MERGE SORT

Aim- Write a C program to implement Merge Sort on array of structures

Problem Statement – Given a array of structures of items implement merge sort to sort the array of structures based on id number in assending order

INPUT - The number of structures in the array = 7

Id num – 12,15,17,19,9,11,42

Item name – Candy ,Chocolate , Chips, Ice-cream , Biscuits ,Book, Pen

Price – 10,40,5,40,15,80,20

OUTPUT – Display the structures in sorted manner

ALGORITHM –

i] Algorithm MergeSort (low,high)

```
// Given a global array arr[low:high} and a global temporary array b and 0<=low<=high
{ if (low<=high ) then {
 mid:= floor ((low+high)/2);
MergeSort (low,mid) ;
MergeSort(mid+1,high);
Merge (low,mid,high) ; } }
```

ii] Algorithm Merge (low,mid,high)

```
//Given two global arrays a,b and 0<=low<=mid<=high
{
i:=low ;
j:= mid +1 ;
k:= low ;
while ((i<=mid) and (j<=high)) do {
```

```
if ( arr[i] <= arr [j] ) then {
b[k] := arr[i] ;
i:= i+1 ;
} else {
e[k] := arr[j] ;
j := j+1 ;
}
X:= x+1 ; }
While (i<=mid) do {
b[k] : = arr[i] ;
i:= i+1 ; k:=k+1 ;}
while (j<=high) do {
b[k] := arr[j] ;
j:=j+1 ; k:=k+1 ;}}
```

Space and time complexity :

I. Algorithm MergeSort

Time Complexity:

i) **Best Case:**

- **O(n log n)**
- Even if the array is already sorted, the algorithm still recursively divides the array and merges it, leading to a time complexity of O(n log n).

ii) **Worst Case:**

- **O(n log n)**
- The algorithm always performs the same number of comparisons and divisions regardless of the input order.

iii) **Average Case:**

- **O(n log n)**
- On average, MergeSort divides the array and merges the sorted parts in logarithmic time for every level, with n operations at each level.

Space Complexity:

i) **Best Case:**

- **O(n)**
- The temporary array b requires linear additional space to store merged elements.

ii) **Worst Case:**

- **O(n)**
- Even in the worst case, the temporary array b is of size n, requiring linear additional space.

iii) **Average Case:**

- **O(n)**
- On average, the same temporary array is used, resulting in linear space usage.

II. Algorithm Merge

Time Complexity:

i) **Best Case:**

- **O(n)**
- Merging two sorted subarrays of total size n requires linear time in all cases.

ii) **Worst Case:**

- **O(n)**
- The merge process is always linear, irrespective of the input.

iii) **Average Case:**

- **O(n)**
- On average, merging two arrays of total size n takes linear time.

Space Complexity:

i) **Best Case:**

- **O(n)**
- A temporary array b of size n is used for merging.

ii) **Worst Case:**

- **O(n)**
- The same temporary array b is required regardless of the case.

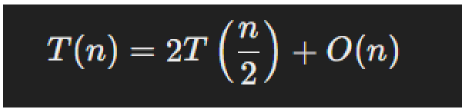
iii) **Average Case:**

- **O(n)**
- On average, merging uses linear additional space for the temporary array.

Recurance Equation :

I. Algorithm MergeSort

The recurrence equation for MergeSort is:



$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

II. Algorithm Merge

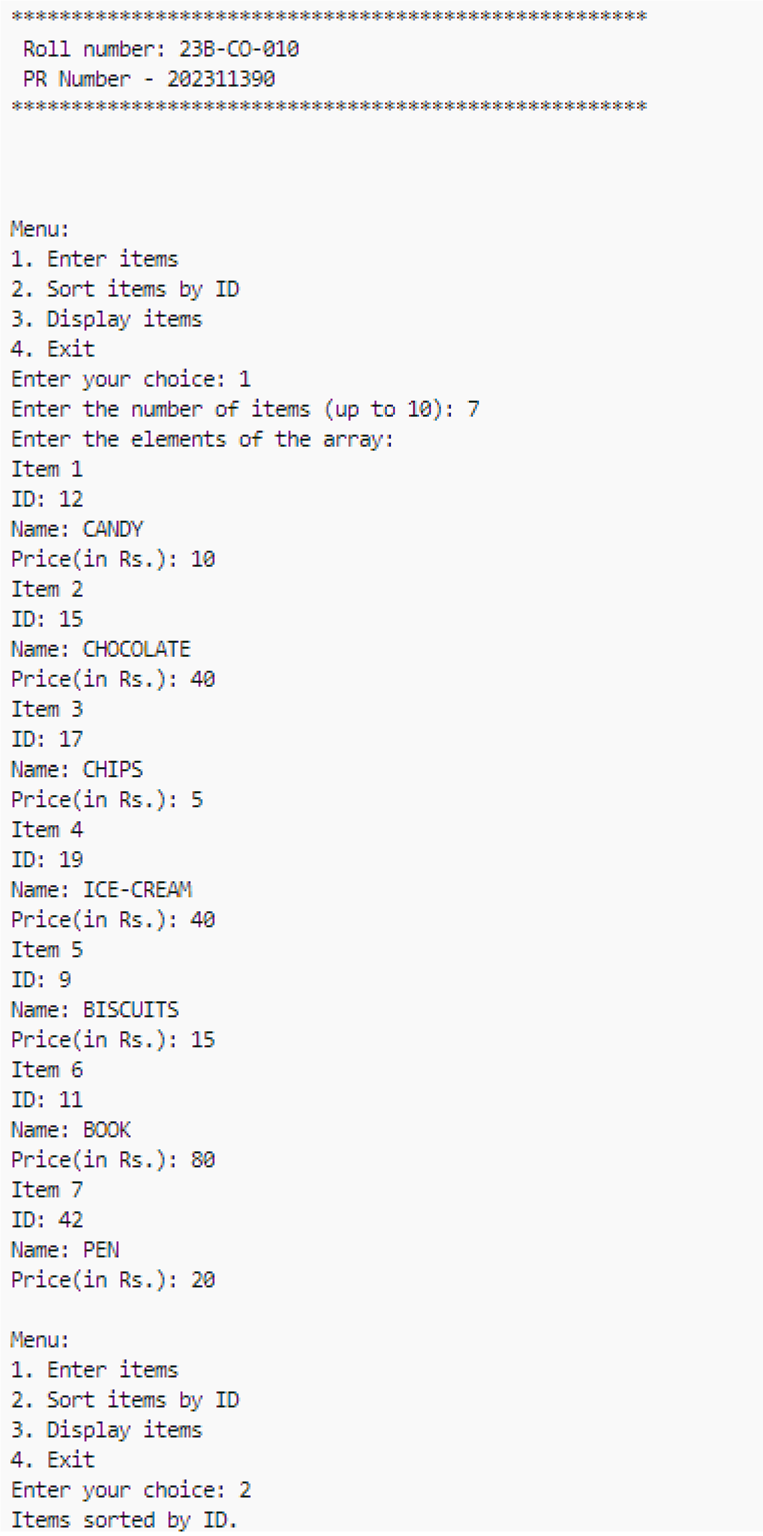
No recurrence equation exists for the Merge algorithm itself since it is not recursive. Instead, it is part of MergeSort and is a linear process.

PROGRAM –

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#define MAX 10
int n=0;

typedef struct {
    int id;
    char name[50];
    float price;
} Item;

Item a[MAX];

void merge(int min, int max) {
    int mid = (min + max) / 2;
    int i = min;
    int j = mid + 1;
    int k = 0;
    Item temp[MAX];

    while (i <= mid && j <= max) {
        if (a[i].id < a[j].id) {
            temp[k] = a[i];
            k++;
            i++;
        } else {
            temp[k] = a[j];
            k++;
            j++;
        }
    }

    while (i <= mid) {
        temp[k] = a[i];
        k++;
        i++;
    }

    while (j <= max) {
        temp[k] = a[j];
        k++;
        j++;
    }

    for (i = min, k = 0; i <= max; i++, k++) {
        a[i] = temp[k];
    }
}

void mergesort(int min, int max) {

    int mid;
    if (min < max) {
        mid = (min + max) / 2;
        mergesort(min, mid);
        mergesort(mid + 1, max);
        merge(min, max);
    }

}

void displayItems() {
    int i;
    printf("\nItems:\n");
    for (i = 0; i < n; i++) {
        printf("Item %d\n", i + 1);

        printf("ID: %d\n", a[i].id);

        printf("Name: %s\n", a[i].name);

        printf("Price: %.2f\n", a[i].price);

        printf("\n");

    }

}


int main() {



      printf
("*******************************************
*********");

    printf ("\n Roll number: 23B-CO-010\n");

    printf (" PR Number - 202311390\n");

    printf("**************************************
***************\n\n\n");

     clock_t start, end;

    double cpu_time_used;

    int choice;

    int i;

    do {

        printf("\nMenu:\n");

        printf("1. Enter items\n");

        printf("2. Sort items by ID\n");

        printf("3. Display items\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter the number of items (up to %d): ",
MAX);

                  scanf("%d", &n);

                if (n > MAX) {

                     printf("You can only enter up to %d
items.\n", MAX);
```

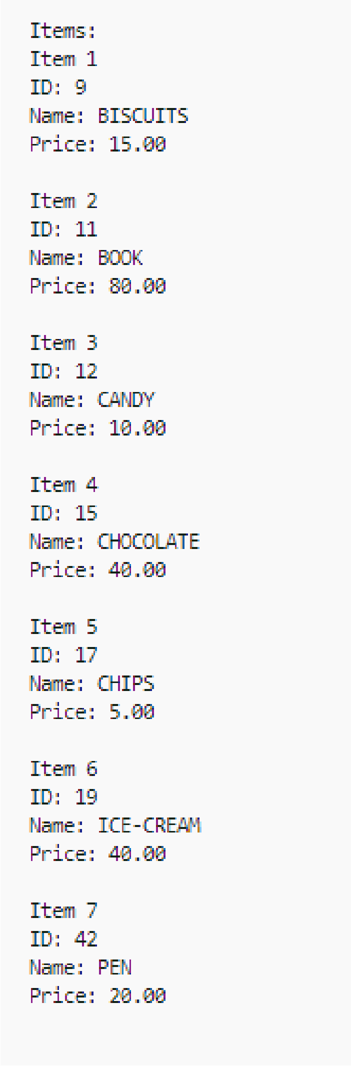
```
                 break;

               }

               printf("Enter the elements of the array:\n");

                for (i = 0; i < n; i++) {

                     printf("Item %d\n", i + 1);

                      printf("ID: ");

                      scanf("%d", &a[i].id);

               printf("Name: ");

                scanf("%s", a[i].name);

               printf("Price(in Rs.): ");

               scanf("%f", &a[i].price);

          }

              break;

          case 2:

          start = clock();

              mergesort(0, n-1);

              printf("Items sorted by ID.\n");

               end = clock();

    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;

    printf("Time taken by Merge Sort: %f seconds\n",
cpu_time_used);

              break;

          case 3:

              displayItems();

              break;

          case 4:

              printf("Exiting...\n");

              break;

          default:

              printf("Invalid choice. Please try again.\n");

        }

    } while (choice != 4);



    return 0; }

}
```

INPUT –



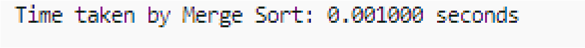
```
*****************************************************
 Roll number: 23B-CO-010
 PR Number - 202311390
*****************************************************



Menu:
1. Enter items
2. Sort items by ID
3. Display items
4. Exit
Enter your choice: 1
Enter the number of items (up to 10): 7
Enter the elements of the array:
Item 1
ID: 12
Name: CANDY
Price(in Rs.): 10
Item 2
ID: 15
Name: CHOCOLATE
Price(in Rs.): 40
Item 3
ID: 17
Name: CHIPS
Price(in Rs.): 5
Item 4
ID: 19
Name: ICE-CREAM
Price(in Rs.): 40
Item 5
ID: 9
Name: BISCUITS
Price(in Rs.): 15
Item 6
ID: 11
Name: BOOK
Price(in Rs.): 80
Item 7
ID: 42
Name: PEN
Price(in Rs.): 20

Menu:
1. Enter items
2. Sort items by ID
3. Display items
4. Exit
Enter your choice: 2
Items sorted by ID.
```

OUTPUT –



```
Items:
Item 1
ID: 9
Name: BISCUITS
Price: 15.00

Item 2
ID: 11
Name: BOOK
Price: 80.00

Item 3
ID: 12
Name: CANDY
Price: 10.00

Item 4
ID: 15
Name: CHOCOLATE
Price: 40.00

Item 5
ID: 17
Name: CHIPS
Price: 5.00

Item 6
ID: 19
Name: ICE-CREAM
Price: 40.00

Item 7
ID: 42
Name: PEN
Price: 20.00
```

TIME TAKEN –



```
Time taken by Merge Sort: 0.001000 seconds
```

CONCLUSION – Array of structures was successfully sorted with any errors using merge sort algorithm