**NAME: KUMUD MOHAN ZIMAL**

**BRANCH: SE COMPS-3**

**ROLL NO:65**

| **Experiment No. 3** |
| --- |
| **To implement Merge Sort** |
| Date of Performance: |
| Date of Submission: |

**Experiment No. 3**

**Title:** Merge Sort

**Aim:** To study, implement and Analyze Merge Sort Algorithm

**Objective:** To introduce the methods of designing and analyzing algorithms

**Theory:**

The merge sort algorithm closely follows the divide-and-conquer paradigm. Intuitively, it operates as follows:

* Divide: Divide the n-element sequence to be sorted into two sub sequences of n=2 elements each.
* Conquer: Sort the two sub sequences recursively using merge sort.
* Combine: Merge the two sorted sub sequences to produce the sorted answer.

During the Merge sort process the object in the collection are divided into two collections. To split a collection, Merge sort will take the middle of the collection and split the collection into its left and its right part. The resulting collections are again recursively sorted via the Merge sort algorithm.

Once the sorting process of the two collections is finished, the result of the two collections is combined. To combine both collections Merge sort start at each collection at the beginning. It pick the object which is smaller and inserts this object into the new collection. For this collection it now selects the next elements and selects the smaller element from both collection.

Once all elements from both collections have been inserted in the new collection, Merge sort has successfully sorted the collection. To avoid the creation of too many collections, typically one new collection is created and the left and right side are treated as different collections.

**Example:** Sort the sequence <33,22,44,0,99,88,11> using Merge Sort

**Algorithm and Complexity:**



Recurrence Relation for Merger Sort:

T(n) = 1 for n=1

T(n) = 2T(n/2) + n  for n>1… (1)

**Solve by Substitution method:**

Solving original recurrence for n/2,

T(n/2) = 2T(n/4) + n/2

Substituting this in equation (1),

T(n) = 2[ 2T(n/4) + n/2 ] + n

= 22 T(n/22) + 2n .

T(n) = 2k T(n/2k) + k.n … (2)

Let us consider that k grows up to log2n,

Let n/2k =1

n = 2k

k = log2n

n = 2k

Substitute these values in equation (2)

T(n) = nT(n/n) + log2n . n

T(n) = O(n.log2n)

**Solve using recursive tree Method:**

**Code:**

#include <stdio.h>

#include <conio.h>

void merge(int arr[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int \*L = (int \*)malloc(n1 \* sizeof(int));

int \*R = (int \*)malloc(n2 \* sizeof(int));

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

free(L);

free(R);

}

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size) {

int i ;

for (i= 0; i < size; i++)

printf("%d ", A[i]);

printf("\n");

}

int main() {

int arr[] = {76,45,32,11,9};

int arr\_size = sizeof(arr) / sizeof(arr[0]);

clrscr();

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

getch();

return 0;

}

**Output:**

**Conclusion:**

The "constant expression required" error in C typically arises when non-constant expressions are used to define array sizes. To resolve this, you can use dynamic memory allocation (malloc) or define array sizes using preprocessor directives (#define). Dynamic memory allocation allows arrays to be sized at runtime, avoiding the compile-time constraints of constant expressions. However, it's essential to remember to free dynamically allocated memory using free() to prevent memory leaks.