TOPIC | DATE

P1.1 Give an efficient sorting algorithm for an array C[1,...,n] whose elements are taken from set {1,2,3,4.,5,6,7}

⇒ We are going to use counting sort because the element of our sorted array is taken from a set of size n (or 7 in our case).

1. First we are going to loop through each element in the array, and increment appropriate count for each of them.
2. then we loop through the count, added its previous value to get summation of count.
3. Finally we loop through items in traverse by use item key to index the count array, decrement that count array and use decremented value as array index to copy item to the sorted array.

Example:

imagine we have this unsorted array

| 7 | 6 | 4 | 4 | 2 | 1 | 3 | 5 | 4 | 1 | 2 | 3 |

↳ loop through and increment the count

1	2	3	4	5	6	7
2	2	2	3	1	1	1

↳ loop through the count to get summation of count

1	2	3	4	5	6	7
2	4	6	9	10	11	12

↳ Final step, loop through this in traverse and put it in sorted array

1	1	2	2	3	3	4	4	4	5	6	7

TOPIC | DATE

Pseudo code:

```
for (int i = 1 , i <= 7 ; i++) {
    while (count[j] > 0) {
        arr[i++] = j
        count[j] -- ; }
}
```

This sorted algorithm has O(n+k) or O(n).

TOPIC | DATE

P1.2 Give an efficient sorting algorithm for an array $D[1, \ldots, n]$ whose elements are distinct $D[i] \neq D[j]$ for every $i \neq j \in \{1, \ldots, n\}$ and are taken from set $\{1, 2, \ldots, 2n\}$

the same as problem 1.1, since we know the specific range of our elements in the array, we are going to use counting sort as it provide the complexity of $O(n+k)$.

+ first we are going to define arr[n] with size n with its elements taken from set $\{1, 2, \ldots, 2n\}$
+ then we ~~are~~ define array as count[2n] by looping through each element to count through it occurance
+ finally we loop through that counted array and rearranging it in a new array

Pseudo code

```
int i=0;
for(i=0; i<(n); i++) {        // n element of the array (like len of array)
    while (count[i] >0 ) {
        arr[i++] = j;
        count[j] = count--;
    }
}
```

This take $O(n+k)$ times and space complexity.

if $k = O(n)$

$\Rightarrow$ take $\Theta(n)$ time and space

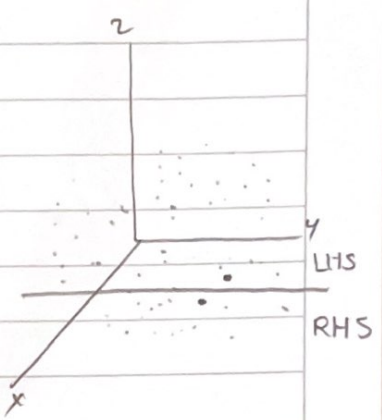
TOPIC | DATE

P2.1 Since we are not using comparison sort for problem 1, the lower bound sorting of $\Omega(n \log n)$ is wrong. The counting sort lower bound is $\Omega(n+k)$ for time complexity.

The reason that $\Omega(n \log n)$ is not applicable for counting sort is because counting sort uses key value as indexes into an array.

TOPIC | DATE

P3. To get O(nlogn) time complexity for this closest pair problem, we are going to implement divide and conquer algorithm.
First we are going to divide our data into left hand side (LHS) and right hand side (RHS) recursively. Then when the elements are small enough, we can use naive approach to find the closest pair.
However, the closest pair could have 1 point in LHS and another point in RHS.
To solve this, we create a new set call "strip" that include the elements that are closed to the divided line. We do not have to use naive method for every element inside the strip because of sparsity.
The elements on the left and right side of the strip cannot have the distance half of the strip width.



To solve this, after we get left and right set, we divide Z into column. Then we put each element in the strip, into column. If element can be in the left column, right column and main column. Hence, we only have to check 3 columns. The point in the strip is created in ascending order of Y. First phase is we will put values from strip in correct column. Because our Y is already sorted in ascending order; hence, the value in the column is also sorted as well. This will save the position in left, right and main. The second phase, we will calculate the shortest distance. It pick first element of the strip then check it upper position of the element using column.

TOPIC | DATE

P3 Pseudo code

- divide the set in $\frac{n}{2}$
- use recursive function to search for min distance in LHS
- use recursive function to search for min distance in RHS
- find strip using min of distance in LHS and distance in RHS

  strip $\delta = \min(\delta_{LHS}, \delta_{RHS})$

- set minimum distance to strip distance (size is number of elements in strip)

if size $< 2 \rightarrow$ return strip distance

if size $< 4 \rightarrow$ return naive method for finding distance

if size > 3 (n size): create column in z dimension.

find the smallest z value

find the largest z value

find the range between z value and column count

$\Rightarrow$ column count $= \frac{\text{largest z} - \text{smallest z}}{\text{strip distance}} - 1$

for each point in the strip, find main column of the ball and save column id and column index

for each point in the strip, find if there are any other smallest distance in the strip by picking the point one by one in an ascending other of y axis

for left, middle, right column $\rightarrow$ select ball

if column out of bound $\rightarrow$ skip

else:

loop through increasing y

if y distance $>$ strip distance $\rightarrow$ break

else min distance = new distance