WIA2005 Algorithm Design & Analysis
Tutorial 3 (by Team 5)

PART 1 – Illustrate the sorting algorithm

Given the following array:

A = <154, 56, 77, 134, 186, 56, 94, 24, 13, 83, 95, 143>

Illustrate how array A is sorted in an ascending order using the following algorithm:

1. Bubble sort

Step 1 : Go through the first iteration by swapping the first element in the array if it is bigger than the element next to it.

Step 2 : Continue swapping the adjacent elements throughout the array until the last element.

154	56	77	134	186	56	94	24	13	83	95	143

56	154	77	134	186	56	94	24	13	83	95	143

56	77	154	134	186	56	94	24	13	83	95	143

56	77	134	154	186	56	94	24	13	83	95	143

56	77	134	154	186	56	94	24	13	83	95	143

56	77	134	154	56	186	94	24	13	83	95	143

56	77	134	154	56	94	186	24	13	83	95	143

56	77	134	154	56	94	24	186	13	83	95	143

56	77	134	154	56	94	24	13	186	83	95	143

56	77	134	154	56	94	24	13	83	186	95	143

56	77	134	154	56	94	24	13	83	95	186	143

Result after first iteration :

56	77	134	154	56	94	24	13	83	95	143	186

Step 3 : Proceed to second iteration.

56	77	134	154	56	94	24	13	83	95	143	186

56	77	134	154	56	94	24	13	83	95	143	186

56	77	134	154	56	94	24	13	83	95	143	186

56	77	134	154	56	94	24	13	83	95	143	186

56	77	134	56	154	94	24	13	83	95	143	186

56	77	134	56	94	154	24	13	83	95	143	186

56	77	134	56	94	24	154	13	83	95	143	186

56	77	134	56	94	24	13	154	83	95	143	186

56	77	134	56	94	24	13	83	154	95	143	186

56	77	134	56	94	24	13	83	95	154	143	186

Result after second iteration :

56	77	134	56	94	24	13	83	95	143	154	186

Step 4 : Proceed to third iteration.

56	77	134	56	94	24	13	83	95	143	154	186

56	77	134	56	94	24	13	83	95	143	154	186

56	77	134	56	94	24	13	83	95	143	154	186

56	77	56	134	94	24	13	83	95	143	154	186

56	77	56	94	134	24	13	83	95	143	154	186

56	77	56	94	24	134	13	83	95	143	154	186

56	77	56	94	24	13	134	83	95	143	154	186

56	77	56	94	24	13	83	134	95	143	154	186

56	77	56	94	24	13	83	95	134	143	154	186

Result after third iteration :

56	77	56	94	24	13	83	95	134	143	154	186

Step 5 : Proceed to fourth iteration.

56	77	56	94	24	13	83	95	134	143	154	186

56	77	56	94	24	13	83	95	134	143	154	186

56	56	77	94	24	13	83	95	134	143	154	186

56	56	77	94	24	13	83	95	134	143	154	186

56	56	77	24	94	13	83	95	134	143	154	186

56	56	77	24	13	94	83	95	134	143	154	186

56	56	77	24	13	83	94	95	134	143	154	186

56	56	77	24	13	83	94	95	134	143	154	186

Result after fourth iteration :

56	56	77	24	13	83	94	95	134	143	154	186

Step 6 : Proceed to fifth iteration.

56	56	77	24	13	83	94	95	134	143	154	186

56	56	77	24	13	83	94	95	134	143	154	186

56	56	77	24	13	83	94	95	134	143	154	186

56	56	24	77	13	83	94	95	134	143	154	186

56	56	24	13	77	83	94	95	134	143	154	186

56	56	24	13	77	83	94	95	134	143	154	186

56	56	24	13	77	83	94	95	134	143	154	186

Result after fifth iteration :

56	56	24	13	77	83	94	95	134	143	154	186

Step 7 : Proceed to sixth iteration.

56	56	24	13	77	83	94	95	134	143	154	186

56	56	24	13	77	83	94	95	134	143	154	186

56	24	56	13	77	83	94	95	134	143	154	186

56	24	13	56	77	83	94	95	134	143	154	186

56	24	13	56	77	83	94	95	134	143	154	186

56	24	13	56	77	83	94	95	134	143	154	186

Result after sixth iteration :

56	24	13	56	77	83	94	95	134	143	154	186

Step 8 : Proceed to seventh iteration.

56	24	13	56	77	83	94	95	134	143	154	186

24	56	13	56	77	83	94	95	134	143	154	186

24	13	56	56	77	83	94	95	134	143	154	186

24	13	56	56	77	83	94	95	134	143	154	186

24	13	56	56	77	83	94	95	134	143	154	186

Result after seventh iteration :

24	13	56	56	77	83	94	95	134	143	154	186

Step 9 : Proceed to eighth iteration.

24	13	56	56	77	83	94	95	134	143	154	186

13	24	56	56	77	83	94	95	134	143	154	186

13	24	56	56	77	83	94	95	134	143	154	186

13	24	56	56	77	83	94	95	134	143	154	186

Result after eighth iteration :

13	24	56	56	77	83	94	95	134	143	154	186

Step 10 : Proceed to ninth iteration.

13	24	56	56	77	83	94	95	134	143	154	186

13	24	56	56	77	83	94	95	134	143	154	186

13	24	56	56	77	83	94	95	134	143	154	186

Result after ninth iteration :

13	24	56	56	77	83	94	95	134	143	154	186

Step 11 : Proceed to tenth iteration.

13	24	56	56	77	83	94	95	134	143	154	186

13	24	56	56	77	83	94	95	134	143	154	186

Result after tenth iteration :

13	24	56	56	77	83	94	95	134	143	154	186

Step 12 : Proceed to eleventh iteration.

13	24	56	56	77	83	94	95	134	143	154	186

Result after eleventh iteration : The array is sorted now.

13	24	56	56	77	83	94	95	134	143	154	186

2. Counting sort

Array A:

154	56	77	134	186	56	94	24	13	83	95	143

Step 1 : Create Count Array (sized 187, the largest value + 1) with initial value of 0 for all elements.

Index	13 …	24 …	56 …	77 …	83 …	94	95 …	134 …	143 …	154 …	186
Freq	0	0	0	0	0	0	0	0	0	0	0

Step 2 : Count the frequency of numbers in the array.

Index	13 …	24 …	56 …	77 …	83 …	94	95 …	134 …	143 …	154 …	186
Freq	1	1	2	1	1	1	1	1	1	1	1

Step 3 : Modify Count Array by adding the previous count.

Index	13 …	24 …	56 …	77 …	83 …	94	95 …	134 …	143 …	154 …	186
Sum	0+1=1	1+1=2	2+2=4	4+1=5	5+1=6	6+1=7	7+1=8	8+1=9	9+1=10	10+1=11	11+1=12
New Freq	1	2	4	5	6	7	8	9	10	11	12

*Note: the …th indexes can be visually ignored because they have 0 frequency and eventually carries on the frequency of the previous index.

Step 4 : Create New Array (with the same size as Array A)

Index	0	1	2	3	4	5	6	7	8	9	10	11
Element												

Step 5 : Following the sequence in Array A, locate the element's frequency in Count Array & place it as element of (Index - 1) in New Array.

Array A

154	56	77	134	186	56	94	24	13	83	95	143

Count

Inde	13	24	56	77	83	94	95	134	143	154	186

Array	x											
	Freq	1	2	4	5	6	7	8	9	10	11	12

Array A pointing 154, placed at index 10 (11 - 1),

New Array	Index	0	1	2	3	4	5	6	7	8	9	10	11
	Element											154	

New Freq of 154 at Count Array = 10

Array A pointing 56, placed at index 3 (4 - 1),

New Array	Index	0	1	2	3	4	5	6	7	8	9	10	11
	Element				56							154	

New Freq of 54 at Count Array = 3

Array A pointing 77, placed at index 4 (5 - 1),

New Array	Index	0	1	2	3	4	5	6	7	8	9	10	11
	Element				56	77						154	

New Freq of 77 at Count Array = 4

Array A pointing 134, placed at index 8 (9 - 1),

New Array	Index	0	1	2	3	4	5	6	7	8	9	10	11
	Element				56	77				134		154	

New Freq of 134 at Count Array = 8

Array A pointing 186, placed at index 11 (12 - 1),

New Array	Index	0	1	2	3	4	5	6	7	8	9	10	11
	Element				56	77				134		154	186

New Freq of 186 at Count Array = 11

Array A pointing 56, note that it has appeared once and the frequency has been updated to 3, placed at index 2 (3 - 1),

New Array	Index	0	1	2	3	4	5	6	7	8	9	10	11
	Element			56	56	77				134		154	186

New Freq of 56 at Count Array = 2

Array A pointing 94, placed at index 6 (7 - 1),

New Array

Index	0	1	2	3	4	5	6	7	8	9	10	11
Element			56	56	77		94		134		154	186

New Freq of 94 at Count Array = 6

Array A pointing 24, placed at index 1 (2 - 1),

New Array

Index	0	1	2	3	4	5	6	7	8	9	10	11
Element		24	56	56	77		94		134		154	186

New Freq of 24 at Count Array = 1

Array A pointing 13, placed at index 0 (1 - 1),

New Array

Index	0	1	2	3	4	5	6	7	8	9	10	11
Element	13	24	56	56	77		94		134		154	186

New Freq of 13 at Count Array = 0

Array A pointing 83, placed at index 5 (6 - 1),

New Array

Index	0	1	2	3	4	5	6	7	8	9	10	11
Element	13	24	56	56	77	83	94		134		154	186

New Freq of 83 at Count Array = 5

Array A pointing 95, placed at index 7 (8 - 1),

New Array

Index	0	1	2	3	4	5	6	7	8	9	10	11
Element	13	24	56	56	77	83	94	95	134		154	186

New Freq of 95 at Count Array = 7

Array A pointing 143, placed at index 9 (10 - 1),

New Array

Index	0	1	2	3	4	5	6	7	8	9	10	11
Element	13	24	56	56	77	83	94	95	134	143	154	186

New Freq of 143 at Count Array = 9

Finally, the array has been sorted in ascending order.

13	24	56	56	77	83	94	95	134	143	154	186

3. Radix sort

Array A:

154	56	77	134	**186**	56	94	24	13	83	95	143

Sort according to the one's digit (least significant digit)

0:
1:
2:
3: 13, 83, 143
4: 154, 134, 94, 24
5: 95
6: 56, 186, 56
7: 77
8:
9:

Array A becomes:

13	83	143	154	134	94	24	95	56	186	56	77

Sort according to the ten's digit

0:
1: 13
2: 24
3: 134
4: 143
5: 154, 56, 56
6:
7: 77
8: 83, 186
9: 94, 95

Array A becomes:

13	24	134	143	154	56	56	77	83	186	94	95

Sort according to the hundred's digit (most significant digit)

0: 13, 24, 56, 56, 77, 83, 94, 95
1: 134, 143, 154, 186

2:
3:
4:
5:
6:
7:
8:
9:

Sorted Array A:

13	24	56	56	77	83	94	95	134	143	154	186

4. Bucket sort

Number of buckets: 5
Range = (maxValue + 1)/ number of bucket
= (186 + 1) / 5
= 37.4
≈ 37

0 - 37	24, 13
38 - 75	56, 56
76 - 113	77, 94, 83, 95
114 - 151	134, 143
152 - 189	154, 186

Use insertion sort to sort items in each bucket

0 - 37	24, 13
38 - 75	56, 56
76 - 113	77, 94, 83, 95
114 - 151	134, 143
152 - 189	154, 186

0 - 37	13, 24
38 - 75	56, 56
76 - 113	77, 94, 83, 95
114 - 151	134, 143
152 - 189	154, 186

0 - 37	13, 24
38 - 75	56, 56
76 - 113	77, 94, 83, 95
114 - 151	134, 143

152 - 189	154, 186

0 - 37	13, 24
38 - 75	56, 56
76 - 113	77, 83, 94, 95
114 - 151	134, 143
152 - 189	154, 186

Sorted array formed: <13, 24, 56, 56, 77, 83, 94, 95, 134, 143, 154, 186>

5. Shell sort

154	56	77	134	186	56	94	24	13	83	95	143

12 numbers to be sorted

Gap = n/2
= 12/2
= 6

154	56	77	134	186	56	94	24	13	83	95	143

94	56	77	134	186	56	154	24	13	83	95	143

94	24	77	134	186	56	154	56	13	83	95	143

94	24	13	134	186	56	154	56	77	83	95	143

94	24	13	83	186	56	154	56	77	134	95	143

94	24	13	83	95	56	154	56	77	134	186	143

94	24	13	83	95	56	154	56	77	134	186	143

Gap = 6/2
= 3

94	24	13	83	95	56	154	56	77	134	186	143

83	24	13	94	95	56	154	56	77	134	186	143

83	24	13	94	95	56	154	56	77	134	186	143

83	24	13	94	95	56	154	56	77	134	186	143

83	24	13	94	95	56	154	56	77	134	186	143

83	24	13	94	56	56	154	95	77	134	186	143

83	24	13	94	56	56	154	95	77	134	186	143

83	24	13	94	56	56	154	95	77	134	186	143

83	24	13	94	56	56	134	95	77	154	186	143

83	24	13	94	56	56	134	95	77	154	186	143

83	24	13	94	56	56	134	95	77	154	186	143

Gap = 3/2
= 1

83	24	13	94	56	56	134	95	77	154	186	143
24	83	13	94	56	56	134	95	77	154	186	143
24	13	83	94	56	56	134	95	77	154	186	143
13	24	83	94	56	56	134	95	77	154	186	143
13	24	83	94	56	56	134	95	77	154	186	143
13	24	83	56	94	56	134	95	77	154	186	143
13	24	56	83	94	56	134	95	77	154	186	143
13	24	56	83	94	56	134	95	77	154	186	143
13	24	56	83	56	94	134	95	77	154	186	143
13	24	56	56	83	94	134	95	77	154	186	143
13	24	56	56	83	94	134	95	77	154	186	143
13	24	56	56	83	94	134	95	77	154	186	143
13	24	56	56	83	94	95	134	77	154	186	143
13	24	56	56	83	94	95	134	77	154	186	143
13	24	56	56	83	94	95	77	134	154	186	143
13	24	56	56	83	94	77	95	134	154	186	143
13	24	56	56	83	77	94	95	134	154	186	143
13	24	56	56	77	83	94	95	134	154	186	143
13	24	56	56	77	83	94	95	134	154	186	143
13	24	56	56	77	83	94	95	134	154	186	143
13	24	56	56	77	83	94	95	134	154	186	143
13	24	56	56	77	83	94	95	134	154	143	186

13	24	56	56	77	83	94	95	134	143	154	186

13	24	56	56	77	83	94	95	134	143	154	186

PART 2 – Time complexity
Each of the algorithms in Part 1 is good for some conditions, and bad for others.
Find the time complexity for each of the algorithm and discuss what is the best application condition of the algorithm.

Bubble sort
Time complexity = O(n^2)

Best application condition: When the array or dataset to be sorted is very small or nearly sorted, bubble sort can be implemented easily and quickly, without requiring additional memory or complex logic.

Bad: When the array or dataset is large, it becomes very slow and inefficient.

Counting sort
Time complexity = O(k+n), where n is the number of elements in the input array and k is the range of input

Best application condition: When the range of input values (k) is relatively small.

Bad: In cases where the scope of input values (k) is significantly greater than the quantity of elements (n) present in the array, counting sort may encounter space complexity problems due to its high memory requirements.

Radix sort
Time complexity = O(d(n+k)), where d is the number of digits in the maximum element and k is the range of the input elements

Best application condition: When the input values are integers with a fixed number of digits. Also, the range of input values (k) is relatively small, and the number of digits (d) is fixed, it will very efficient and faster.

Bad: If the range of input values (k) is significantly larger than the number of elements (n) in the array, or if the number of digits (d) in the largest number is very large, hence, the time complexity of radix sort can become very high.

Bucket sort
Time complexity = O(n+k), where n is the number of elements to be sorted and k is the number of buckets.

Best application condition: When the input values are uniformly distributed over a range and the number of elements to be sorted is not too large, its time complexity can be close to O(n).

Bad: If the input values are not uniformly distributed over a range, it may not perform well. Besides, if the number of elements to be sorted is very large, it may require a large number of buckets, which can make the algorithm less efficient.

Shell sort
Time complexity:
Worst case: O(n log 2n), where n is the size of the array
Best case: O(n log n)

Best application condition: When the array is large, and the elements are randomly distributed.

Bad: When the array is highly ordered or reversed, shell sort may not be as effective.