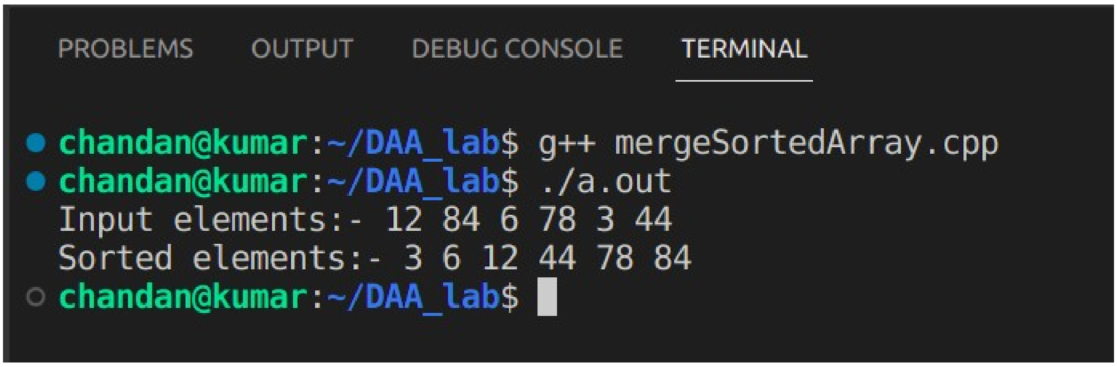
Practical No:- 02

Implement merge sort algorithm and demonstrate divide and conquer technique.

Code->

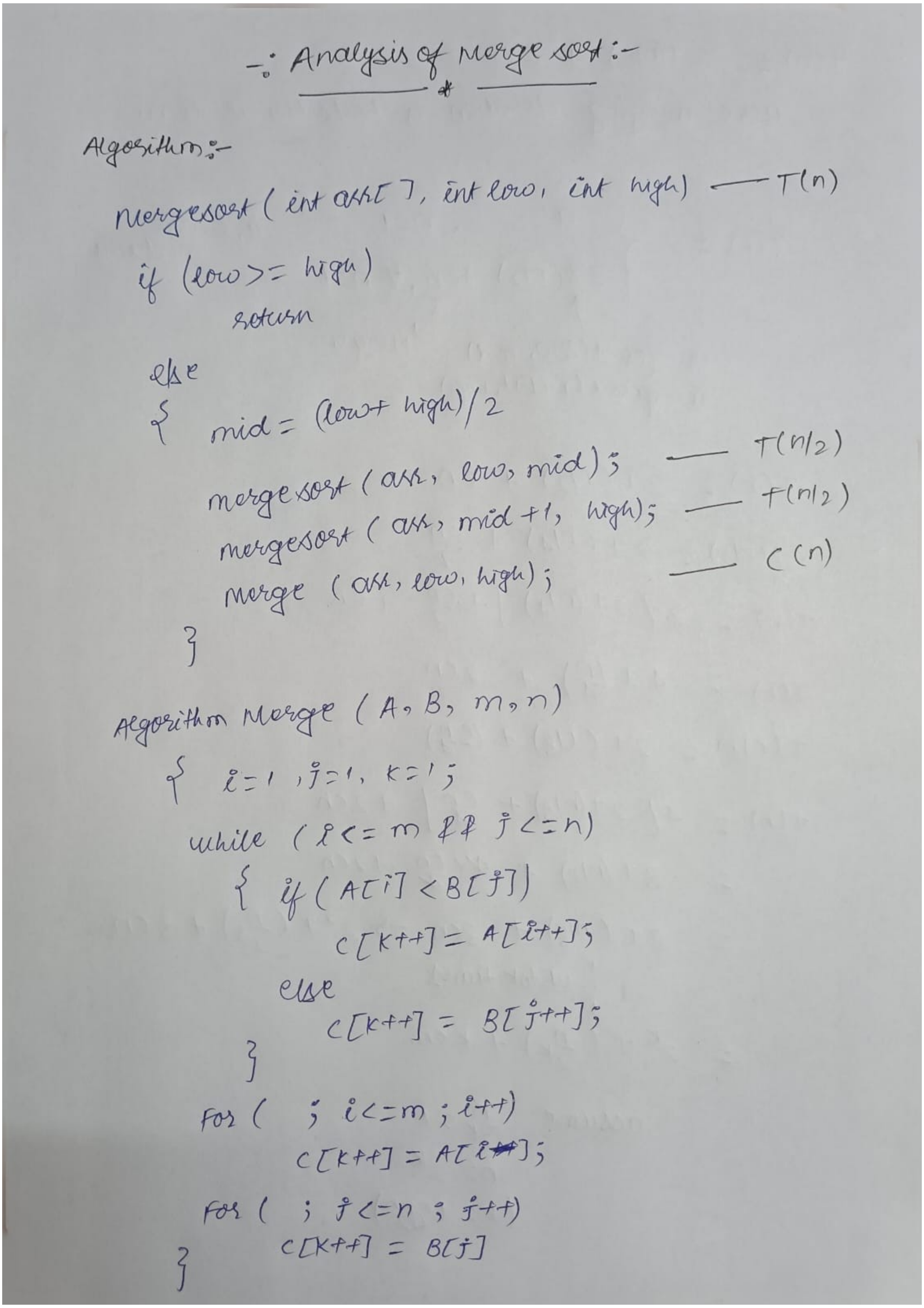
```
#include <bits/stdc++.h>
using namespace std;
// ---MERGING OF ARRAY---
void merge(int *arr, int low, int high)
{

int mid = (low + high) / 2;
int len1 = mid - low + 1;
int len2 = high - mid;
int *first = new int[len1];
int *second = new int[len2];
int mainArrayIndex = low;
for (int i = 0; i < len1; i++)
{
first[i] = arr[mainArrayIndex++];
}
mainArrayIndex = mid + 1;
for (int i = 0; i < len2; i++)
{
second[i] = arr[mainArrayIndex++];
}

int index1 = 0;
int index2 = 0;
mainArrayIndex = low;

while (index1 < len1 && index2 < len2)
{
if (first[index1] < second[index2])
{
arr[mainArrayIndex] = first[index1];
mainArrayIndex++;
index1++;
}
else
{
arr[mainArrayIndex] = second[index2];
mainArrayIndex++;
index2++;
}
}
while (index1 < len1)
{
arr[mainArrayIndex] = first[index1];
mainArrayIndex++;
index1++;
}
while (index2 < len2)
{
arr[mainArrayIndex] = second[index2];
mainArrayIndex++;
index2++;
}
delete[] first;
delete[] second;
}
// --- MERGE SORT---
void merge_sort(int *arr, int low, int high)
{

if (low >= high)
{
return;
}

int mid = (low + high) / 2;
merge_sort(arr, low, mid);
merge_sort(arr, mid + 1, high);
merge(arr, low, high);
}
int main()
{
int arr[] = {12, 84, 6, 78, 3, 44};
int size = sizeof(arr) / sizeof(arr[0]);

merge_sort(arr, 0, size - 1);

for (int i = 0; i < size; i++)
{
cout << arr[i] << " ";
}
cout << endl;

return 0;
}
```

Output:-



```
● chandan@kumar:~/DAA_lab$ g++ mergeSortedArray.cpp
● chandan@kumar:~/DAA_lab$ ./a.out
  Input elements:- 12 84 6 78 3 44
  Sorted elements:- 3 6 12 44 78 84
○ chandan@kumar:~/DAA_lab$ 
```

Analysis of algorithm:-



-: Analysis of Merge sort :-

Algorithm:-

```
mergesort ( int arr[ ], int low, int high)   —— T(n)
  if (low >= high)
      return
  else
  {  mid = (low+ high)/2
     mergesort ( arr, low, mid);            —— T(n/2)
     mergesort ( arr, mid +1, high);        —— T(n/2)
     merge ( arr, low, high);               —— C(n)
  }
```

```
Algorithm Merge ( A, B, m, n)
{  i=1 , j=1, k=1;
   while ( i<= m && j <=n)
     {  if ( A[i] < B[j])
             C[k++] = A[i++];
        else
             C[k++] = B[j++];
     }
   For (  ; i<=m ; i++)
          C[k++] = A[i~~++~~];
   For (  ; j<=n ; j++)
          C[K++] = B[j]
}
```

$T(n) = 2T(n/2) + cn$

Here merging operation is proportional to n.

$$T(n) = \begin{cases} a & , n=1 \\ 2T(n/2) + cn, & \forall\, n>1 \end{cases}$$

a, c are constants.

mergesort(low, mid)
mergesort(mid+1, high)

Merge

$T(n) = 2T(n/2) + cn$

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

$T(n) = 2[2T(n/4)] + 2.\frac{cn}{2} + cn$

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

$T(n/4) = 2T(n/8) + \left(\frac{cn}{4}\right)$

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

$= 8T(n/8) + \cancel{4} \times \frac{cn}{\cancel{4}} + 2cn$

$= 8T(n/8) + 3cn \Rightarrow 2^3 T\left(\frac{n}{2^3}\right) + 3cn$

upto k times:

$= 2^k T\left(\frac{n}{2^k}\right) + kcn$

assume $\frac{n}{2^k} = 1$

$n = 2^k$

$(k = \log_2 n)$

$$= 2^K T\left(\frac{\cancel{2^K}}{\cancel{2^K}}\right) + KCn$$

$$= 2^K \underset{\downarrow a.}{T(1)} + KCn$$

$$= an + \log_2 n * c * n$$

$$= an + cn\log_2 n$$

By ignoring the lower order terms and constants

$$T(n) = O(n\log n)$$

$$\boxed{T(n) = O(n\log n)}$$