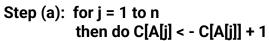
DEPARTMENT OF CYBER SECURITY



DAA Home Work 4

SUBMITTED BY	211042 Noman Masood Khan A
SEMESTER	Fifth
SUBJECT	DAA
SUBMITTED TO	Sir Ammar Masood

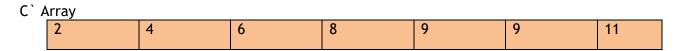
8.2-1: illustrate the operation of COUNTING SORT on the array $A = \{6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2\}$



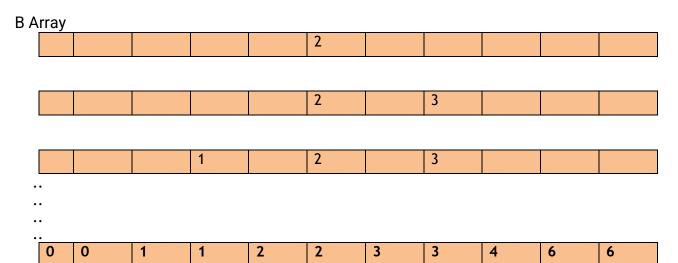
A array

	6	0	2	0	1	3	4	6	1	3	2
C A	rray										
	2		2	2		2		1	0		2

Step (b): C[i] = C[i] + C[i-1]



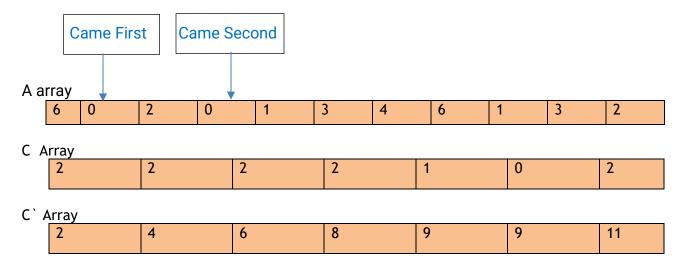
Step (c): B[C[A[j]]] = A[j]



8.2-3: Suppose that we were to rewrite the for loop header in line 11 of the COUNTING-SORT

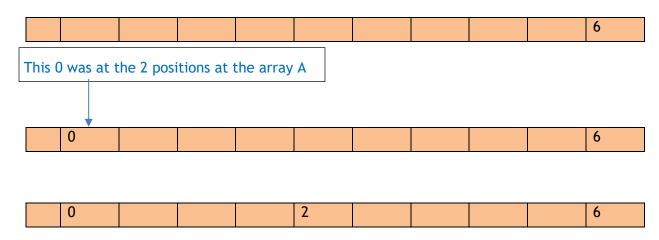
♣ Suppose we have this array : A = {6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2}.

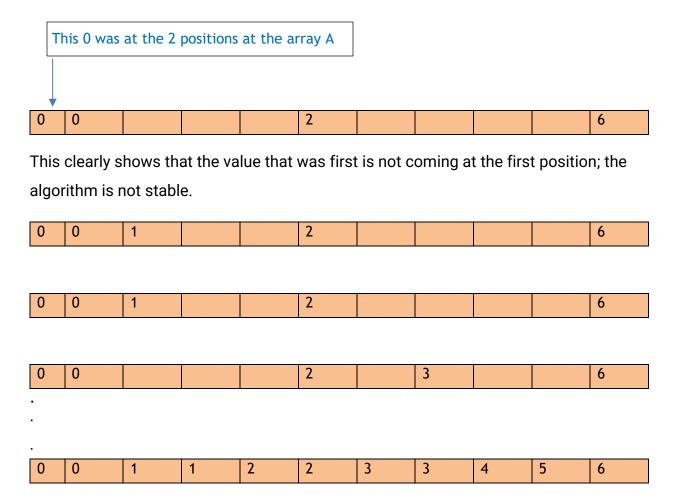
In the Previous question, I have sorted an array with the counted sorting and now I am taking that sorting an example. I already have solved the C and C` Array in the previous so now I am



Step (c): Now the last loop will run from the j = 1 down to 11

B Array





♣ Clearly Showed that the algorithm sorts the elements very well but not stable.

The code for the increasing index with the elements being sorted and stable order.

8.2-5: Suppose that the array being sorted contains only integers in the range 0 to k and that there are no satellite data.

```
11 for j = n downto 1

12 B[C[A[j]]] = A[j]

13 C[A[j]] = C[A[j]] - 1 // to handle duplicate values

14 return B
```

```
COUNTING-SORT(A, n, k)

1let B[1:n] and C [0:k] be new arrays

2for i = 0 to k

3 C [i] = 0

4for j = 1 to n

5 C [A[j]] = C [A[j]] + 1

6// C [i] now contains the number of elements equal to i.

7for 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// Copy A to B, starting from the end of A.
```

```
ror j = n down to 1

value = C[ A[ j ] ]

swap = A[value]

A[value] = A[i]

A[i] = swap
```

Return A

8.2-3: illustrate the operation of RADIX-SORT on the following list of English words:

С	0	W
D	0	G
S	E	A G
R	U	
R	0	W
М	0	В
В	0	X
Т	Α	В
В	A	R
Е		R
Т	Α	R
D	I	G
В	I	G
Т	E	A
N	0	W
F	0	X

S	E	Α
Т	Ε	Α
M	0	В
Т	Α	В
D	0	G
R	U	G
D	I	G
В	I	G
В	A	R
Е	A	R
Т		R
С	0	W
R	0	W
N	0	W
В	0	X
F	0	Х

Т	Α	В
В	A	R
Е	A	R
Т	Α	R A
S	E	Α
Т	Е	Α
D	I	G G
В	I	G
M	0	В
D	0	G
С	0	W
R	0	
N	0	W
В	0	Χ
F	0 U	X
R	U	G

В	Α	R
В	I	G
В	0	Χ
С	0	W
D	I	G
D	0	G
E	Α	R
F	0	Χ
M	0	В
N	0	W
R	0	W
R	U	G
S	E	A B
Т	Α	В
Т	Α	R
Т	E	Α

8.3-5: Show how to sort n integers in the range 0 to $n^3 - 1$ in O(n) time

- ♣ Using the Counting Sorting will take O(n^2) time. Likewise, using the Merge Sorting, Heap sorting and others will take O(nlogn) time or some other time which is not O(n).
- ♣ Radix Sort will be the Sort that will have the O(n) time.

The Running Time complexity of the Radix sort is O(d*(n + b)).

Where the d is the number of the digits; n is the integers and b is the base of the n.

- > Prove following the following Steps:
- \checkmark Since the numbers are in base n, the range of digits is 1 to n so k = n.
- ✓ The number of passes needed is 3 since n^2 = 100_n so d = 3.
- ✓ The running time of the Radix Sort is $\Theta(d^*n + d^*k) = \Theta(3n + 3n) = \Theta(n) \in O(n)$.

9.1-2: You want to find a number that is neither the minimum nor the maximum. The minimum number of steps for finding that.

Only 1 comparison is need in this function but the rest of the comparisons were done in the maximum and the minimum function.

9.2-2: Write an iterative version of RANDOMIZED-SELECT.

```
RANDOMIZED-SELECT(A, p, r, i)

while p < r do

q \leftarrow RANDOMIZED-PARTITION(A, p, r)

k \leftarrow q - p + 1

if i \le k then

r \leftarrow q

else

p \leftarrow q + 1

i \leftarrow i - k
```

9.2-4: Argue that the expected running time of RANDOMIZED-SELECT does not depend on the order of the elements in its input array A[p : r].

Argument

The algorithm of the randomized-select does not depend on the order of the elements in input array. Therefore, the expected running time for the randomized-select is same for any permutation of the input array[p:r].

Prove By Induction

Base Case:

When the input array has only one element, the randomized-select will simply return one element regardless of its position in the array. The time complexity remains constant and not effected by the order of elements.

Inductive Step:

Assume that the claim is true for an Array [n-1].

For an input array of length n-1, RANDOMIZED-SELECT works as follows:

- Randomly selects a pivot element from the array.
- Partitions the array around the pivot, placing elements smaller than the pivot to its left and larger elements to its right.
- Recursively runs RANDOMIZED-SELECT one of the (left or right) array based on the pivot's position
- The expected running time of the partitioning is $\Theta(n)$ which does not depend on the order of the element.
- The expected running time of the recursive calls is T(max(i, k) min(i, k).
- The total running time of the algorithm is $T(n) = \theta(n) + T(\max(i, k) \min(i, k))$.

The induction step has shown us the time complexity for the any array of the length will have the same running time; this means that the running time of an algorithm does not depend on the order of the element.