Swati Gupta
20/0846.

Q1. Algorithm for sorting kth node from end and front in a doubly linked list.

:: swaps(int i) → //taking i as k

Taking *temp as header and *temp2 as trailer and then traversing through the list i-1 times to point a ~~node~~ to the kth node from front and end.

from front

```
while(counter < i-1)
{
   temp = temp→next;
   counter++;
}
   temp = temp→next;
```

from the end

```
while (count < i-1)
{
   temp2 = temp2→prev;
   count++;
}
   temp2 = temp2→prev;
```

while we traverse i-1 times we point to i-1 th node ∵ the index starts from 0. then we take one more ~~traversion~~ like pointing which goes to the nextor the previous of that particular pointer.

like:- temp = temp→next; //now temp points to the kth node from the front
and temp2 then points to the kth node from the back.

We have a boundary condition of header and trailer

Taking the boundary condition.

```
if (temp == header || temp2 == header).
{
    if(temp == header).
```

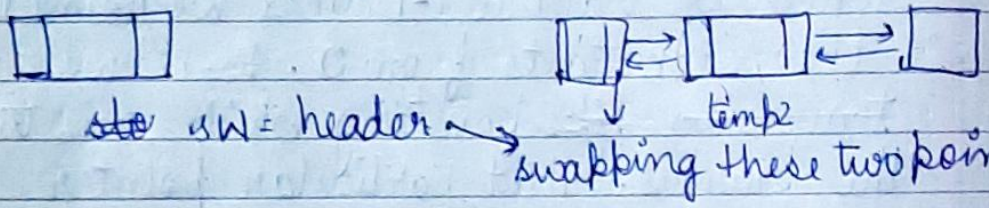
// Now taking the condition for temp to point to header
We have already taken a pointer which is address equal to NULL.

```
Swapping (header, temp2 -> prev -> next).
```

// We are storing header into the NULL pointer

```
sw = header;
```

and now we will take the predecessor next of the second node and store in header.
and also take the original header stored in sw and now store it in the predecessor next of the second node.



Now swapping (temp -> prev, temp2 -> prev).
and swapping (temp -> next, temp2 -> next).

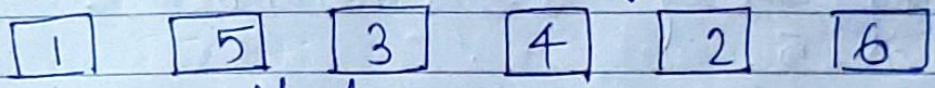
Basically, we have broken the existing links and swapped the nodes creating new links.

Taking value of k as 3.

Now we are breaking links by first storing address of 2 in the prev of pointer of 6..
And prev of 6 into address of 2.

Doing the same with next of 2 and next of 5 and prev of 2 and prev of 5.

~~We are have pointing the next.~~



output.

Q2. Recursive function to sum elements of an array:-

```
int sum(int a[], int n).
{
    if(n==0)    // boundary condition.
        return 0;
    return a[n-1] + sum(a, n-1);
}.
```

n → size of the array.
a → array.

Taking array a = {1, 2, 3, 4, 5}.
n = 5.

sum(a)	n=1 → a[1-1] + 0 → a[0] + 0 = 1
sum(1)	n=2 → a[2-1] + 1 = 3
sum(2)	n=3 → a[3-1] + 3 = a[2] + 3 = 6
sum(3)	n=4 → a[4-1] + 6 = a[3] + 6 = 10
sum(4)	n=5 → a[5-1] + 10 = a[4] + 10 = 15

now sum = 15.

Q3. Array of integers
12, 14, 9, 18, 120, 30, 40, 35, 60.

Insertion sort will be best suited because the array is almost sorted and only. 9, 120 and 35 are in the wrong position.

Algorithm:-

```
for(int i = 1; j; i<n; i++)
  {
      T tmp = data[i];
    for(j = i; j>0, && tmp < data[j-1]; j--)
    {
       data[j] = data[j-1];
    }
    data[j] = tmp;
  }
```

Also, In average case bubble sort makes approximately twice as many comparisons and same number of moves as insertion sort, as many comparisons as selection sort and n times more than selectionsort.

Insertion sort is fast among these two sorting algorithms. ~~compo~~

Comparison of runtime of random numbers.

Insertion sort → .438
Selection sort → .818
Bubble sort → 17.700.

array = {12, 14, 9, 18, 120, 30, 40, 35, 60}.

i=1
j=1

12	14	9	18	120	30	40	35	60

tmp = 14. 14 is not less than 12
no swapping.

i=2
j=2.

12	14	9	18	120	30	40	35	60

tmp = 9 9 is less than 14
swapping.

j=j-1

12	14	14	18	120	30	40	35	60

tmp = 9 9 is less than 12.
swapping.

12	12	14	18	120	30	40	35	60.

no swapping.

9	12	14	18	120	30	40	35	60

i=3
j=3.

9	12	14	18	120	30	40	35	60

tmp = 18. 18 is not less than 14.
no swapping.

i=4
j=4

9	12	14	18	120	30	40	35	60

120 is not less than 18

no swapping.

i=5
j=5

9	12	14	18	120	30	40	35	60

30 is less than 120

swapping.

j--
j=4

9	12	14	18	120	120	30	40	35	60

30 is not less than 18

no swapping.

i=6
j=6

9	12	14	18	30	120	40	35	60

40 is less than 120.

swapping.

j--
j=5

9	12	14	18	30	120	120	35	60

40 is not less than 30.

no swapping.

i=7
j=7

9	12	14	18	30	40	120	35	60

9	12	14	18	30	40	120	120	60

9	12	14	18	30	40	35	120	60

9	12	14	18	30	40	35	120	120

9	12	14	18	30	40	35	60

9	12	14	18	30	40	40	120	60

9	12	14	18	30	35	40	120	60

9	12	14	18	30	35	40	120	120

9	12	14	18	30	35	40	60	120

Q4 (i) (4,5,8,1) (1,5,8,4) (1,4,8,5)
(1,4,5,8).

→ Selection sort is used here because in intermediate configuration, smallest element is selected and exchanged with the element in 1st position, then the smallest value among the remaining elements is found and put in 2nd position.

Algo:

```
for( int i=0 ; j, least; i<n-1; i++)
{
  for( j=i+1, least=i ; j<n; j++)
  {
    if ( data[j] <data[least])
      least = j;
  }
  swap( data[least], data[i]);
```

4	5	8	1

i = 0 j = 1. least = 0.

false (5<4) least = 1. j++. ⇒ j = 2
false (8<5) least = 1. j++ ⇒ j = 3.
true (1<5) least = 3 j++ ⇒ j = 4.

(j < n) so out from inner loop.
then swap(data[3], data[0])

1	5	8	4.

i = 1 j = 2. least = 1

false. (8<5) least = 1 j++; j = 3.
true (4<5) least = 3 j++, j = 4.

(j < n) out from inner loop.
then swap(data[3], data[1])

1	4	8	5.

i = 2 j = 3 least = 2.

true. (5<8). least = 3.
swap(data[3], data[2]).

1	4	5	8

i=3 false condition i<n-1 ; i<2 .

. Sorted array [1|4|5|8.]

(ii) (4,5,8,1) (4,5,1,8) (4,1,5,8) (1,4,5,8)

- Bubble sort is used because in bubble sort, the array is scanned from top to bottom up and two adjacent elements are interchanged if they found to be out of order with respect to the each other. first data[n-1] and data[n-2] are compared and swapped if they are out of order, next data[n-2] and data[n-3] are compared and their order is changed if necessary and so on upto data[1] and data[0].

algorithm:

```
for(int i=0; i<n-1; i++)
  {
    for(int j=n-1; j>i; --j)
      {
        if(data[j] < data[j-1])
          swap(data[j], data[j-1])
      }
  }
```

Array:- {4, 5, 8, 1}

i=0 ; j=3. (1<8) true swap [data[3] data[2]

4	5	1	8

i = 0 j = 2 (1<5) true swap(data[2], data[i]

4	1	5	8

i = 0 j = 1 (1<4) true swap(data[1], data[

1	4	5	8