Homework-4

Name: Sai Rohit Kalyan Grandhan.

StudentID: 1002070724

EmailID: Sxg0724@mavs.uta.edu

8.2.1) Given array A = {6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2}

Sol) We have to prove the operation of Counting-Sort

→ In order to do Counting-Sort we have two have Extra two arrays

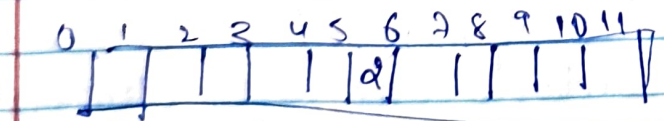
→ We Construct the array of Counts

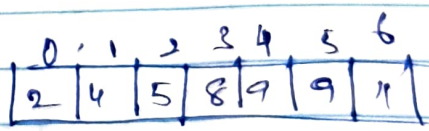
	0	1	2	3	4	5	6
C:	2	2	2	2	1	0	2

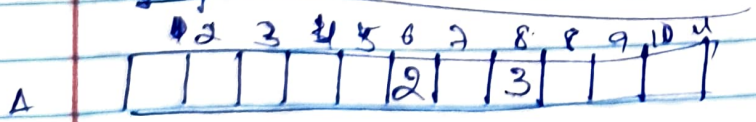
→ The No. of Elements before each

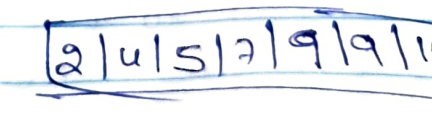
	0	1	2	3	4	5	6
	2	4	6	8	9	9	11

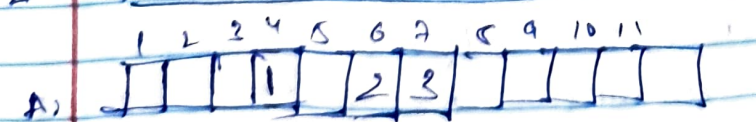
Counting Sort:

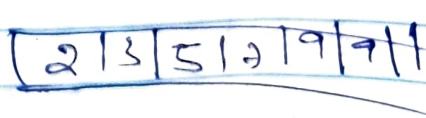


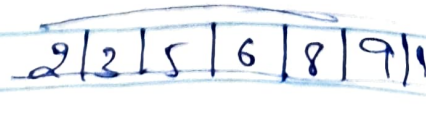


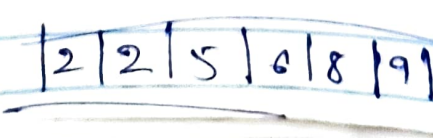


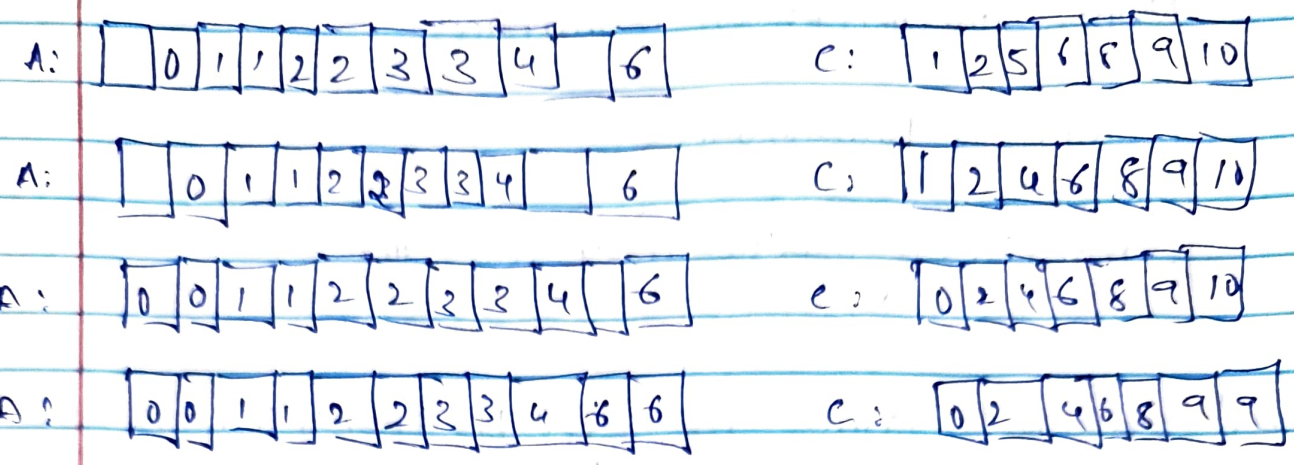












8.2.3) Given to change the code at line for j = 1 to A.length

Sol)

for j = 1 to A.length

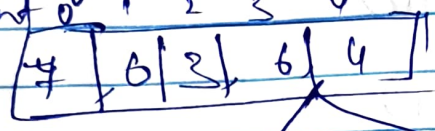
Sudo Code for Counting Sort

```
1. let C[0...k] be a new array
2. for i = 0 to k
3.         C[i] = 0
4. for j = 1 to A.length
5.         C[A[j]] = C[A[j]] + 1
6. // C[i] now contains the no. of elements equal
   to i
7. for i = 1 to k
8.         C[i] = C[i] + C[i-1]
9. // C[i] now contains the no. of elements
      less than or equal to i
10. for j = A.length downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

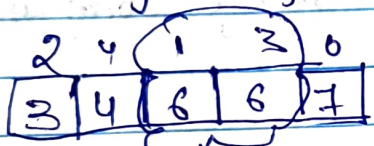
* When change line (10) from
for j = A.length downto 1 to
for j = 1 to A.length.

* What happens instead of getting first/last index Element first, we we get the Second highest index Element

Ex:

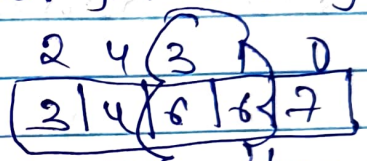


for j = A.length to down 1



* Here (6) form position (1) will come first

for j = 1 to A.length



* Here (6) from position (3) will come first

* So from Example we can say it will sort correctly, but they are not Stable.

* The Element will appear in Equal but order will be reverse sorted.

8.3-4) Given to show sort of (n) integers in
Sol) the range 0 to $n^3 - 1$ is O(n) times

* WKT Radix sort is $\left(\frac{b}{r}\right)(n + 2^r)$

* using radix formula,
* Given the given Numbers in range digits 1 to n So u = n.
* The No. of passes needed is = 3

$\Rightarrow n^3 - 1 = 100$

So $d = 2$

* So when run Radix Sort we get

$$O(dn + dk) = O(2n + 2n)$$

$$\Rightarrow O(4n)$$

$$\Rightarrow O(n)$$

* Hence proved for $(n^3 - 1)$ range Complexity is $O(n)$.

9.1.1) When you compare the elements in a tournament fashion - we split them into pairs, compare each pair and then proceed to compare the winners in the same fashion. We need to keep track of each "match" the potential winners have participated in.

We select a winner in $(n-1)$ match, At this point WLT second smallest element is one of $\log(n)$ element that lost to the smallest each of the them is smaller than the ones it has been compared to, prior to losing.

* In another $(\lg(n))$ comparisions we can find smallest element one of these.

9.3-1) Given the Select the input elements are
Sol) divided into group (5), if we divid the groups into (7)

* The changed algorithm when we divided in group of (7) is less than (or greater than) x

$$4\left(\tfrac{1}{2}\lfloor n/7 \rfloor - 2\right) \geq 2n/7 - 8$$

* When we divid it by (7) the partition will reduce the subproblems at most $(5n/7 + 8)$

$$T(n) = \begin{cases} O(1) & \text{if } n < n_0 \\ T(\lceil n/7 \rceil) + T(5n/7 + 8) + O(n) & \text{if } n \geq n_0 \end{cases}$$

* for the $T(n) \leq cn$ and bound the non-recursive term with an

$$\begin{aligned} T(n) &\leq c\lceil n/7 \rceil + c(5n/7 + 8) + an \\ &\leq c\lceil n/2 \rceil + c + 5nc/7 + 8c + an \\ &= 6nc/7 + 9c + an \\ &= cn(-cn/7 + 9c + an) \rightarrow (i) \\ &\leq cn \\ &= O(n) \end{aligned}$$

* So for $T(n) = 1 \Rightarrow$ we take last step (i) hold for ≤ 0

$$-cn/7 + 9c + an \leq 0$$

$$c(n/7 - 9) \geq an$$

$$c\left(\frac{n - 63}{7}\right) \geq an$$

$$c(n-63) \geq an$$

$$c \geq \frac{7an}{(n-63)}$$

Now:

Groups of (3)

$$2([1/2[n/3]-2) \geq n/3-4$$

$$T(n) \geq T[n/3] + T[2n/3+4) + O(n)$$

* We have to prove $T(n) = \omega(n)$

$$T(n) \geq c[n/3] + c(2n/3+2) + an$$
$$\geq cn/3 + 2cn/3 + 2c + an$$
$$= cn + 3c + an$$
$$\geq cn$$
$$= \omega(n)$$

∴ Hence proved.

9.38) Given two array's $X[1...n]$ and $Y[1...n]$

* Need to find median of all $(2n)$ Elements in array X and Y

Sol)

* If
* Let assume nums1 : [1, 3]
nums2 : [2, 4]

median : $(2+3)/2 = 2.5$

median: is the value separating the highest Element of the half of the data sample, a population, or a probability distribution, from other half

For Odd n, median (M): Value of $((n+1)/2)^{th}$ item.
For Even n, median (M): Value of $\{(n/2)^{th}$ item + $(n/2+1)$ item $\}/2$

* Here we have to Compare two algorithmic
① linear ② logarithmic

=> ① Counting while Comparing: find the middle Elements after merging the sorted array using the merge procedure of merge sort

② Comparing medians: Get the medians of two sorted arrays and Compare the medians and move accordingly until the two medians become equal! following divided.

=> ① Counting while Comparing:

Sudo code:

1. Create two pointers i and j pointing two nums and nums2 respectively and initialise them with 0
2. if nums1[i] < nums2[j] then increment i

Otherwise increment (j)
3. Repeat step(2) until i (or) j become it's Equal to size
4. Return median of the two Elements

Pseudo Code:

```
float getmedian (int nums1[], int nums2[] int size)
int i,j
int m1 = -1, m2 = -1
for (count = 0 to size) {
    if (i == size) {
        m1 = m2
        m2 = nums2[0]
        break }
Else-if (j == size) {
        m1 = m2
        m2 = nums1[0]
        break }
if (nums1[i] < nums2[j]) {
        m1 = m2
        m2 = nums1[i]
        i = i+1 }
Else {
        m1 = m2
        m2 = nums2[j]
        j = j+1 } }
return (m1+m2)/2
```

Comparing medians

* if $m_1 == m_2$ then return m_1
* if $m_1 > m_2$ then median will be present in either of sub-arrays nums1[0...m_1] and nums2[m_2 to 0]
* if $m_2 > m_1$ => nums2[0 to m_2]
* Return median = (max(array1[0], array2[0]) + min(array1[1], array2[1])) / 2

Pseudo Code:

```
float getmedian(int n1[], n2[], int Size){
    if (Size < 0) return -1
    if (Size == 1) return (n1[0]+n2[0])/2
    if (Size == 2) return (max(num1[0],n2[0])+
        min(n1[1], n2[1]))/2

int med1 = median(n1, Size)
int med2 = median(n2, Size)

    if (med1 == med2) return med1
    if (med1 < med2) {

        if (Size % 2 == 0) return
            getmedian(n1+Size/2-1, n2, Size-Size/2+1)
        Else
        return getmedian(n1+Size/2, n2, Size-Size/2)}
        Else
            if (Size % 2 == 0) {
```

return getmedian (n2+size/2-1, num1, size-size/2+1)
Else
return getmedian (n2+size/2, n1, size-size/2)

	Time Complexity	Space Complexity
Computing while Comparing	O(n)	O(1)
Comparing medians	O(log(n))	O(log(n))