SORT n NUMBERS WHERE EACH NUMBER
$\in [1 \ldots B]$

100 11 225 6 89 30 60

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100 11 225 6 89 30 60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	. . . .	0	0
	1	2	3	4	. . . .		B

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100 11 225 6 89 30 60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	....	1	....	0 0
	1	2	3	4	....	100		B

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100   11   225   6   89   30   60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	...	1	...	1	....	0	0
	1	2	3	4	..	11		100			B

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100 11 225 6 89 30 60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	...	1	...	1	....	0	0
	1	2	3	4	..	11		100			B

Q: HOW WILL YOU FIND THE FINAL SORTED ARRAY?

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100   11   225   6   89   30   60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	...	1 ...	1	....	0	0
	1	2	3	4	..	11	100			B

Q: HOW WILL YOU FIND THE FINAL SORTED ARRAY?

A: GO OVER THE ARRAY FROM LEFT TO RIGHT

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100 11 225 6 89 30 60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	...	1	...	1	....	0	0
	1	2	3	4	..	11		100			B

Q: HOW WILL YOU FIND THE FINAL SORTED ARRAY?

A: GO OVER THE ARRAY FROM LEFT TO RIGHT

FOR $i \leftarrow 1$ TO n

  $Q[A[i]] \leftarrow 1;$

FOR $i \leftarrow 1$ TO B

{ IF

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100   11   225   6   89   30   60

MAKE AN ARRAY Q OF SIZE B

Q

0	0	0	0	...	1	...	1	....	0	0
1	2	3	4	..	11		100			B

Q: HOW WILL YOU FIND THE FINAL SORTED ARRAY?

A: GO OVER THE ARRAY FROM LEFT TO RIGHT

```
FOR i ← 1 TO n
    Q[A[i]] ← 1;
FOR i ← 1 TO B
{   IF Q[i] = 1
        PRINT i
}
```

RUNNING TIME

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \ldots B]$

100   11   225   6   89   30   60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	...	1 ...	1	....	0	0
	1	2	3	4	..	11	100			B

Q: HOW WILL YOU FIND THE FINAL SORTED ARRAY?

A: GO OVER THE ARRAY FROM LEFT TO RIGHT

```
FOR i ← 1 TO n                 ] O(n)
    Q[A[i]] ← 1;
FOR i ← 1 TO B                 ] O(B)
{   IF Q[i] = 1
          PRINT i
}
```

RUNNING TIME : $O(n+B)$

SPACE :

SORT n NUMBERS WHERE EACH NUMBER $\in [1 \dots B]$

100   11   225   6   89   30   60

MAKE AN ARRAY Q OF SIZE B

Q	0	0	0	0	...	1 ...	1	....	0	0
	1	2	3	4	..	11	100			B

Q: HOW WILL YOU FIND THE FINAL SORTED ARRAY?

A: GO OVER THE ARRAY FROM LEFT TO RIGHT

```
FOR i ← 1 TO n                  ]
    Q[A[i]] ← 1;                ] O(n)
FOR i ← 1 TO B                  ]
{   IF  Q[i] = 1                ] O(B)
          PRINT i               ]
}                               ]
```

RUNNING TIME : $O(n+B)$

SPACE : $O(n+B)$

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

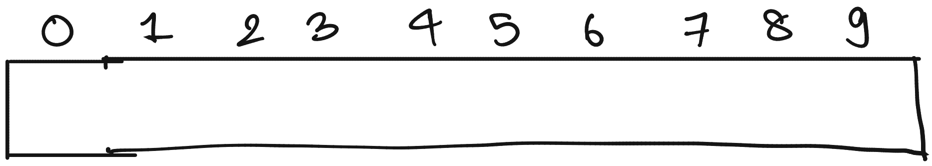
001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?



WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?

0	1	2	3	4	5	6	7	8	9
001									

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?

0	1	2	3	4	5	6	7	8	9
001									991

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

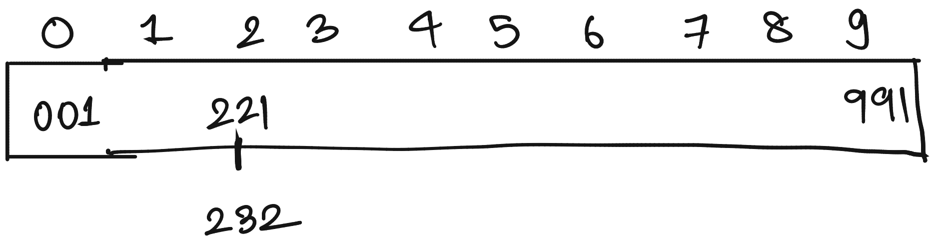
Q: WHAT IS THE SIZE OF ARRAY Q?

0	1	2	3	4	5	6	7	8	9
001		221							991

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?



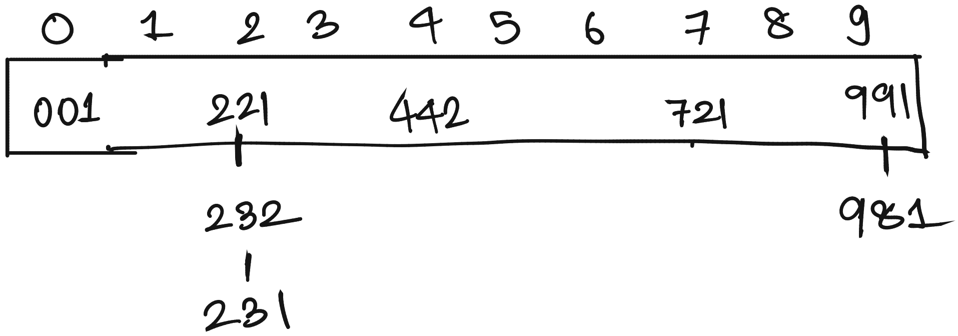
A LIST AT EACH CELL IN THE ARRAY

ASSUME THAT YOU CAN ADD AN ELEMENT IN THE LIST IN O(1) TIME

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

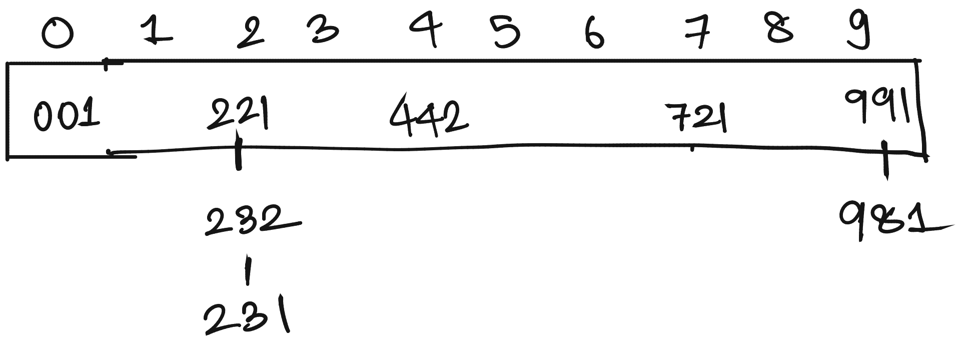
Q: WHAT IS THE SIZE OF ARRAY Q?



WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?

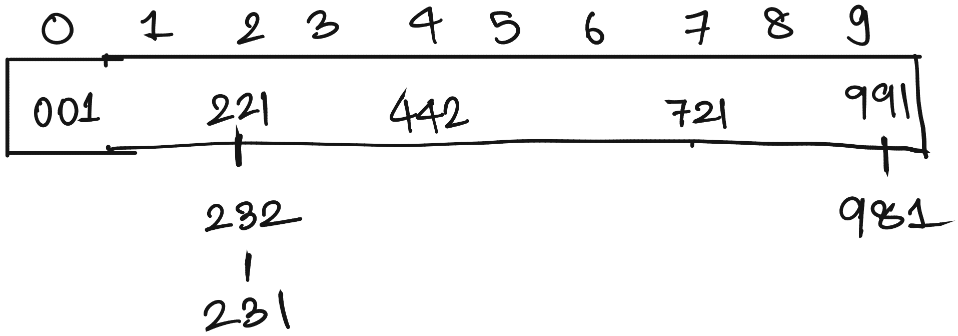


001 221 232 231 442 721 991 981

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?



001 221 232 231 442 721 991 981

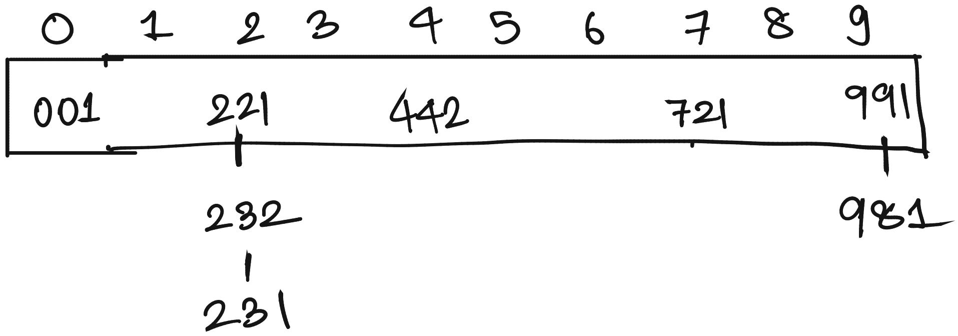
BUCKETSORT(A[1...n], d)
{ MAKE A NEW ARRAY Q[0...9];
FOR $i \leftarrow 1$ TO n
{ IF d^{th} DIGIT OF A[i] IS K
ADD A[i] TO THE LIST IN
Q[K];
}

ADD THE ELEMENTS FROM Q TO
A FROM LEFT TO RIGHT
}

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?



001 221 232 231 442 721 991 981

BUCKETSORT(A[1...n], d)
{ MAKE A NEW ARRAY Q[0...9];

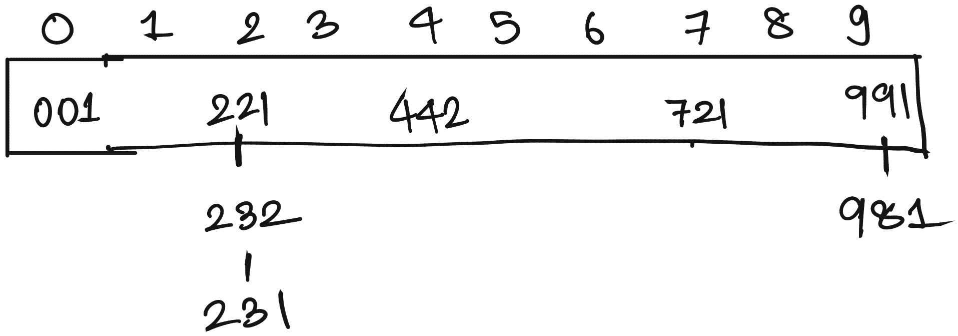
O(n) FOR i ← 1 TO n
{ IF d^{th} DIGIT OF A[i] IS K
ADD A[i] TO THE LIST IN Q[K];
}

ADD THE ELEMENTS FROM Q TO A FROM LEFT TO RIGHT
}

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?



001 221 232 231 442 721 991 981

BUCKETSORT(A[1...n], d)

{ MAKE A NEW ARRAY Q[0...9];

$O(n)$

FOR $i \leftarrow 1$ TO n

{ IF d^{th} DIGIT OF A[i] IS K

ADD A[i] TO THE LIST IN Q[K];

}

$O(n+10)$

ADD THE ELEMENTS FROM Q TO A FROM LEFT TO RIGHT

}

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?

0	1	2	3	4	5	6	7	8	9
001		221		442			721		991
		232							981
		231							

001 221 232 231 442 721 991 981

BUCKETSORT(A[1...n], d)
{ MAKE A NEW ARRAY Q[0....9];

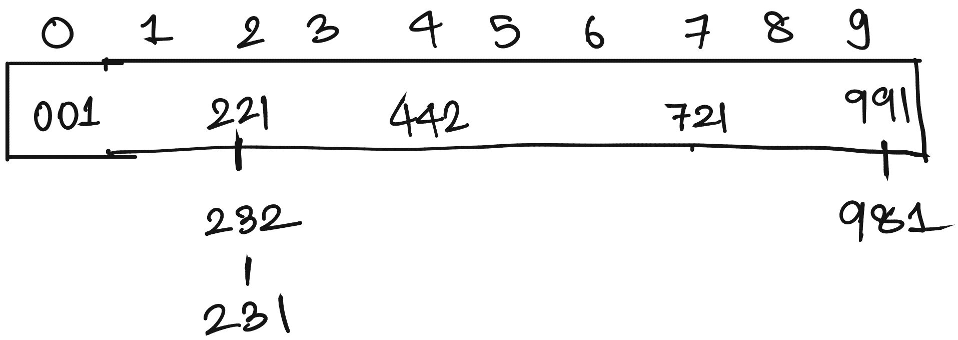
O(n) FOR i ← 1 TO n
{ IF dth DIGIT OF A[i] IS K
ADD A[i] TO THE LIST IN Q[K];
}

O(n+10) ADD THE ELEMENTS FROM Q TO A FROM LEFT TO RIGHT
}
RUNNING TIME = O(n)

WHAT IF WE SORT THESE NUMBERS
BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?



001 221 232 231 442 721 991 981

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?

0	1	2	3	4	5	6	7	8	9
001		221		442			721		991
		232							981
		231							

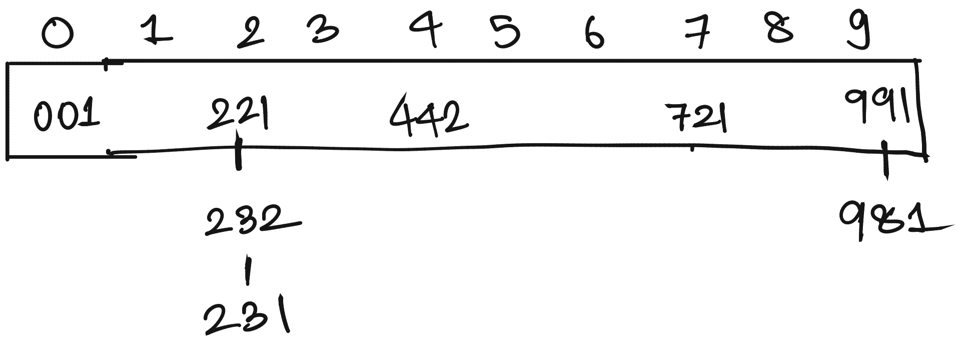
001 221 232 231 442 721 991 981

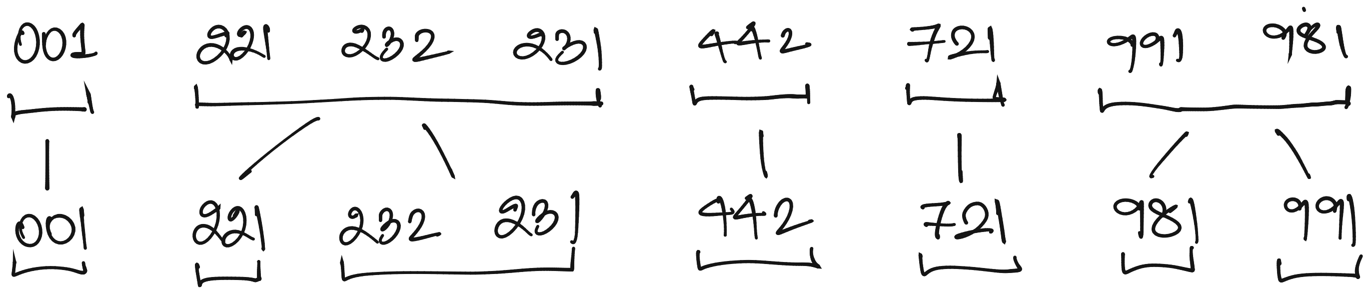
BUCKET SORT ON THESE SUBARRAYS USING THE SECOND MOST SIGNIFICANT DIGIT

WHAT IF WE SORT THESE NUMBERS
BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q ?

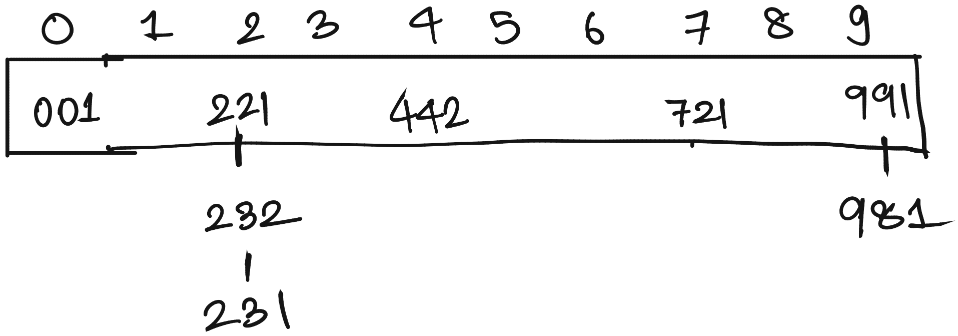


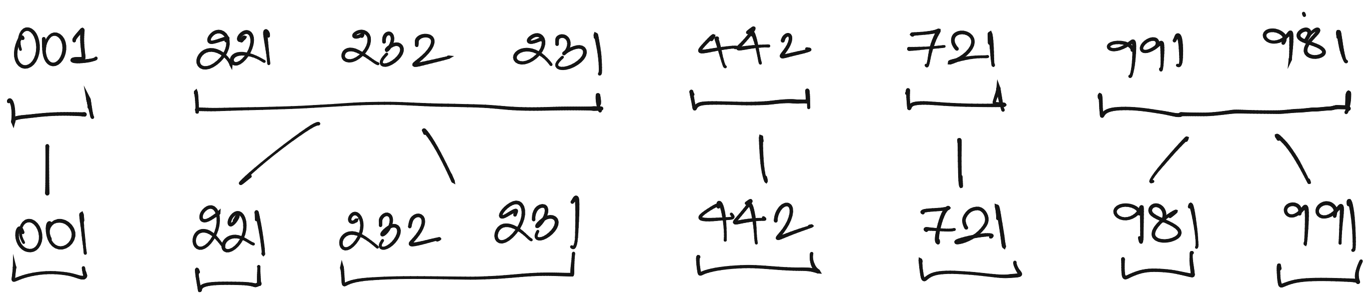


WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?



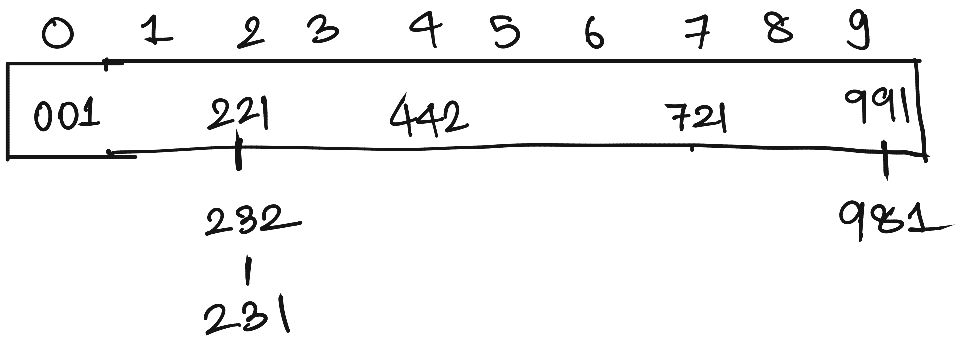


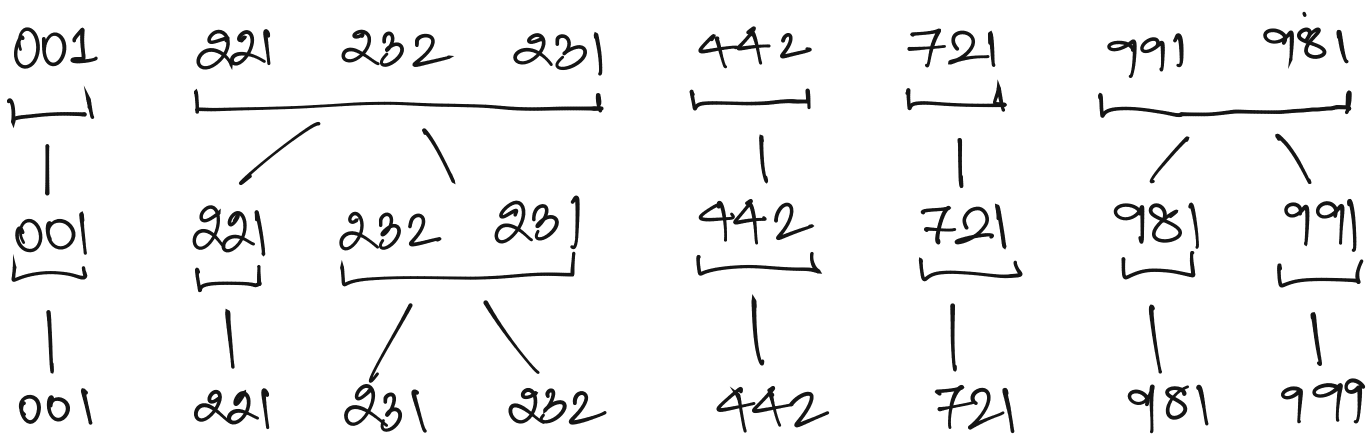
BUCKET SORT ON ALL THESE SUBARRAY USING THE THIRD MOST SIGNIFICANT BIT

WHAT IF WE SORT THESE NUMBERS BASED ON THEIR MOST SIGNIFICANT NUMBER

001 991 221 232 981 721 231 442

Q: WHAT IS THE SIZE OF ARRAY Q?





RADIX SORT $(A[1 \ldots n], d)$

{ $\quad A \leftarrow$ BUCKETSORT (A, d)

RADIX SORT $(A[1 \ldots n], d)$

{ $\quad A \leftarrow$ BUCKETSORT (A, d)

// NEED TO FIND SUBARRAY STARTING WITH

// NUMBER 0

RADIX SORT $(A[1 \ldots n], d)$

{ $A \leftarrow$ BUCKETSORT (A, d)

// NEED TO FIND SUBARRAY STARTING WITH

// NUMBER 0

DIGIT $\leftarrow 0$, $i \leftarrow 1$; START $\leftarrow 1$;

WHILE (TRUE)

{ IF

```
RADIX SORT (A[1...n], d)
{     A ← BUCKETSORT (A, d)
      // NEED TO FIND SUBARRAY STARTING WITH
      // NUMBER 0
      DIGIT ← 0;   i ← 1; START ← 1
      WHILE(TRUE)
      {     IF d^th DIGIT OF A[i] > DIGIT
            {     END ← i-1;
                  IF   END - START +1  > 0
```

```
RADIX SORT (A[1...n], d)
{    A ← BUCKETSORT (A, d)
     // NEED TO FIND SUBARRAY STARTING WITH
     // NUMBER 0
     DIGIT ← 0;  START ← 1;  i ← 1
     WHILE (TRUE)
     {     IF d^th DIGIT OF A[i] > DIGIT
           {  END ← i-1
              IF   END - START + 1 > 0
              {    BUCKETSORT (A[START.... END],
                                          d-1)

                   START ← i;  DIGIT ← DIGIT+1
              }

              ELSE
              {
```

```
RADIX SORT (A[1...n], d)
{       A ← BUCKETSORT (A, d)
        // NEED TO FIND SUBARRAY STARTING WITH
        // NUMBER 0
        DIGIT ← 0;  START ← 1;  i ← 1
        WHILE (TRUE)
        {       IF d^th DIGIT OF A[i] > DIGIT
                {   END ← i-1
                    IF   END - START +1 > 0
                    {       BUCKETSORT (A [START.... END],
                                                  d-1)

                         START ← i;  DIGIT ← DIGIT+1
                    }

                    ELSE
                         DIGIT ← DIGIT+1;
```

```
RADIX SORT ( A[1...n], d)
{     A ← BUCKETSORT (A, d)
      // NEED TO FIND SUBARRAY STARTING WITH
      // NUMBER 0
      DIGIT ← 0;  START ← 1 ;  i ← 1
      WHILE ( TRUE)
      {      IF  i > n
                 BREAK ;
             IF  d^th DIGIT OF A[i] > DIGIT
             {  END ← i-1 ;
                IF   END - START +1  > 0
                {    BUCKETSORT ( A [START.... END],
                                                d-1 )
                     START ← i ;  DIGIT ← DIGIT+1
                }
                ELSE
                     DIGIT ← DIGIT+1 ;
             }
             ELSE
```

```
RADIX SORT (A[1...n], d)
{    A ← BUCKETSORT (A, d)
     // NEED TO FIND SUBARRAY STARTING WITH
     // NUMBER 0
     DIGIT ← 0;  START ← 1;  i ← 1
     WHILE (TRUE)
     {     IF  i > n
               BREAK;
           IF  d^th DIGIT OF A[i] > DIGIT
           {  END ← i-1;
              IF   END - START + 1 > 0
              {    BUCKETSORT (A [START.... END], d-1)
                   START ← i;  DIGIT ← DIGIT+1
              }
              ELSE
                   DIGIT ← DIGIT+1;
           }
           ELSE
               i ← i+1;
     }
}
```

```
RADIX SORT (A[1...n], d)
{ o(n) [A ← BUCKETSORT (A, d)
      // NEED TO FIND SUBARRAY STARTING WITH
      // NUMBER 0
  O(1) [DIGIT ← 0; START ← 1; i ← 1
      WHILE (TRUE)
      {     IF  i > n
                BREAK;
            IF d^th DIGIT OF A[i] > DIGIT
            { END ← i-1;
              IF  END - START + 1 > 0
              {   BUCKETSORT (A[START.... END], d-1)

                  START ← i; DIGIT ← DIGIT+1
              }

              ELSE
                  DIGIT ← DIGIT+1;
            }
            ELSE
                i ← i+1;
      }
}
```

```
RADIX SORT (A[1...n], d)
{ O(n) [A ← BUCKETSORT (A, d)
        // NEED TO FIND SUBARRAY STARTING WITH
        // NUMBER 0
  O(1) [DIGIT ← 0; START ← 1; i ← 1
O(n+10) [WHILE (TRUE)
        {     IF  i > n
                  BREAK;
              IF d^th DIGIT OF A[i] > DIGIT
              {  END ← i-1;
                 IF   END - START +1 > 0
                 {    BUCKETSORT (A[START.... END], d-1)

                      START ← i; DIGIT ← DIGIT+1
                 }
                 ELSE
                      DIGIT ← DIGIT+1;
              }
              ELSE
                   i ← i+1;
        }
}
```

```
RADIX SORT ( A[1...n], d)
{ O(n) [A ←  BUCKETSORT (A, d)
        // NEED TO FIND SUBARRAY STARTING WITH
        // NUMBER 0
  O(1) [DIGIT ← 0;  START ← 1 ; i ← 1
O(n+10) [ WHILE ( TRUE)
        {     IF   i > n               ] O(n)
                   BREAK;
     O(n)  - IF  d^th DIGIT OF A[i] > DIGIT
              { [END ← i-1;
           10   [IF   END - START +1  > 0
                 {    BUCKETSORT ( A [START.... END], d-1)

          10 [          START ← i; DIGIT ← DIGIT+1
                }

                ELSE
          10 [          DIGIT ← DIGIT+1;
              }
              ELSE
                    i ← i+1 ;          } O(n)
        }
}
```

$T(n, d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$\Rightarrow \quad T(n, d) =$

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

$T(n, d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

BASE CASE:

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1,\ d-1) + cn.$$

BASE CASE: $d = 1$

$$T(n,1) =$$

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1,\ d-1) + cn.$$

BASE CASE: $d = 1$

$$T(n,1) = cn$$

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

$$T(n,1) = cn$$

A USEFUL METHOD, GUESS AND PROVE

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

$$T(n,1) = cn$$

A USEFUL METHOD, GUESS AND PROVE

$$T(n,d) \leq cdn$$

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

$$T(n,1) = cn$$

A USEFUL METHOD, GUESS AND PROVE

$$T(n,d) \leq cdn$$

BASE CASE : $d = 1$

$T(n,d) \leq cn \Rightarrow$ WHICH IS INDEED TRUE

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

$$T(n,1) = cn$$

A USEFUL METHOD, GUESS AND PROVE

$$T(n,d) \leq cdn$$

BASE CASE : $d=1$

$T(n,d) \leq cn \Rightarrow$ WHICH IS INDEED TRUE

INDUCTION HYPOTHESIS

$$T(n,d-1) \leq c(d-1)n$$

T.P.T $\quad T(n,d) \leq cdn$

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

$$T(n,1) = cn$$

A USEFUL METHOD, GUESS AND PROVE

$$T(n,d) \leq cdn$$

BASE CASE : $d=1$

$T(n,d) \leq cn \Rightarrow$ WHICH IS INDEED TRUE

INDUCTION HYPOTHESIS

$$T(n,d-1) \leq c(d-1)n$$

T.P.T $T(n,d) \leq cdn$

$$T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn$$

$$\leq \sum_{i=0}^{9} c(d-1)(\text{END}_i - \text{START}_i + 1) + cn$$

$T(n,d) \leftarrow$ RUNNING TIME TO SORT n NUMBERS OF d DIGITS

$$\Rightarrow \quad T(n,d) = \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn.$$

$$T(n,1) = cn$$

A USEFUL METHOD, GUESS AND PROVE

$$T(n,d) \le cdn$$

BASE CASE : $d=1$

$T(n,d) \le cn \Rightarrow$ WHICH IS INDEED TRUE

INDUCTION HYPOTHESIS

$$T(n,d-1) \le c(d-1)n$$

T.P.T $\quad T(n,d) \le cdn$

$$
\begin{aligned}
T(n,d) &= \sum_{i=0}^{9} T(\text{END}_i - \text{START}_i + 1, d-1) + cn \\
&\le \sum_{i=0}^{9} c(d-1)(\text{END}_i - \text{START}_i + 1) + cn \\
&= c(d-1)n + cn \\
&= cdn
\end{aligned}
$$

CORRECTNESS:

CORRECTNESS:

<u>LEMMA</u>: IF abc < def THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

CORRECTNESS:

<u>LEMMA</u>: IF abc < def THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

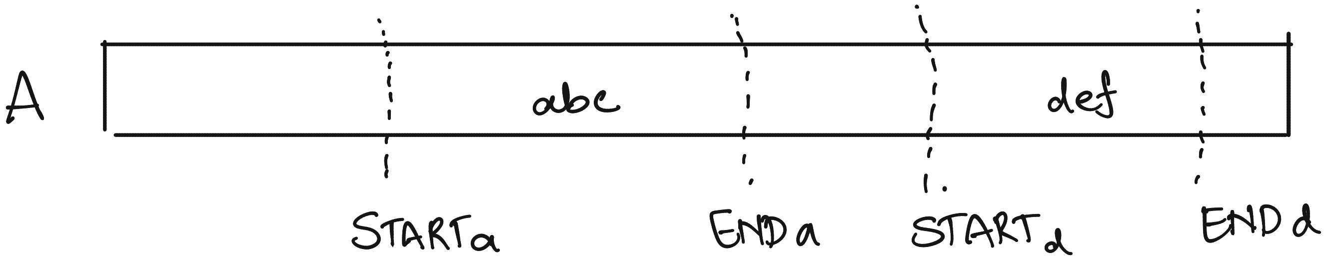
$\Downarrow$

WHY DOES THIS IMPLY THAT THE ARRAY IS SORTED ?

CORRECTNESS:

LEMMA: IF abc < def THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

PROOF: (1) If a < d

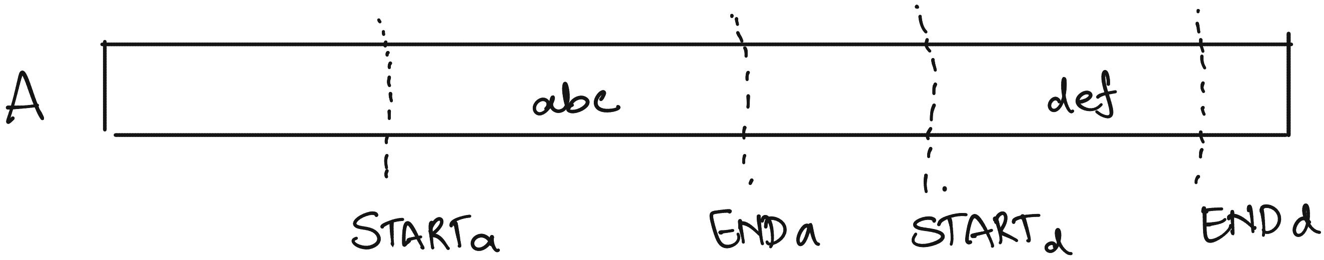


Q: CAN IT EVER HAPPEN THAT WE PUT def TO THE LEFT OF abc IN FUTURE

CORRECTNESS:

<u>LEMMA</u>: IF abc < def THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

<u>PROOF</u>: (1) If $a < d$



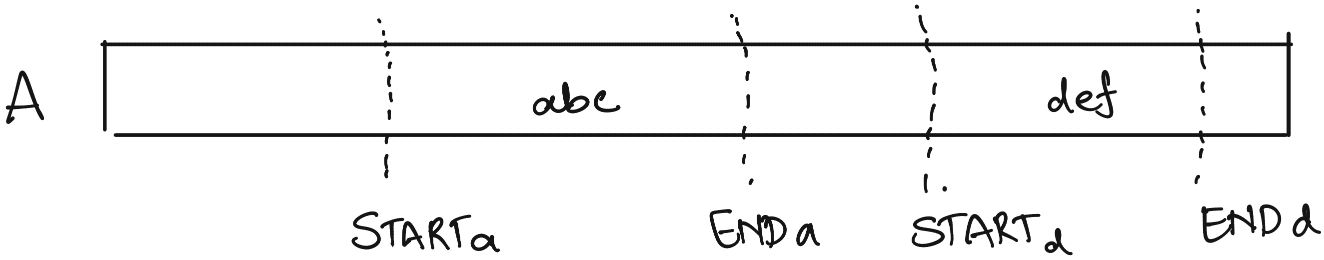
<u>Q</u>: CAN IT EVER HAPPEN THAT WE PUT def TO THE LEFT OF abc IN FUTURE

A: NO

CORRECTNESS:

<u>LEMMA</u>: IF $abc < def$ THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

<u>PROOF</u>: (1) If $a < d$

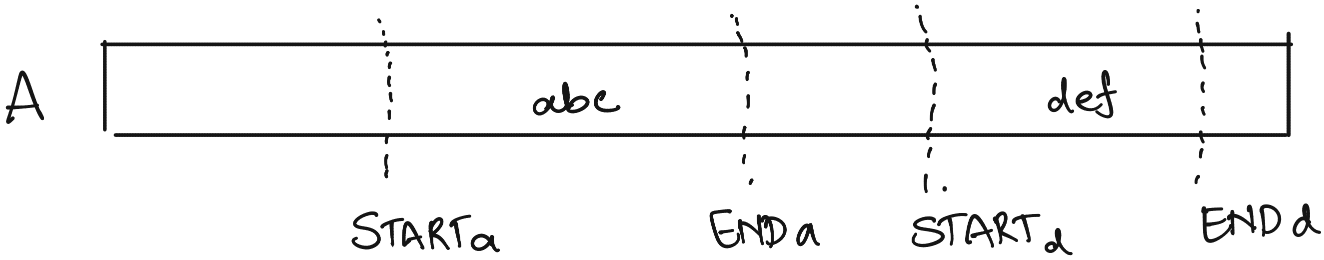


(2) $a = d$ & $b < e$

CORRECTNESS:

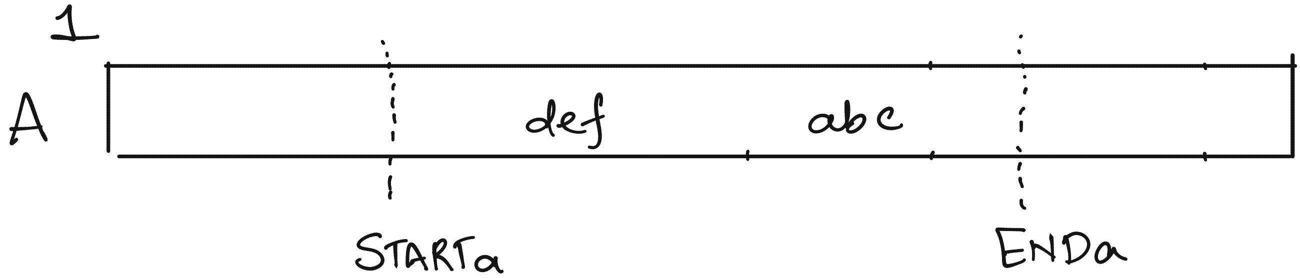
<u>LEMMA</u>: IF abc < def THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

<u>PROOF</u>: (1) If $a < d$



(2) $a = d$ & $b < e$

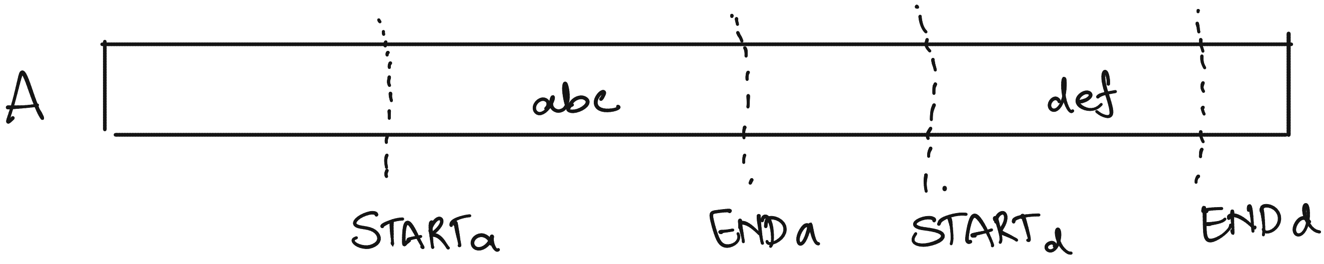
ITERATION 1



CORRECTNESS:

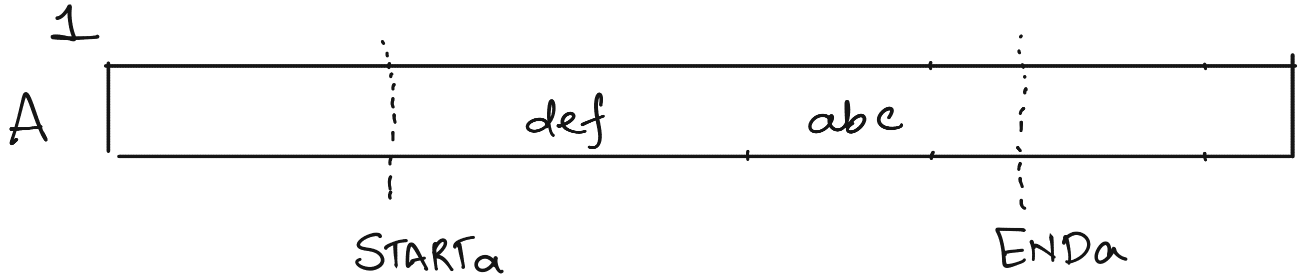
LEMMA : IF abc < def THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

PROOF: (1) If $a < d$

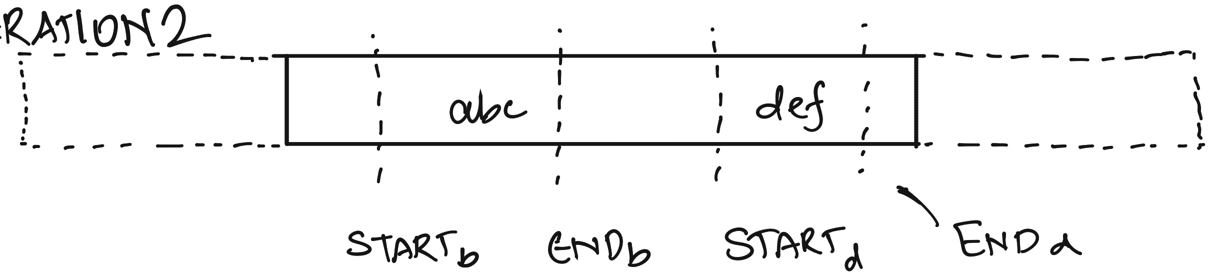


(2) $a = d$ & $b < e$

ITERATION 1



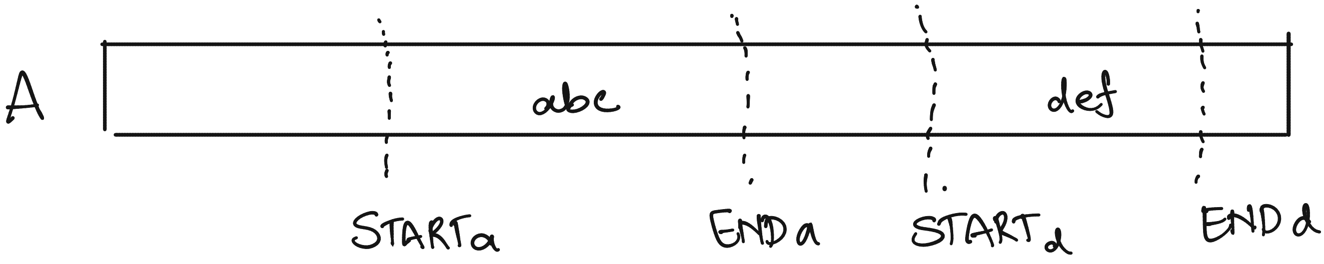
ITERATION 2



CORRECTNESS:

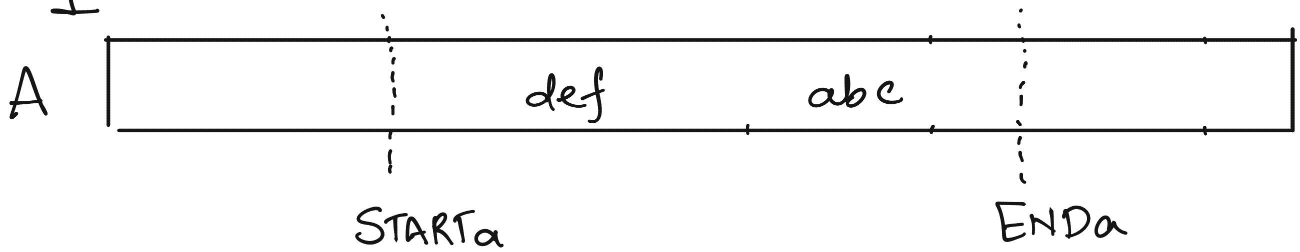
LEMMA: IF $abc < def$ THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY.

PROOF: (1) If $a < d$

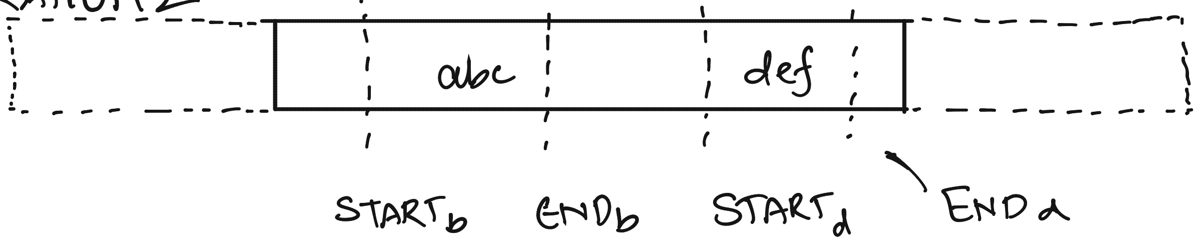


(2) $a = d$ & $b < e$

ITERATION 1



ITERATION 2



(3) $a = d$, $b = e$, $c < f$

SIMILAR ARGUMENT

RADIX SORT : MSD → LSD

LSD → MSD

321

932

561

325

323

961

001

962

RADIX SORT : MSD $\rightarrow$ LSD

LSD $\rightarrow$ MSD

321

932

561

325

323

961

001

962

SORT ON LEAST

SIGNIFICANT DIGIT

USING BUCKET SORT

RADIX SORT : MSD → LSD

LSD → MSD

321	321
932	561
561	961
325	001
323	932
961	962
001	323
962	325

SORT ON LEAST
SIGNIFICANT DIGIT
USING BUCKET SORT

RADIX SORT : MSD → LSD

LSD → MSD

321	321
932	561
561	961
325	001
323	932
961	962
001	323
962	325

RADIX SORT : MSD → LSD

LSD → MSD

321	321
932	561
561	961
325	001
323	932
961	962
001	323
962	325

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

RADIX SORT : MSD → LSD

LSD → MSD

321	321
932	561 ←
→ 561	961
325	001 ←
323	932
961	962
→ 001	323
962	325

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: IS BUCKET SORT STABLE ?

RADIX SORT : MSD → LSD

LSD → MSD

321	321
932	561
561	961
325	001
323	932
961	962
001	323
962	325

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: IS BUCKET SORT STABLE ?

1	2	3	4	5	6	7	8	9
321	932	323		325				
561	962							
961								
001								

RADIX SORT : MSD → LSD

LSD → MSD

321	321
932	561 ←
→ 561	961
325	001 ←
323	932
961	962
→ 001	323
962	325

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: IS BUCKET SORT STABLE?

A: YES

RADIX SORT : MSD → LSD

LSD → MSD

321	321	001
932	561	321
561	961	323
325	001	325
323	932	932
961	962	561
001	323	961
962	325	962

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: IS BUCKET SORT STABLE ?

A: YES

RADIX SORT : MSD → LSD

LSD → MSD

321	321	001	001
932	561	321	321
561	961	323	323
325	001	325	325
323	932	932	561
961	962	561	932
001	323	961	961
962	325	962	962

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: IS BUCKET SORT STABLE ?

A: YES

RADIX SORT : MSD → LSD

LSD → MSD

321	321	001	001
932	561	321	321
561	961	323	323
325	001	325	325
323	932	932	561
961	962	561	932
001	323	961	961
962	325	962	962

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: WHAT WOULD HAVE HAPPENED IF BUCKET SORT WAS NOT STABLE

RADIX SORT : MSD → LSD

LSD → MSD

321	321	001
932	561	321
561	961	323
325	001	325
323	932	932
961	962	561
001	323	962
962	325	961

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: WHAT WOULD HAVE HAPPENED IF BUCKET SORT WAS NOT STABLE

RADIX SORT : MSD → LSD

LSD → MSD

321	321	001	
932	561	321	
561	961	323	
325	001	325	
323	932	932	
961	962	561	
001	323	962	UNSTABLE PAIR
962	325	961	

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: WHAT WOULD HAVE HAPPENED IF BUCKET SORT WAS NOT STABLE

RADIX SORT : MSD → LSD

LSD → MSD

321	321	001		001
932	561	321		321
561	961	323		323
325	001	325		325
323	932	932		561
961	962	561		932
001	323	962	UNSTABLE	962
962	325	961	PAIR	961

THE RELATIVE ORDERING OF "SAME" NUMBER DOES NOT CHANGE

SUCH A SORTING ALGORITHM IS CALLED STABLE SORTING ALGORITHM.

Q: WHAT WOULD HAVE HAPPENED IF BUCKET SORT WAS NOT STABLE

RADIX SORT (A[1.. n])

{ FOR $i \leftarrow 1$ TO d

{ SORT THE ARRAY USING THE d^{th} DIGIT USING BUCKETSORT

}

}

RADIX SORT $(A[1..n])$

{ FOR $i \leftarrow 1$ TO d

$O(n)$ { SORT THE ARRAY USING THE d^{th} DIGIT USING BUCKETSORT }

}

RUNNING TIME: $O(nd)$

CORRECTNESS:

LEMMA: If $abc < def$, THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY

CORRECTNESS:

LEMMA: If $abc < def$, THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY

PROOF: (1) $a < d$

CORRECTNESS :

LEMMA : If $abc < def$, THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY

PROOF : (1) $a < d$

NO MATTER WHAT HAPPENS IN ITERATION 1 & 2, IN ITERATION 3 abc WILL COME BEFORE def.

CORRECTNESS:

LEMMA: If $abc < def$, THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY

PROOF: (1) $a < d$

NO MATTER WHAT HAPPENS IN ITERATION 1 & 2, IN ITERATION 3 abc WILL COME BEFORE def.

(2) $a = d$ & $b < e$

CORRECTNESS:

LEMMA: If $abc < def$, THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY

PROOF: (1) $a < d$

NO MATTER WHAT HAPPENS IN ITERATION 1 & 2, IN ITERATION 3 abc WILL COME BEFORE def.

(2) $a = d$ & $b < e$

IN THE SECOND ITERATION abc COMES BEFORE def.

IN THE THIRD ITERATION,

CORRECTNESS:

LEMMA: If $abc < def$, THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY

PROOF: (1) $a < d$

NO MATTER WHAT HAPPENS IN ITERATION 1 & 2, IN ITERATION 3 abc WILL COME BEFORE def.

(2) $a = d$ & $b < e$

IN THE SECOND ITERATION abc COMES BEFORE def. IN THE THIRD ITERATION, THE RELATIVE ORDERING OF abc AND def DOES NOT CHANGE (STABLE SORTING)

CORRECTNESS:

LEMMA: If $abc < def$, THEN abc LIES TO THE LEFT OF def IN THE FINAL ARRAY

PROOF: (1) $a < d$

NO MATTER WHAT HAPPENS IN ITERATION 1 & 2, IN ITERATION 3 abc WILL COME BEFORE def.

(2) $a = d$ & $b < e$

IN THE SECOND ITERATION abc COMES BEFORE def. IN THE THIRD ITERATION, THE RELATIVE ORDERING OF abc AND def DOES NOT CHANGE (STABLE SORTING)

(3) $a = d$, $b = e$ & $c < f$

SIMILAR TO ABOVE ARGUMENT