CSC 225 - Summer 2019

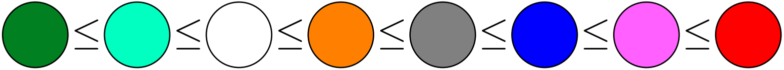
Sorting III

Bill Bird

Department of Computer Science
University of Victoria

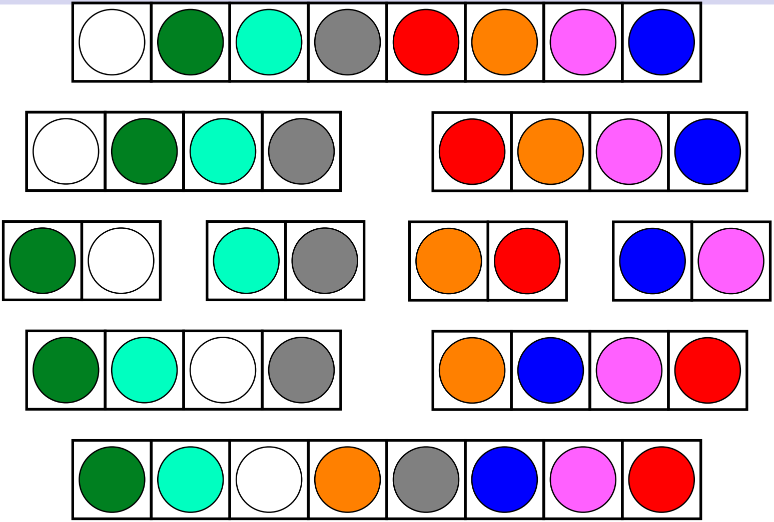
June 5, 2019

Comparison Sorting (1)



- General sorting algorithms cannot make any assumptions about the input data.
- Comparison sorting algorithms, such as Merge Sort, Quicksort and Heap Sort, only interact with the input data by asking questions of the form '$a \leq b$'.
- We assume that for all input items x and y, either $x \leq y$ or $y \leq x$ (or both). This property is called 'total ordering'.
- The sorting algorithm is not concerned with how the $\leq$ operator works.

Comparison Sorting (2)



- Comparison sorting algorithms must make $\Omega(n \log n)$ queries of the form $a \leq b$ in the worst case.
- The proof of this lower bound is now part of CSC 226.

Comparison of Comparison Sorting Algorithms (1)

n	Max. Comparisons			
	Selection Sort	Merge Sort	Quicksort	Heap Sort
3	3	3	3	3
4	6	5	6	7
5	10	8	10	12
6	15	11	15	17
7	21	14	21	22
8	28	17	28	29
9	36	21	36	37
10	45	25	45	44
11	55	29	55	51
12	66	33	66	59
13	78	37	78	66
14	91	41	91	73
15	105	45	105	80

This table gives the maximum number of comparisons performed by each algorithm over all arrays of size $n = 3, \ldots, 15$.

Comparison of Comparison Sorting Algorithms (2)

n	Average Comparisons			
	Selection Sort	Merge Sort	Quicksort	Heap Sort
3	3.0	2.7	2.7	3.0
4	6.0	4.7	4.8	6.5
5	10.0	7.2	7.4	11.0
6	15.0	9.8	10.3	15.1
7	21.0	12.7	13.5	19.8
8	28.0	15.7	16.9	25.8
9	36.0	19.2	20.6	32.5
10	45.0	22.7	24.4	38.6
11	55.0	26.3	28.5	45.2
12	66.0	30.0	32.7	51.8
13	78.0	33.8	37.0	59.0
14	91.0	37.7	41.6	65.5
15	105.0	41.7	46.2	72.5

This table gives the average number of comparisons performed by each algorithm over all arrays of size $n = 3, \ldots, 15$.

Sorting Under Assumptions (1)

314		323		314		159
159		843		323		265
265		383		843		314
358		314		358		323
979	$\rightarrow$	265	$\rightarrow$	159	$\rightarrow$	358
323		358		265		383
843		159		979		843
383		979		383		979

When assumptions can be made about the input data, the $\Omega(n \log n)$ bound can sometimes be broken.

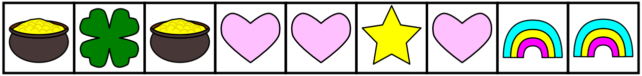
Sorting Under Assumptions (2)

314		323		314		159
159		843		323		265
265		383		843		314
358	$\rightarrow$	314	$\rightarrow$	358	$\rightarrow$	323
979		265		159		358
323		358		265		383
843		159		979		843
383		979		383		979

For example, when all input values are 3-digit integers, the Radix Sort algorithm requires $O(n)$ time.

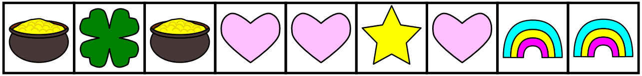
Bucket Sort (1)

A:



Bucket Sort is a simple algorithm for sorting sequences with a small number of possible values. In the above example, there are only five possible elements.

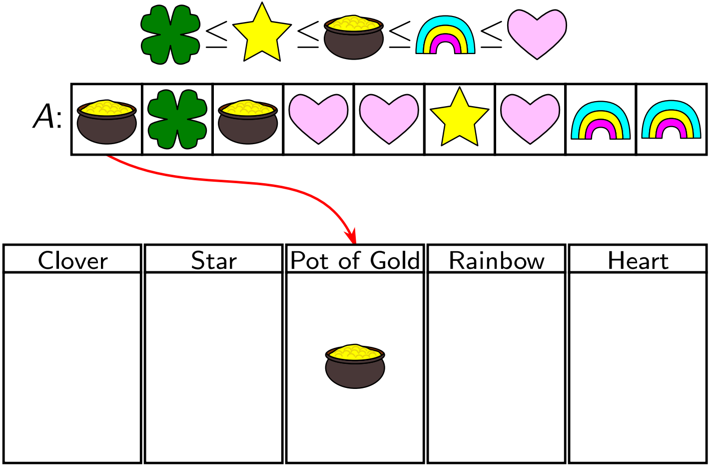
Bucket Sort (2)

A: 

Clover	Star	Pot of Gold	Rainbow	Heart

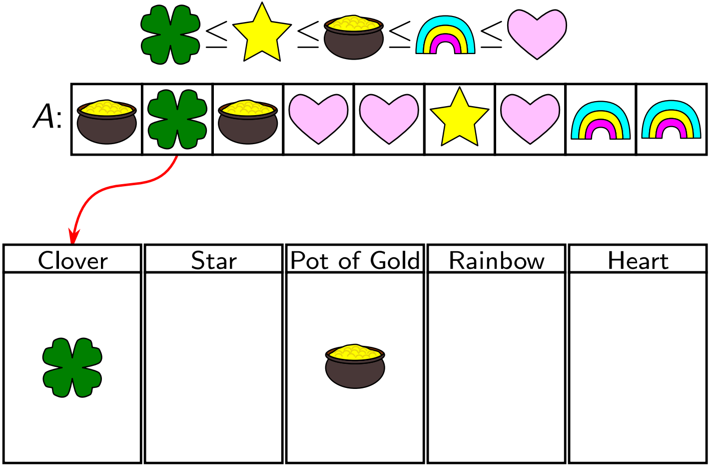
A **bucket** is created for each type of element.

Bucket Sort (3)



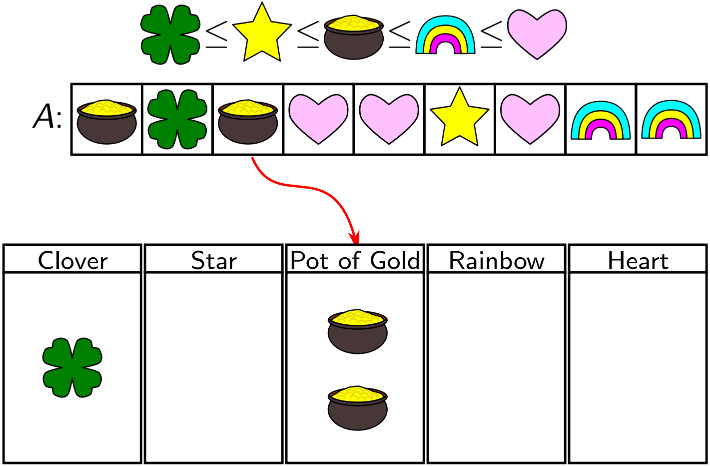
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (4)



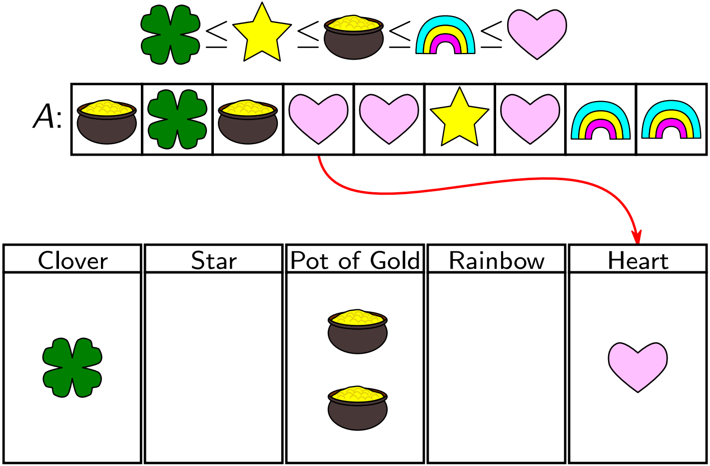
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (5)



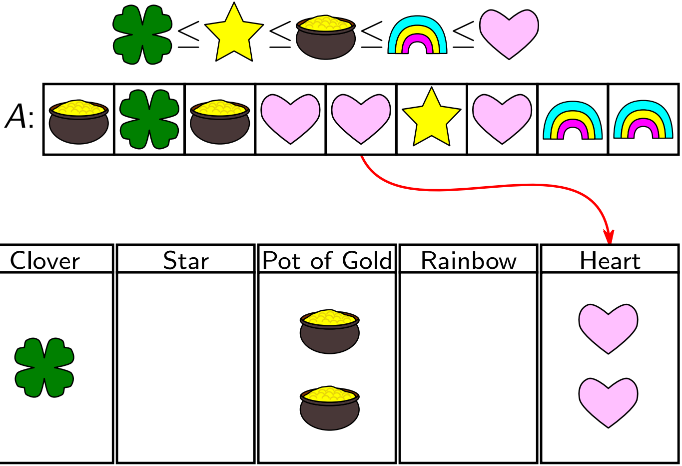
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (6)



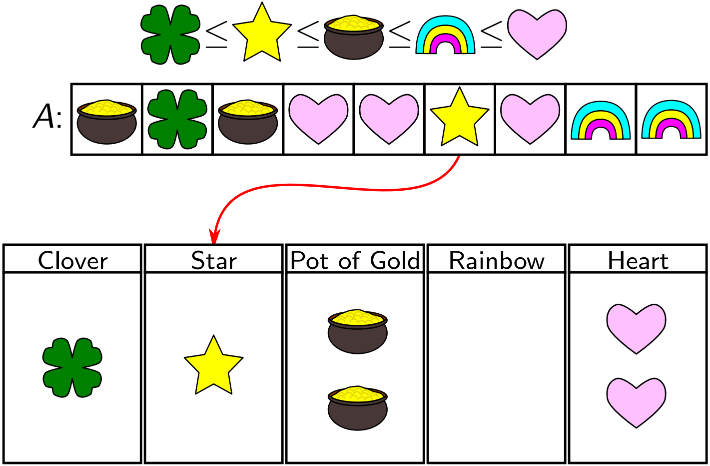
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (7)



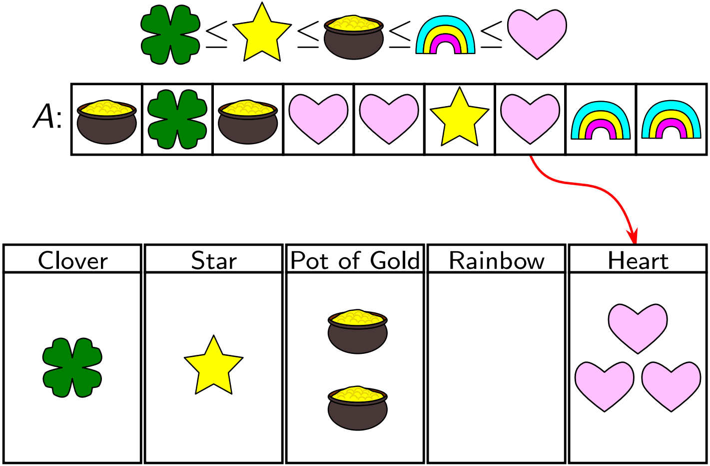
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (8)



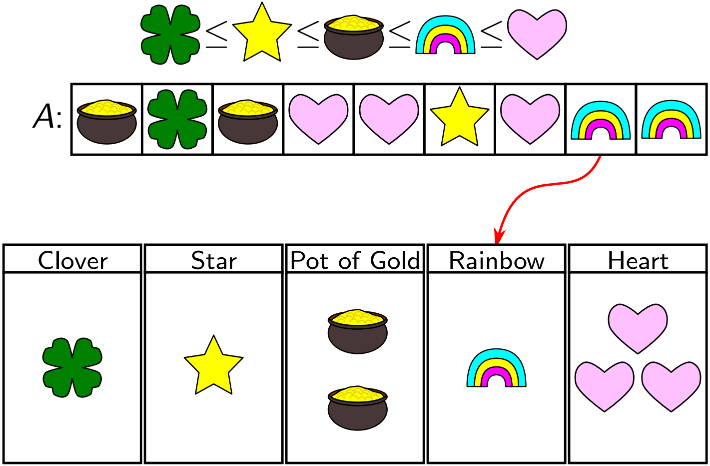
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (9)



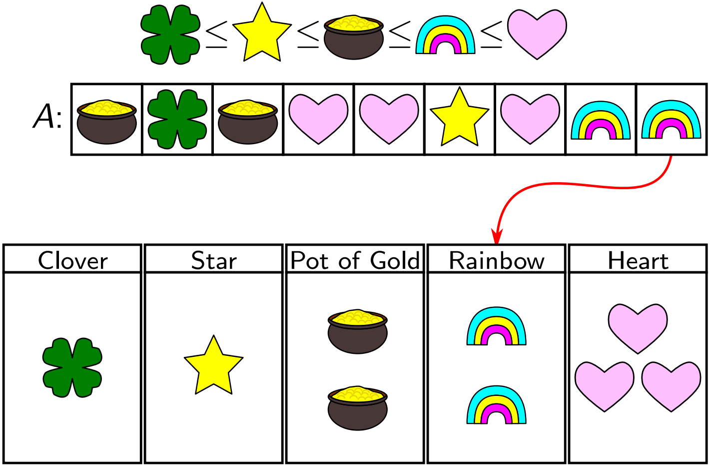
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (10)



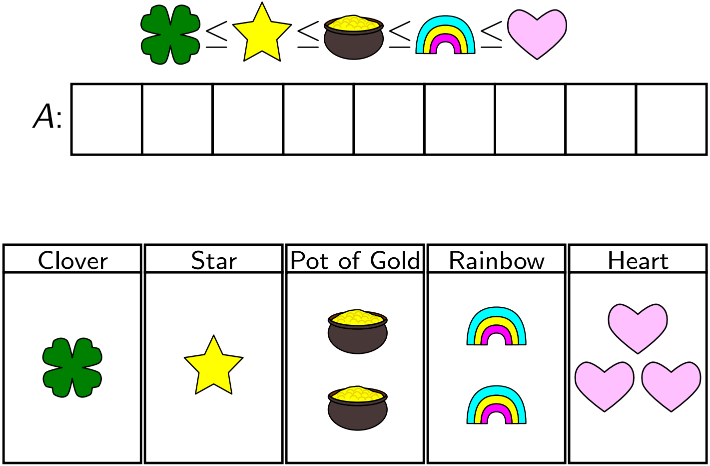
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (11)



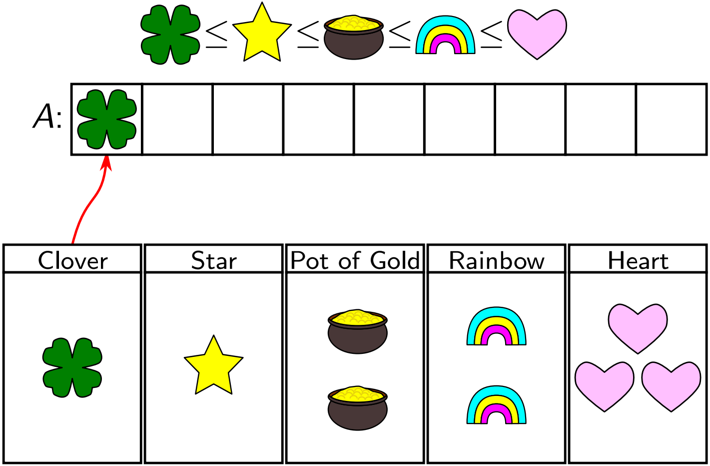
The first part of bucket sort loops over the input array, adding each element to its respective bucket.

Bucket Sort (12)



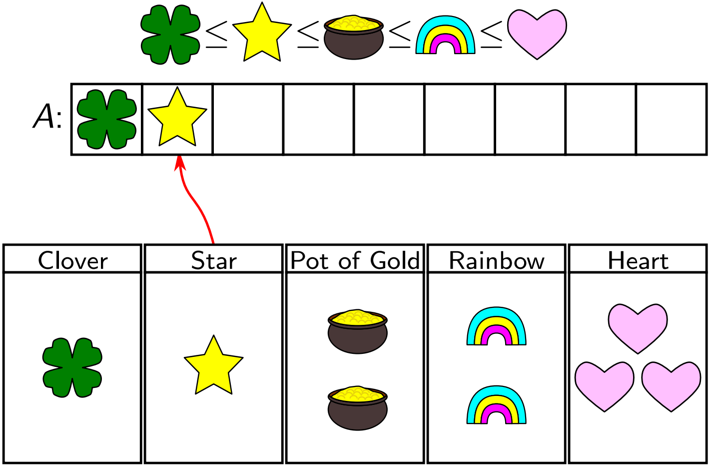
After all elements have been added to buckets, the array is cleared and used to store the sorted output.

Bucket Sort (13)



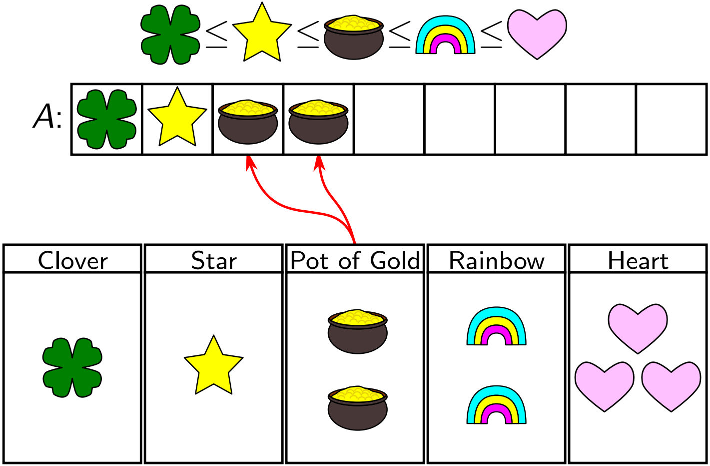
To generate the sorted array, the buckets are visited in ascending order and each bucket's contents are appended to the array.

Bucket Sort (14)

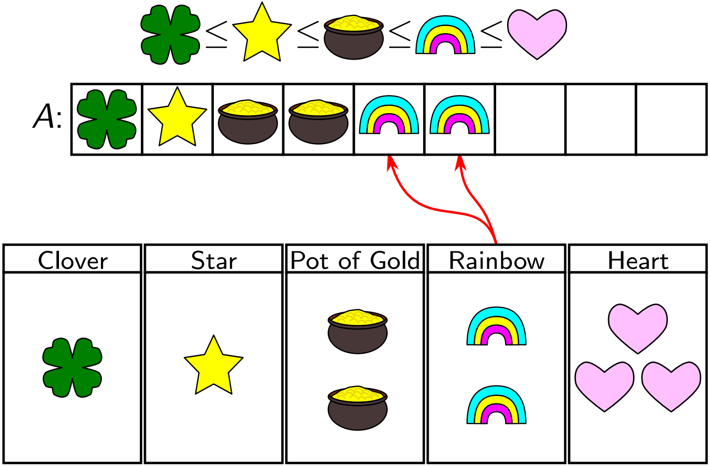


To generate the sorted array, the buckets are visited in ascending order and each bucket's contents are appended to the array.

Bucket Sort (15)

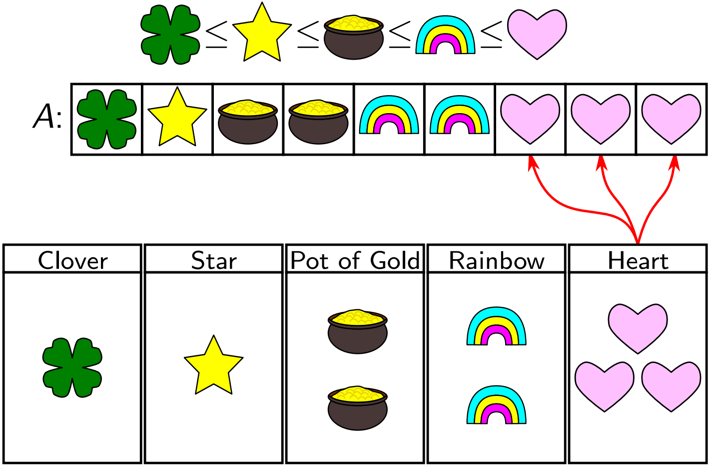


To generate the sorted array, the buckets are visited in ascending order and each bucket's contents are appended to the array.



To generate the sorted array, the buckets are visited in ascending order and each bucket's contents are appended to the array.

Bucket Sort (17)



To generate the sorted array, the buckets are visited in ascending order and each bucket's contents are appended to the array.

Bucket Sort (18)

Name	Dept.
Bill	CSc
Gus	Business
Hank	Psychology
Jesse	Chemistry
Lydia	Business
Marie	Physics
Mike	Physics
Skyler	Business
Venkatesh	CSc
Walter	Chemistry

Business	Chemistry	CSc	Physics	Psych.
Gus	Jesse	Bill	Marie	Hank
Lydia	Walter	Venkatesh	Mike	
Skyler				

Name	Dept.
Gus	Business
Lydia	Business
Skyler	Business
Jesse	Chemistry
Walter	Chemistry
Bill	CSc
Venkatesh	CSc
Marie	Physics
Mike	Physics
Hank	Psychology

Bucket sort is very efficient when sorting on a key with a limited number of values (such as sorting employees by department in the example above).

Bucket Sort (19)

Name	Dept.
Bill	CSc
Gus	Business
Hank	Psychology
Jesse	Chemistry
Lydia	Business
Marie	Physics
Mike	Physics
Skyler	Business
Venkatesh	CSc
Walter	Chemistry

Business	Chemistry	CSc	Physics	Psych.
Gus	Jesse	Bill	Marie	Hank
Lydia	Walter	Venkatesh	Mike	
Skyler				

Name	Dept.
Gus	Business
Lydia	Business
Skyler	Business
Jesse	Chemistry
Walter	Chemistry
Bill	CSc
Venkatesh	CSc
Marie	Physics
Mike	Physics
Hank	Psychology

If there are b buckets, the running time of bucket sort on an array of size n is $O(n + b)$.

Bucket Sort (20)

Name	Dept.
Bill	CSc
Gus	Business
Hank	Psychology
Jesse	Chemistry
Lydia	Business
Marie	Physics
Mike	Physics
Skyler	Business
Venkatesh	CSc
Walter	Chemistry

Business	Chemistry	CSc	Physics	Psych.
Gus	Jesse	Bill	Marie	Hank
Lydia	Walter	Venkatesh	Mike	
Skyler				

Name	Dept.
Gus	Business
Lydia	Business
Skyler	Business
Jesse	Chemistry
Walter	Chemistry
Bill	CSc
Venkatesh	CSc
Marie	Physics
Mike	Physics
Hank	Psychology

Bucket sort does not use pairwise comparisons at all, which is why the running time breaks the $\Omega(n \log n)$ bound.

Alphabetizing With Bucket Sort (1)

Country
Canada
Belgium
Sweden
China
Cambodia
Australia
Cameroon
Argentina
Belize
Slovenia

Country
Australia
Argentina
Belgium
Belize
Canada
China
Cambodia
Cameroon
Sweden
Slovenia

A	B	C	D		S		Z
Australia	Belgium	Canada		···	Sweden	···	
Argentina	Belize	China			Slovenia		
		Cambodia					
		Cameroon					

Using 26 buckets, a set of strings can be sorted by first letter using bucket sort.

Alphabetizing With Bucket Sort (2)

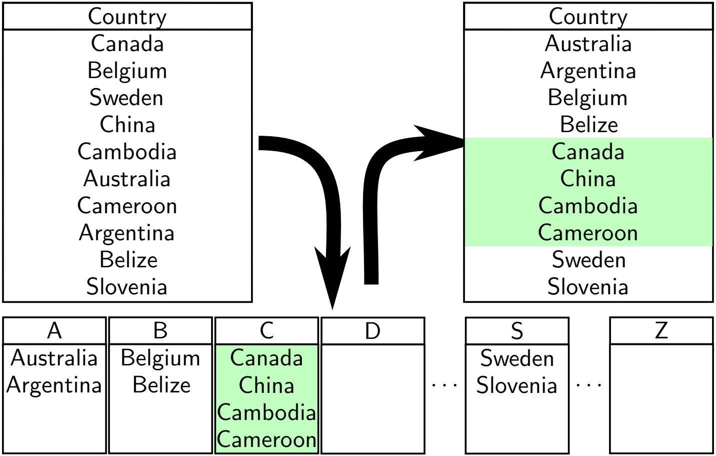
Country
Canada
Belgium
Sweden
China
Cambodia
Australia
Cameroon
Argentina
Belize
Slovenia

Country
Australia
Argentina
Belgium
Belize
Canada
China
Cambodia
Cameroon
Sweden
Slovenia

A	B	C	D	…	S	…	Z
Australia	Belgium	Canada			Sweden		
Argentina	Belize	China			Slovenia		
		Cambodia					
		Cameroon					

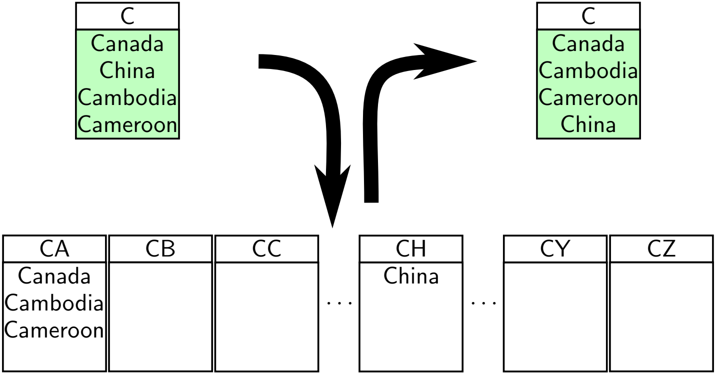
However, bucket sort does not alphabetize the strings.

Alphabetizing With Bucket Sort (3)



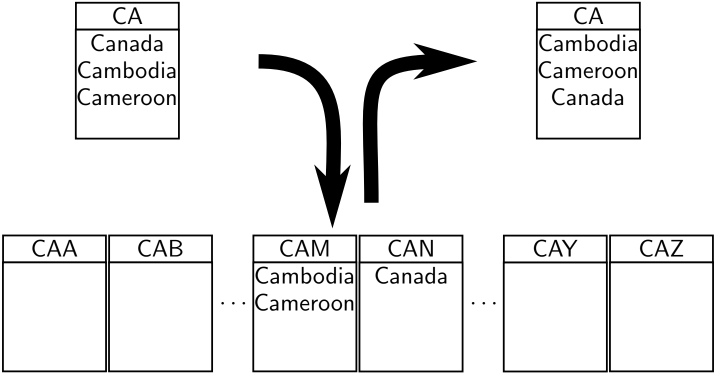
For example, the elements of the ‘C’ bucket are not in alphabetical order, since bucket sort puts them in the bucket in the order they were found.

Alphabetizing With Bucket Sort (4)



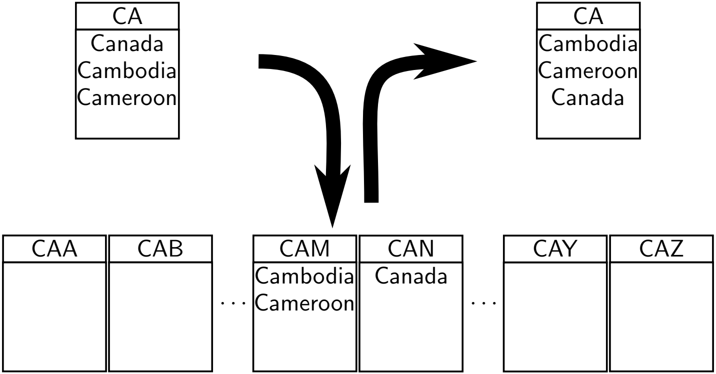
One solution is to recursively use bucket sort on each bucket and sort by the second letter...

Alphabetizing With Bucket Sort (5)



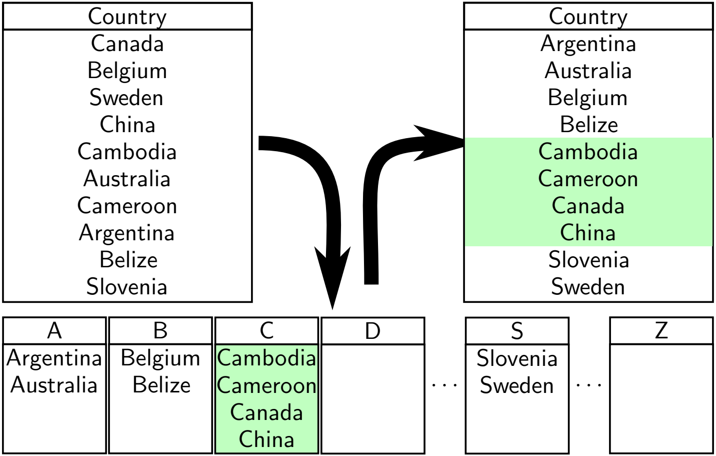
...then by the third letter, and continue until all buckets have size 1.

Alphabetizing With Bucket Sort (6)



If the strings in the list have maximum length k, sorting may require k passes.

Alphabetizing With Bucket Sort (7)



After the contents of each bucket are alphabetized, the resulting list is in alphabetical order.

Alphabetizing With Bucket Sort (8)

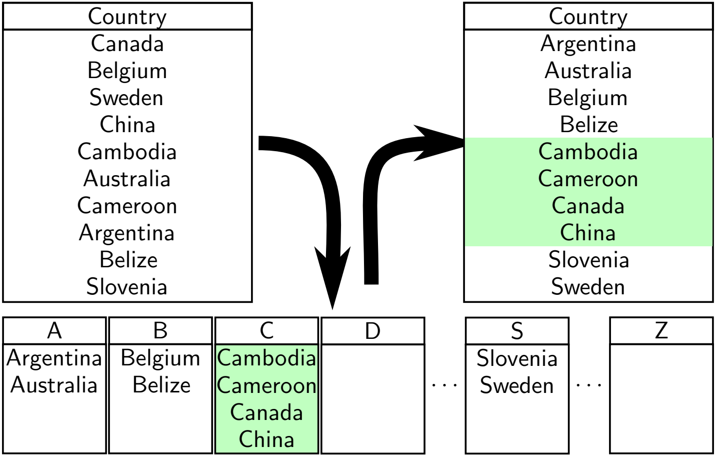
Country
Canada
Belgium
Sweden
China
Cambodia
Australia
Cameroon
Argentina
Belize
Slovenia

Country
Argentina
Australia
Belgium
Belize
Cambodia
Cameroon
Canada
China
Slovenia
Sweden

A	B	C	D		S		Z
Argentina	Belgium	Cambodia		···	Slovenia	···	
Australia	Belize	Cameroon			Sweden		
		Canada					
		China					

In a list of n strings with maximum length k, each string will be subjected to at most k sorting passes. Therefore, the list can be alphabetized in $O(nk)$ time.

Alphabetizing With Bucket Sort (9)



This variant of bucket sort is often called 'lexicographical sort' or 'dictionary sort'.

Alphabetizing With Bucket Sort (10)

Country
Canada
Belgium
Sweden
China
Cambodia
Australia
Cameroon
Argentina
Belize
Slovenia

Country
Argentina
Australia
Belgium
Belize
Cambodia
Cameroon
Canada
China
Slovenia
Sweden

A	B	C	D		S		Z
Argentina	Belgium	Cambodia		···	Slovenia	···	
Australia	Belize	Cameroon			Sweden		
		Canada					
		China					

When the maximum length k is bounded by a constant (e.g. 10 characters), lexicographical sort runs in $O(n)$ time.

Sorting Algorithms

	Running Time			Extra Space	Stable?
	Best Case	Expected Case	Worst Case		
Selection Based					
Heap Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(1)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	No
Divide and Conquer					
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Other					
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Radix Sort[1]	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(n + b)$	Yes

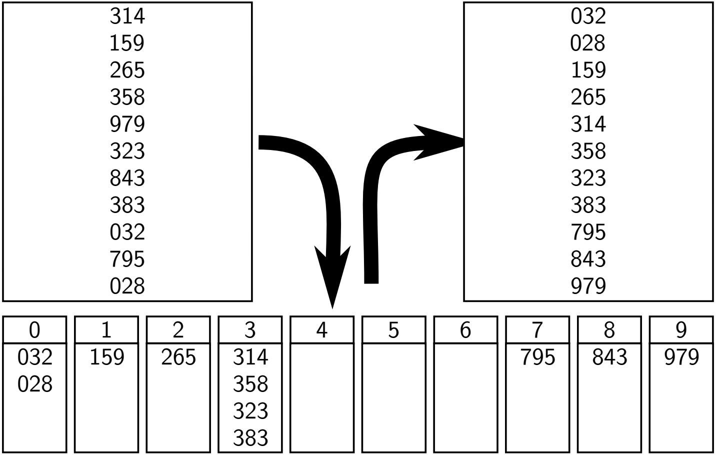
[1] Integers only: d-digit values in base b

Sorting Algorithms

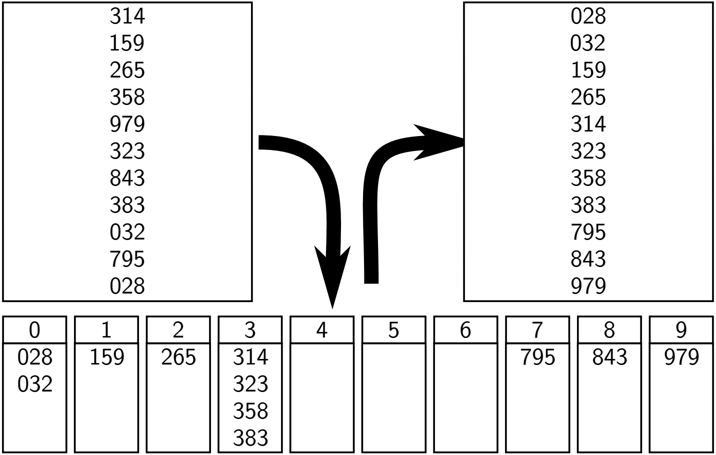
	Running Time			Extra Space	Stable?
	Best Case	Expected Case	Worst Case		
Selection Based					
Heap Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(1)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	No
Divide and Conquer					
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Other					
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Radix Sort[1]	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(n + b)$	Yes

[1] Integers only: d-digit values in base b

Radix Sort (1)

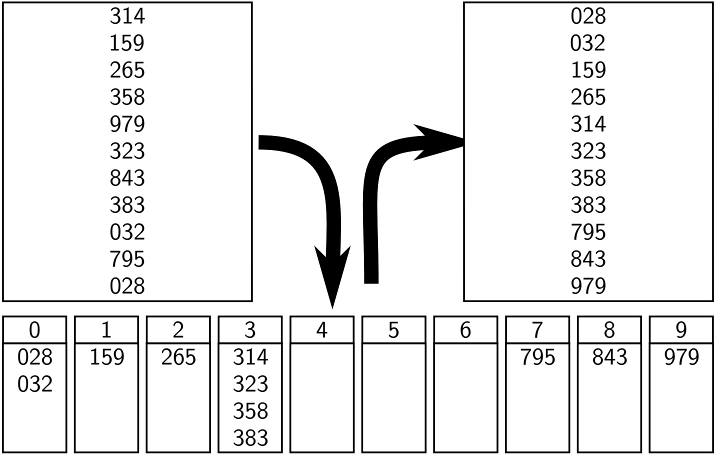


Idea: Treat integers as strings of digits and use lexicographic sort.



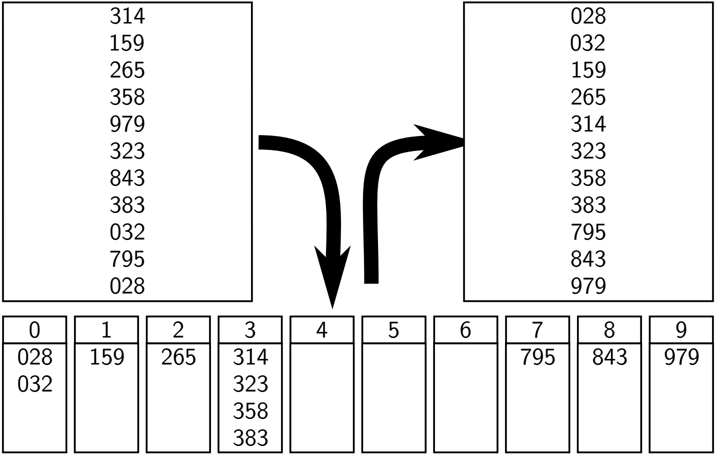
By recursively sorting the buckets as before, this strategy does produce an effective sorting algorithm.

Radix Sort (3)



The resulting algorithm is **radix sort**. It can be performed in any base (base 10 is used here).

Radix Sort (4)



The running time of radix sort is $\Theta(d(n+b))$, where

$d =$ number of digits

$n =$ number of elements in input list

$b =$ base (i.e. number of buckets)

MSD Radix Sort (1)

```
procedure MSDRadixSort(A, n, k)
    if k < 0 then
        return
    end if
    B ← Array of 10 empty lists.
    for i = 0, 1, ..., n − 1 do
        digit ← k^th digit of A[i]
        Add A[i] to the end of list B[digit]
    end for
    S ← Empty list.
    for j = 0, 1, ..., 9 do
        if len(B[j]) > 1 then
            MSDRadixSort(B[j], length(B[j]), k − 1)
        end if
        Append B[j] to the end of S
    end for
    Copy S into A
end procedure
```

This variant is called Most Significant Digit (MSD) Radix Sort, since it starts with the most significant (leftmost) digit.

MSD Radix Sort (2)

```
procedure MSDRadixSort(A, n, k)
    if k < 0 then
        return
    end if
    B ← Array of 10 empty lists.
    for i = 0, 1, ..., n − 1 do
        digit ← k^th digit of A[i]
        Add A[i] to the end of list B[digit]
    end for
    S ← Empty list.
    for j = 0, 1, ..., 9 do
        if len(B[j]) > 1 then
            MSDRadixSort(B[j], length(B[j]), k − 1)
        end if
        Append B[j] to the end of S
    end for
    Copy S into A
end procedure
```

To sort numbers with d digits, the initial call is $\textsc{MSDRadixSort}(A, \text{length}(A), d-1)$

MSD Radix Sort (3)

```
procedure MSDRadixSort(A, n, k)
    if k < 0 then
        return
    end if
    B ← Array of 10 empty lists.
    for i = 0, 1, ..., n − 1 do
        digit ← k^th digit of A[i]
        Add A[i] to the end of list B[digit]
    end for
    S ← Empty list.
    for j = 0, 1, ..., 9 do
        if len(B[j]) > 1 then
            MSDRadixSort(B[j], length(B[j]), k − 1)
        end if
        Append B[j] to the end of S
    end for
    Copy S into A
end procedure
```

Note that digits are numbered starting at the right (so digit 0 is the least significant digit).

LSD Radix Sort (1)

Input Array
314
159
265
358
979
323
843
383
032
795

Pass 1
032
323
843
383
314
265
795
358
159
979

__0	__1	__2	__3	__4	__5	__6	__7	__8	__9
		032	323	314	265			358	159
			843		795				979
			383						

An alternative is Least Significant Digit (LSD) Radix Sort. Instead of recursively calling bucket sort, LSD Radix Sort does a sequence of d bucket sorts over the whole array.

LSD Radix Sort (2)

Input Array
314
159
265
358
979
323
843
383
032
795

Pass 1
032
323
843
383
314
265
795
358
159
979

__0	__1	__2	__3	__4	__5	__6	__7	__8	__9
		032	323	314	265			358	159
			843		795				979
			383						

In an array of d digit numbers, d passes are needed.

LSD Radix Sort (3)

Input Array
314
159
265
358
979
323
843
383
032
795

Pass 1
032
323
843
383
314
265
795
358
159
979

__0	__1	__2	__3	__4	__5	__6	__7	__8	__9
		032	323	314	265			358	159
			843		795				979
			383						

In pass 1, the array is sorted by its least significant digit (digit 0). The output of pass 1 is used as the input of pass 2. After pass 1, the buckets can

LSD Radix Sort (4)

Pass 1
032
323
843
383
314
265
795
358
159
979

Pass 2
314
323
032
843
358
159
265
979
383
795

_0_	_1_	_2_	_3_	_4_	_5_	_6_	_7_	_8_	_9_
	314	323	032	843	358	265	979	383	795
					159				

In pass 2, the output of pass 1 is sorted by digit 1.

LSD Radix Sort (5)

Pass 1
032
323
843
383
314
265
795
358
159
979

Pass 2
314
323
032
843
358
159
265
979
383
795

_0_	_1_	_2_	_3_	_4_	_5_	_6_	_7_	_8_	_9_
	314	323	032	843	358	265	979	383	795
					159				

The bucket sort at each step must always insert values at the end of each bucket's list for LSD Radix Sort to work.

LSD Radix Sort (6)

Pass 2
314
323
032
843
358
159
265
979
383
795

Output Array
032
159
265
314
323
358
383
795
843
979

0__	1__	2__	3__	4__	5__	6__	7__	8__	9__
032	159	265	314				795	843	979
			323						
			358						
			383						

In pass 3, the output of pass 2 is sorted by digit 2.

LSD Radix Sort (7)

Pass 2
314
323
032
843
358
159
265
979
383
795

Output Array
032
159
265
314
323
358
383
795
843
979

0__	1__	2__	3__	4__	5__	6__	7__	8__	9__
032	159	265	314				795	843	979
			323						
			358						
			383						

Since the values in the example above are 3 digits long, pass 3 is the final pass.

LSD Radix Sort (8)

Pass 2
314
323
032
843
358
159
265
979
383
795

Output Array
032
159
265
314
323
358
383
795
843
979

0__	1__	2__	3__	4__	5__	6__	7__	8__	9__
032	159	265	314				795	843	979
			323						
			358						
			383						

Like MSD Radix Sort, the running time of LSD Radix Sort is $\Theta(d(n+b))$.

LSD Radix Sort (9)

```
procedure LSDRadixSort(A, n, d)
    for k = 0, 1, ..., d − 1 do
        B ← Array of 10 empty lists.
        for i = 0, 1, ..., n − 1 do
            digit ← k^th digit of A[i]
            Add A[i] to the end of list B[digit]
        end for
        S ← Empty list.
        for j = 0, 1, ..., 9 do
            Append B[j] to the end of S
        end for
        Copy S into A
    end for
end procedure
```

For sorting lists of integers, LSD Radix Sort tends to have less overhead in practice than MSD Radix Sort.

Radix Sort vs. Comparison Sorting Bound (1)

1111		1110		1100		1000		0000
1110		1100		1000		0000		0001
1101		1010		0100		1001		0010
1100		1000		0000		0001		0011
1011		0110		1101		1010		0100
1010		0100		1001		0010		0101
1001		0010		0101		1011		0110
1000		0000		0001		0011		0111
0111	$\rightarrow$	1111	$\rightarrow$	1110	$\rightarrow$	1100	$\rightarrow$	1000
0110		1101		1010		0100		1001
0101		1011		0110		1101		1010
0100		1001		0010		0101		1011
0011		0111		1111		1110		1100
0010		0101		1011		0110		1101
0001		0011		0111		1111		1110
0000		0001		0011		0111		1111

The number of digits in the base b representation of n is $\lceil \log_b n \rceil$. Therefore, sorting n integers in the range $1, \ldots, n$ with radix sort requires $\Omega(n \log_b n)$ time.

Radix Sort vs. Comparison Sorting Bound (2)

1111		1110		1100		1000		0000
1110		1100		1000		0000		0001
1101		1010		0100		1001		0010
1100		1000		0000		0001		0011
1011		0110		1101		1010		0100
1010		0100		1001		0010		0101
1001		0010		0101		1011		0110
1000	$\rightarrow$	0000	$\rightarrow$	0001	$\rightarrow$	0011	$\rightarrow$	0111
0111		1111		1110		1100		1000
0110		1101		1010		0100		1001
0101		1011		0110		1101		1010
0100		1001		0010		0101		1011
0011		0111		1111		1110		1100
0010		0101		1011		0110		1101
0001		0011		0111		1111		1110
0000		0001		0011		0111		1111

Good Assignment Question: Give an algorithm, based on Radix Sort, which sorts an array of n integers in the range $[1, n^2]$ in $O(n)$ time.

Sorting Algorithm Summary (1)

None of the standard sorting algorithms covered in this course is 'always' the best choice. Depending on the application, and what assumptions can be made about the input data, any of the covered algorithms[1] could be the best choice.

The sorting algorithm currently used for general purpose comparison sorting by Python and Java is **Timsort**, which uses a combination of Merge Sort and Insertion Sort.

[1] Except possibly Bubble Sort.

Sorting Algorithm Summary (2)

Selection Sort (worst case $\Theta(n^2)$):

- Tends to be surprisingly fast on small arrays.
- Predictable branching structure (which can speed up pipelined processors).
- Very easy to implement and likely to require few instructions.

Insertion Sort (worst case $\Theta(n^2)$):

- On nearly-sorted arrays, insertion sort is worst-case $\Theta(n)$
- Insertion sort is often the fastest algorithm on very small arrays.

Sorting Algorithm Summary (3)

Quicksort (worst case $\Theta(n^2)$):

- Expected case is $\Theta(n \log n)$.
- Tends to be faster in practice than Merge Sort.
- When pivots are chosen randomly, the likelihood of worst-case behavior is negligible.

Merge Sort (worst case $\Theta(n \log n)$):

- Guaranteed $\Theta(n \log n)$ running time on all inputs.
- Requires $\Theta(n)$ extra space.
- Can be used when the input sequence is too big to fit into memory (and is instead stored on disk), since all sequence processing is sequential.

Sorting Algorithm Summary (4)

Heap Sort (worst case $\Theta(n \log n)$):

- Guaranteed $\Theta(n \log n)$ running time on all inputs.
- Using HEAPIFY, Heap Sort can be implemented as an in-place algorithm.
- Very high overhead compared to Merge Sort and Quicksort.
- Heap Sort is not stable.

Radix Sort (worst case $\Theta(dn + b)$):

- $\Theta(n)$ worst-case performance on inputs with a constant number of digits.
- Relatively low overhead.
- No need for comparable data (Radix Sort performs zero comparisons).