4	2	0	2

Index	0	1	2	3	4	5
Count	0	2	2	5	6	7

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	0	0	2	2	2	4	4	5

Output

Index	0	1	2	3	4	5
Count	0	2	2	5	5	7

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	2	2	5	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data					110012			

Output

Index	0	1	2	3	4	5
Count	2	2	4	5	7	8

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	2	2	4	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data		110010			110012			

Output

Index	0	1	2	3	4	5
Count	1	2	4	5	7	8

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	1	2	4	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data		110010		110022	110012			

Output

Index	0	1	2	3	4	5
Count	1	2	3	5	7	8

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	1	2	3	5	7	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data		110010		110022	110012		110004	

Output

Index	0	1	2	3	4	5
Count	1	2	3	5	6	8

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	1	2	3	5	6	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	110020	110010		110022	110012		110004	

Output

Index	0	1	2	3	4	5
Count	0	2	3	5	6	8

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	0	2	3	5	6	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	110020	110010	110002	110022	110012		110004	

Output

Index	0	1	2	3	4	5
Count	0	2	2	5	6	8

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	0	2	2	5	6	8

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	110020	110010	110002	110022	110012		110004	110005

Output

Index	0	1	2	3	4	5
Count	0	2	2	5	6	7

Updated Cumulative Frequency Table

Example 2 contd...: Writing the Ouput

110014	110005	110002	110020	110004	110022	110010	110012

Index	0	1	2	3	4	5
Count	0	2	2	5	6	7

Cumulative Frequency Table

Index	1	2	3	4	5	6	7	8
Data	110020	110010	110002	110022	110012	110014	110004	110005

Output

Index	0	1	2	3	4	5
Count	0	2	2	5	5	7

Updated Cumulative Frequency Table

Count Sort Algorithm

input : Array: $A[1 \dots n]$, Range R of keys say $0 \dots m-1$
output: Sorted Array A

Count-Sort(A, R)

1. Compute the frequencies.
2. Compute the cumulative frequencies.
3. Scan the input in reverse order. For every scanned element x, get it's location i from the CFT and write x in the i^{th} location of the output array and decrement the value corresponding to x in CFT by 1.
4. Copy the output array to the input array.

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:

Count Sort Analysis

- Let $[1 \dots m]$ or $[0 \dots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?
  - Time to compute the cumulative frequencies?

Count Sort Analysis

- Let $[1 \dots m]$ or $[0 \dots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?
  - Time to compute the cumulative frequencies?
  - Time to write the output?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?
  - Time to compute the cumulative frequencies?
  - Time to write the output?
- How much extra-space does it take? Is it in-place?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?
  - Time to compute the cumulative frequencies?
  - Time to write the output?
- How much extra-space does it take? Is it in-place?
- Is it stable?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?
  - Time to compute the cumulative frequencies?
  - Time to write the output?
- How much extra-space does it take? Is it in-place?
- Is it stable?
- Given a set of elements drawn from the Capital English alphabet set $\{A, B, \ldots Z\}$. Can Count sort be used to sort them?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?
  - Time to compute the cumulative frequencies?
  - Time to write the output?
- How much extra-space does it take? Is it in-place?
- Is it stable?
- Given a set of elements drawn from the Capital English alphabet set $\{A, B, \ldots Z\}$. Can Count sort be used to sort them? What is m?

Count Sort Analysis

- Let $[1 \ldots m]$ or $[0 \ldots m-1]$ (doesn't really matter) be the range in which the keys lie.
  - What is the key in Eg. 2?
  - What is it's range?
- Time Complexity of Count Sort:
  - Time to compute the frequencies?
  - Time to compute the cumulative frequencies?
  - Time to write the output?
- How much extra-space does it take? Is it in-place?
- Is it stable?
- Given a set of elements drawn from the Capital English alphabet set $\{A, B, \ldots Z\}$. Can Count sort be used to sort them? What is m? What is the time complexity?