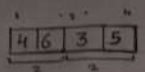
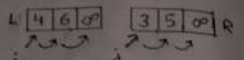
Time Complexity Analysis :

Here, we get recurrence relation as:



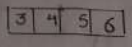
$$\therefore f(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(\frac{n}{2})+n & \text{otherwise} \end{cases}$$



$\Rightarrow T(n) = 2T(n/2) + n$ — ①

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}$ — ②

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n}{4}$ — ③



$\therefore T(n) = 2[2T(\frac{n}{4}) + \frac{n}{2}] + n$

$= 4T(\frac{n}{4}) + 2n$

$= 2^2 T(\frac{n}{2^2}) + 2n$

$\therefore T(n) = 2^3[2T(n/8) + \frac{n}{4}] + 2n$

$= 2^3 T(n/2^3) + 3n$

$\vdots$

$= 2^4 T(\frac{n}{2^4}) + 4n$

$= 2^5 T(\frac{n}{2^5}) + 5n$

$\vdots$

k times

$= 2^k T(\frac{n}{2^k}) + kn$

$= 2^k T(1) + kn$

$= 2^k \cdot 1 + kn$

$= n + n\log n$

$O(n \log n)$

$$\left[\begin{array}{l} \because \frac{n}{2^k} = 1 \\ n = 2^k \\ \text{or, } \log n = \log 2^k \\ \therefore \log n = k \end{array}\right]$$

The time complexity of the provided merge sort code is $O(n \log n)$ in all cases - worst case, average case, and best case.

Experiment : 3

Title : Merge Sort

Aim : To implement and analyze merge sort algorithm

Algorithm :

Step 1 : Start

Step 2 : Read the size of the array (n) and its element from the user.

Step 3 : Declare and populate an array with user-input elements.

Step 4 : Apply the merge sort algorithm to arrange the elements in ascending order.

Step 5 : Print the initially entered array.

Step 6 : Print the array after sorting using merge sort.

Step 7 : Stop

Program - Implementation

```
#include <stdio.h>
#include <conio.h>
void printArray(int *A, int n){
    for(int i=0; i<n; i++){
        printf("%d", A[i]);
    }
    printf("\n");
}
void merge(int A[n], int mid, int low, int high){
    int i, j, k, B[100];
    i = low;
    j = mid + 1;
    k = low;
    while(i<=mid && j<=high){
        if(A[i]<A[j]){
            B[k]=A[i];
            i++;
            k++;
        }
```

Dry run with sample input and output :

Sample Input :

Enter the size of the array : 4
Enter 4 elements :
4
6
3
5

Sample Output :

Original Array : 4 6 3 5
Sorted Array : 3 4 5 6

```
        else {
            B[k] = A[j];
            j++;
        }
        k++;
    }
    while (i <= mid) {
        B[k] = A[i];
        k++;
        i++;
    }
    while (j <= high) {
        B[k] = A[j];
        k++;
        j++;
    }
    for (int i=low; i<=high; i++){
        A[i] = B[i];
    }
}
void mergeSort (int A[], int low, int high) {
    int mid;
    if (low < high) {
        mid = (low+high) / 2;
        mergeSort (A, low, mid);
        mergeSort (A, mid+1, high);
        merge (A, mid, low, high);
    }
}

int main() {
    clrscr();
    int n;
    printf ("Enter the size of the array : ");
    scanf ("%d", &n);
    int A[n];
    printf ("Enter %d elements : \n", n);
    for (int i = 0; i<n ; i++) {
        scanf ("%d", &A[i]);
    }
```

```
printf ("Original Array :");
printArray (A,n);
mergeSort (A, 0, n-1);
printf ("Sorted Array :");
printArray (A,n);
getch();
return 0;
}
```

Result : Merge Sort algorithm was implemented and analyzed successfully.