WIA2005 Algorithm Design and Analysis
Tutorial 3

1. Bubble Sort

A:	56	77	134	56	34

2. Counting Sort

A:	9	15	13	7	10	5	5	11	9	6

3. Radix Sort

A:	459	95	22	792	63	7	186	11	39	372

4. Bucket Sort

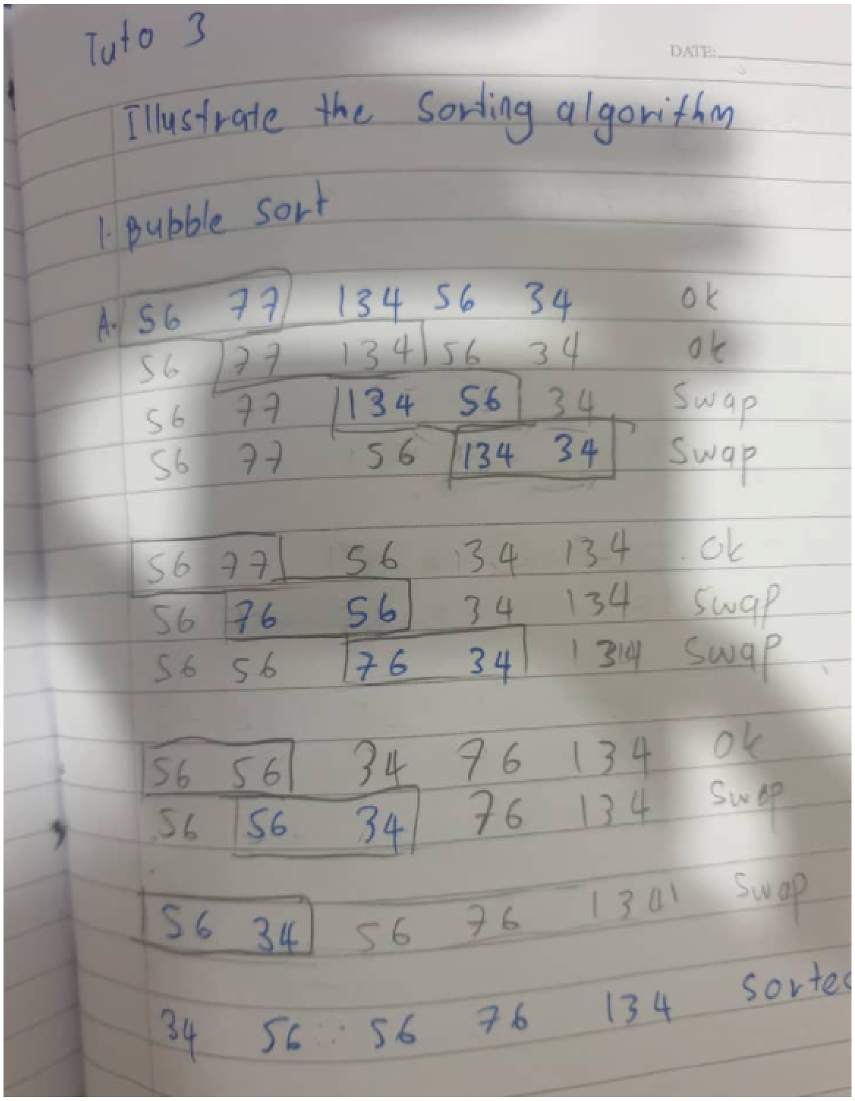
A:	58	92	67	76	69	73	81	64	75	97

5. Shell Sort

A:	46	198	27	73	4	10	112	18	65	85

- Each of the algorithms in Part 1 is good for some condition, and bad for others.
- Find the time complexity for each of the algorithms and discuss what is the best application condition of the algorithm.
- Include the algorithm used for the algorithm.
- Discuss the classification of each algorithm as well.

Bubble Sort



Tuto 3

DATE:

Illustrate the Sorting algorithm

1. Bubble sort

A. 56 77 134 56 34 ok

56 77 134 56 34 ok

56 77 134 56 34 Swap

56 77 56 134 34 Swap

56 77 56 34 134 ok

56 76 56 34 134 Swap

56 56 76 34 134 Swap

56 56 34 76 134 ok

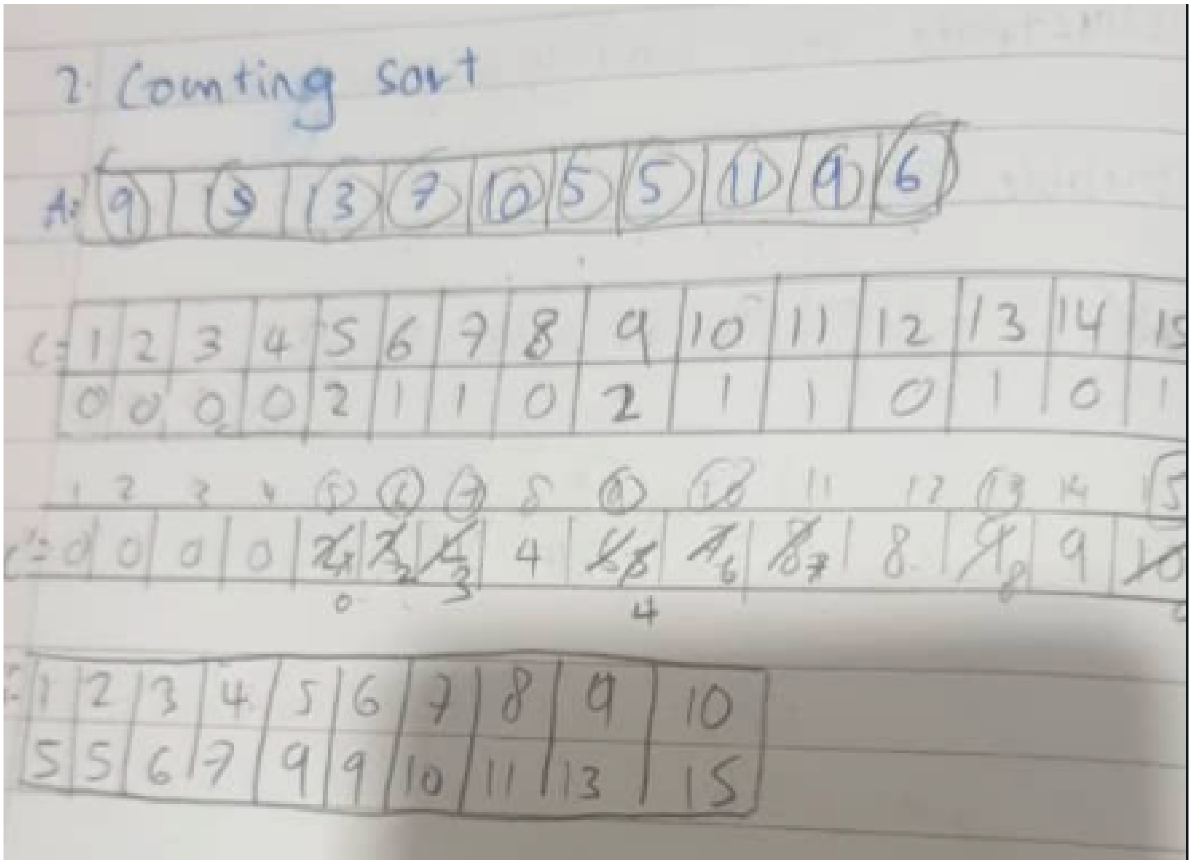
56 56 34 76 134 Swap

56 34 56 76 134 Swap

34 56 56 76 134 Sorte

Classifications

- In-Place
- Stable
- Adaptive - taking advantage of the existing order of the array
- Offline



2. Counting sort

A:

9	15	13	7	10	5	5	11	9	6

C:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	2	1	1	0	2	1	1	0	1	0	1

C':

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	~~2~~ 1 0	~~3~~ 2	~~4~~ 3	4	~~6~~ 5 4	~~7~~ 6	~~8~~ 7	8	~~9~~ 8	9	~~10~~

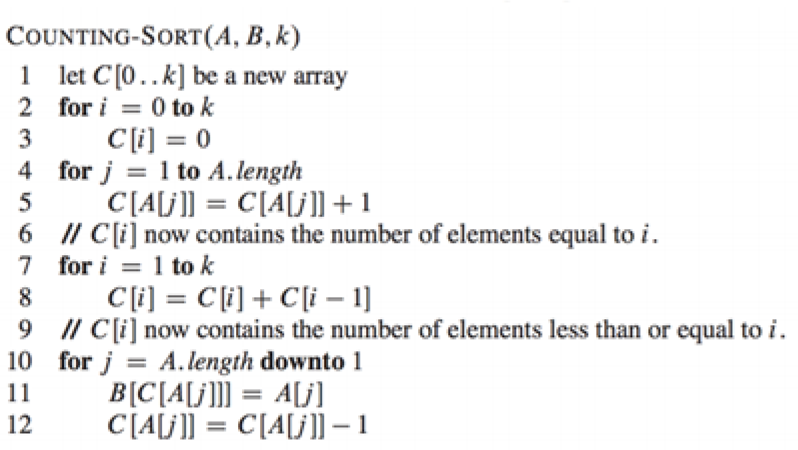
B:

1	2	3	4	5	6	7	8	9	10
5	5	6	7	9	9	10	11	13	15

Counting sort

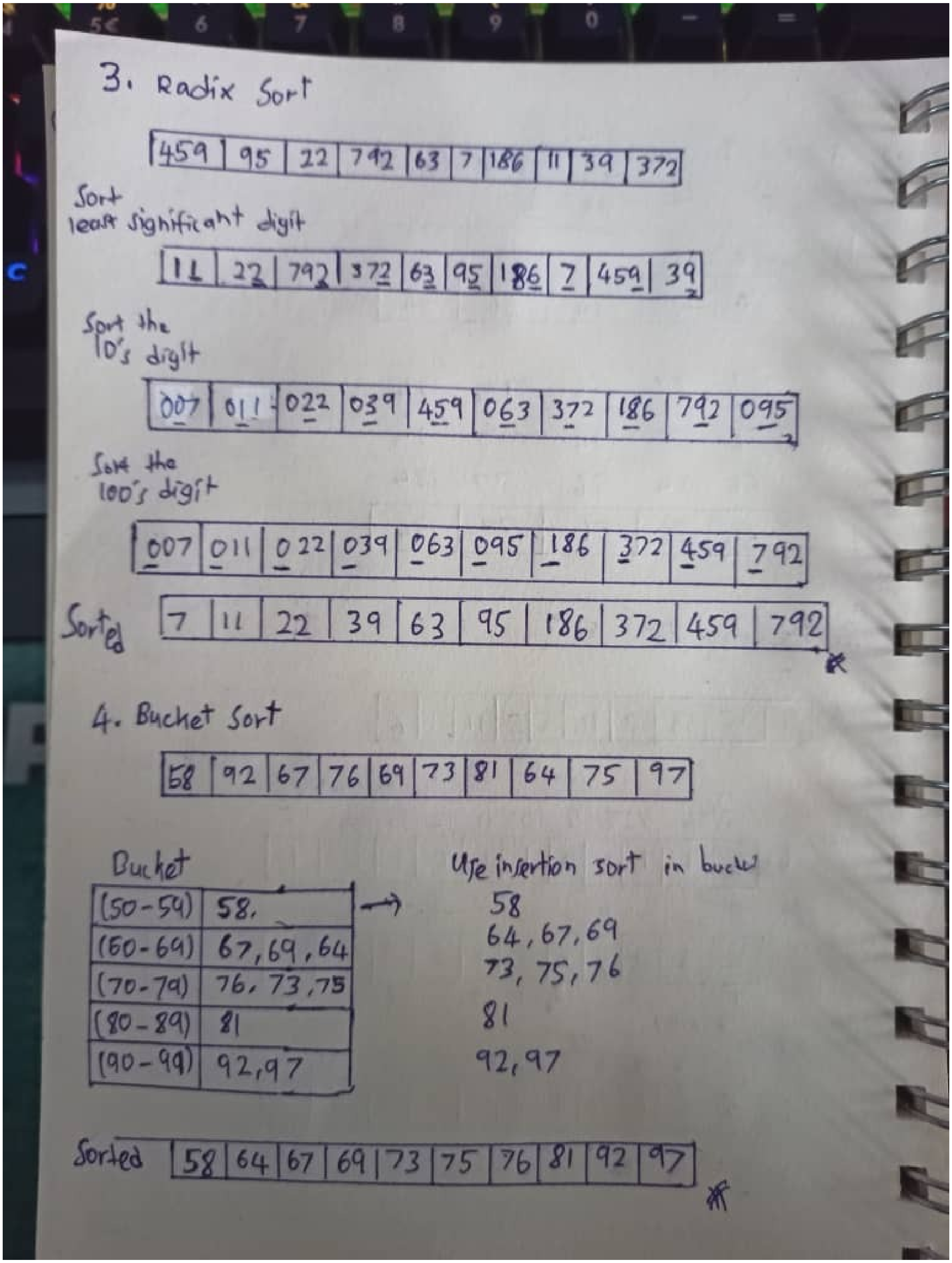
Classification

- Not in-place - have to create an additional Array to carry out the counting and running sum
- Stable -
- Non-adaptive
- Offline



```
COUNTING-SORT($A, B, k$)
let $C[0\,..\,k]$ be a new array
for $i = 0$ to $k$
    $C[i] = 0$
for $j = 1$ to $A.length$
    $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to $i$.
for $i = 1$ to $k$
    $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to $i$.
for $j = A.length$ downto 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$
```

Radix sort



3. Radix Sort

459	95	22	792	63	7	186	11	39	372

Sort least significant digit

11	22	792	372	63	95	186	7	459	39

Sort the 10's digit

007	011	022	039	459	063	372	186	792	095

Sort the 100's digit

007	011	022	039	063	095	186	372	459	792

Sorted

7	11	22	39	63	95	186	372	459	792

4. Bucket Sort

58	92	67	76	69	73	81	64	75	97

Bucket

(50-59)	58.
(60-69)	67,69,64
(70-79)	76, 73,75
(80-89)	81
(90-99)	92,97

→ Use insertion sort in bucke

58
64,67,69
73, 75,76
81
92,97

Sorted

58	64	67	69	73	75	76	81	92	97

Running time=$\Theta(d(n^2))$
Θ(d(n^2))

If using bubble sort -> Θ(d(n^2))

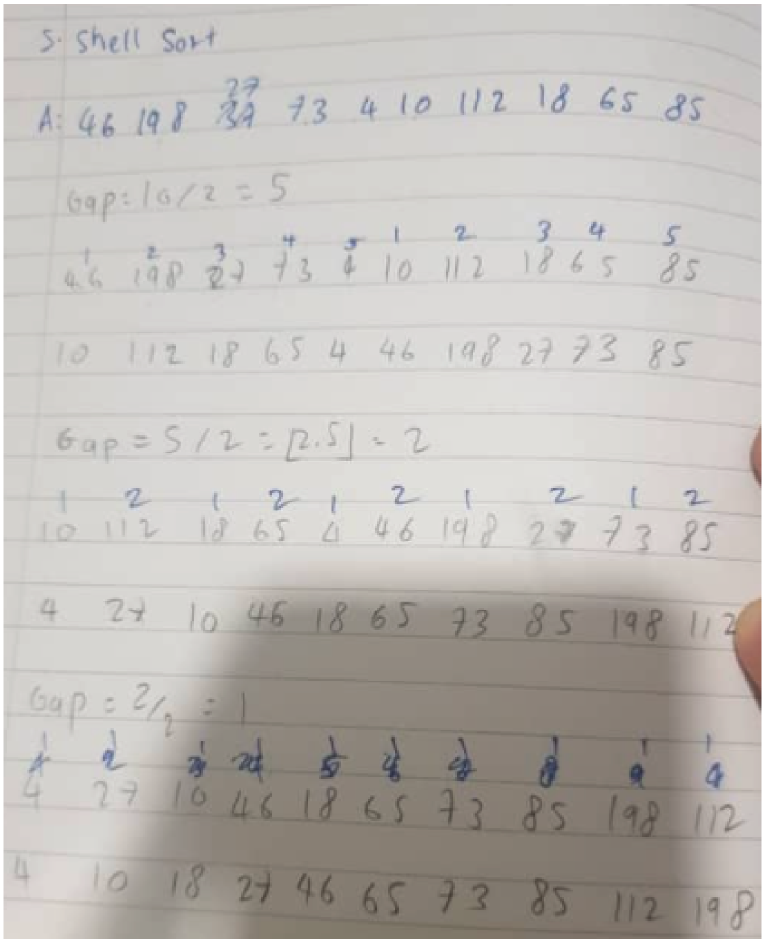
Classification:

- Not In-place
- Stable
- Offline
-

Bucket sort

- Not in-place
- Stable
- Non-adaptive
- Online as long as the numbers in the bucket had not been sorted?

Shell sort



5. Shell Sort

A: 46 198 ~~37~~ 27 73 4 10 112 18 65 85

Gap: 10/2 = 5

1 2 3 4 5 1 2 3 4 5
46 198 27 73 4 10 112 18 65 85

10 112 18 65 4 46 198 27 73 85

Gap = 5/2 = ⌊2.5⌋ = 2

1 2 1 2 1 2 1 2 1 2
10 112 18 65 4 46 198 27 73 85

4 27 10 46 18 65 73 85 198 112

Gap = 2/2 = 1

1 1 1 1 1 1 1 1 1 1
~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~4~~
4 27 10 46 18 65 73 85 198 112

4 10 18 27 46 65 73 85 112 198

- In-place
- Unstable - may change relative to the order of elements

- Adaptive - it helps if the input is already sorted
- Offline - Gap depends on the input (total number of elements)