**EXPERIMENT N0-4**

AIM: Comparative analysis of Quick sort and Merge sort

**QUICK SORT:**

**CODE:**

#include <stdio.h>

int partition(int arr[], int low, int high)

{

int temp;

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

if (arr[j] <= pivot)

{

i++;

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return (i + 1);

}

void quick\_sort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quick\_sort(arr, low, pi - 1);

quick\_sort(arr, pi + 1, high);

}

}

int main()

{

int n, i;

printf("enter the number of elements:\n");

scanf("%d", &n);

int arr[n];

printf("enter the elements:\n");

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

{

scanf("%d", &arr[i]);

}

quick\_sort(arr, 0, n - 1);

printf("\nAFTER APPLYING QUICK SORT\n");

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

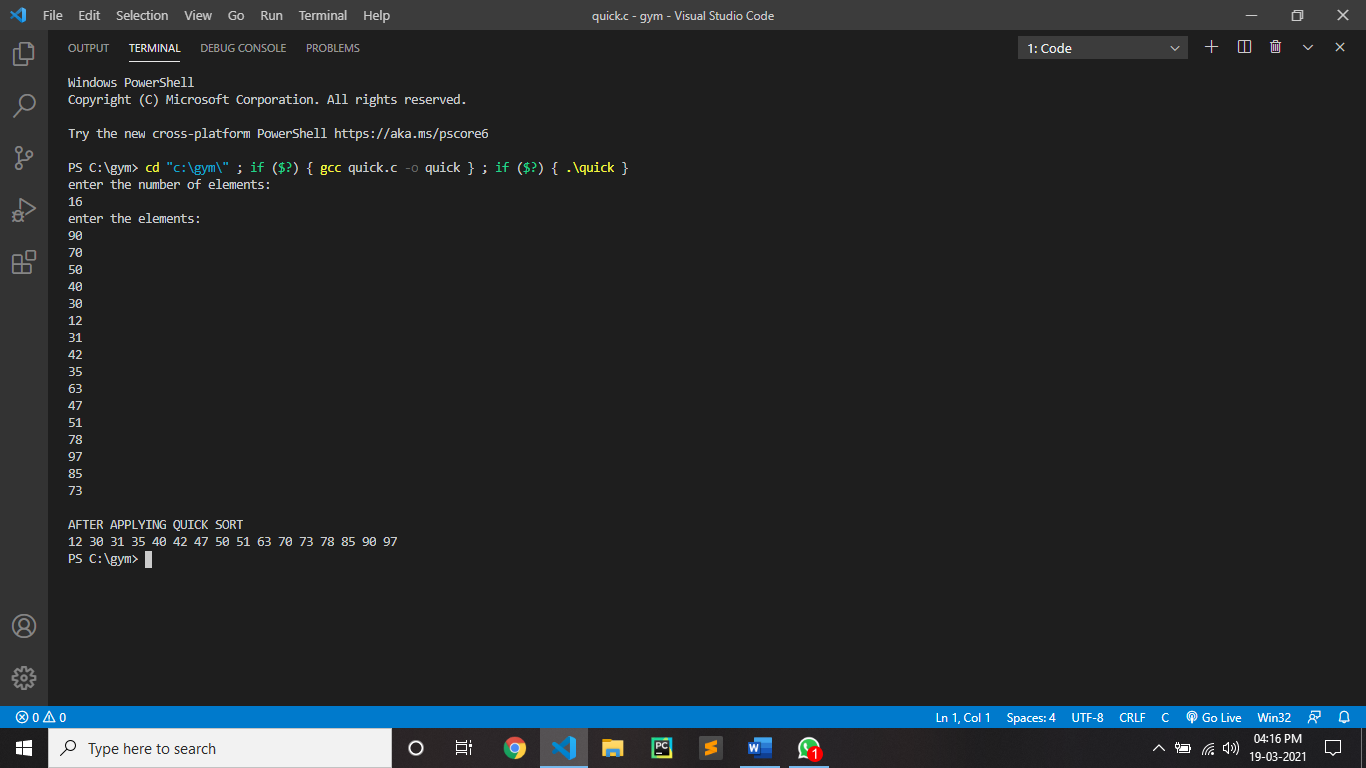
{

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

}

}

**OUTPUT:**



**MERGE SORT:**

**CODE:**

#include <stdio.h>

int merge(int arr[], int start, int mid, int end)

{

int i, j, k;

int num1 = mid - start + 1;

int num2 = end - mid;

int arr1[num1], arr2[num2];

for (i = 0; i < num1; i++)

arr1[i] = arr[start + i];

for (j = 0; j < num2; j++)

arr2[j] = arr[mid + 1 + j];

i = 0;

j = 0;

k = start;

while (i < num1 && j < num2)

{

if (arr1[i] <= arr2[j])

{

arr[k] = arr1[i];

i++;

}

else

{

arr[k] = arr2[j];

j++;

}

k++;

}

while (i < num1)

{

arr[k] = arr1[i];

i++;

k++;

}

while (j < num2)

{

arr[k] = arr2[j];

j++;

k++;

}

}

int divide(int arr[], int start, int end)

{

if (start < end)

{

int mid;

mid = (start + end) / 2;

divide(arr, start, mid);

divide(arr, mid + 1, end);

merge(arr, start, mid, end);

}

}

int main()

{

int n, i;

printf("enter the number of elements:\n");

scanf("%d", &n);

int arr[n];

printf("enter the elements:\n");

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

{

scanf("%d", &arr[i]);

}

divide(arr, 0, n - 1);

printf("\nAFTER APPLYING MERGE SORT\n");

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

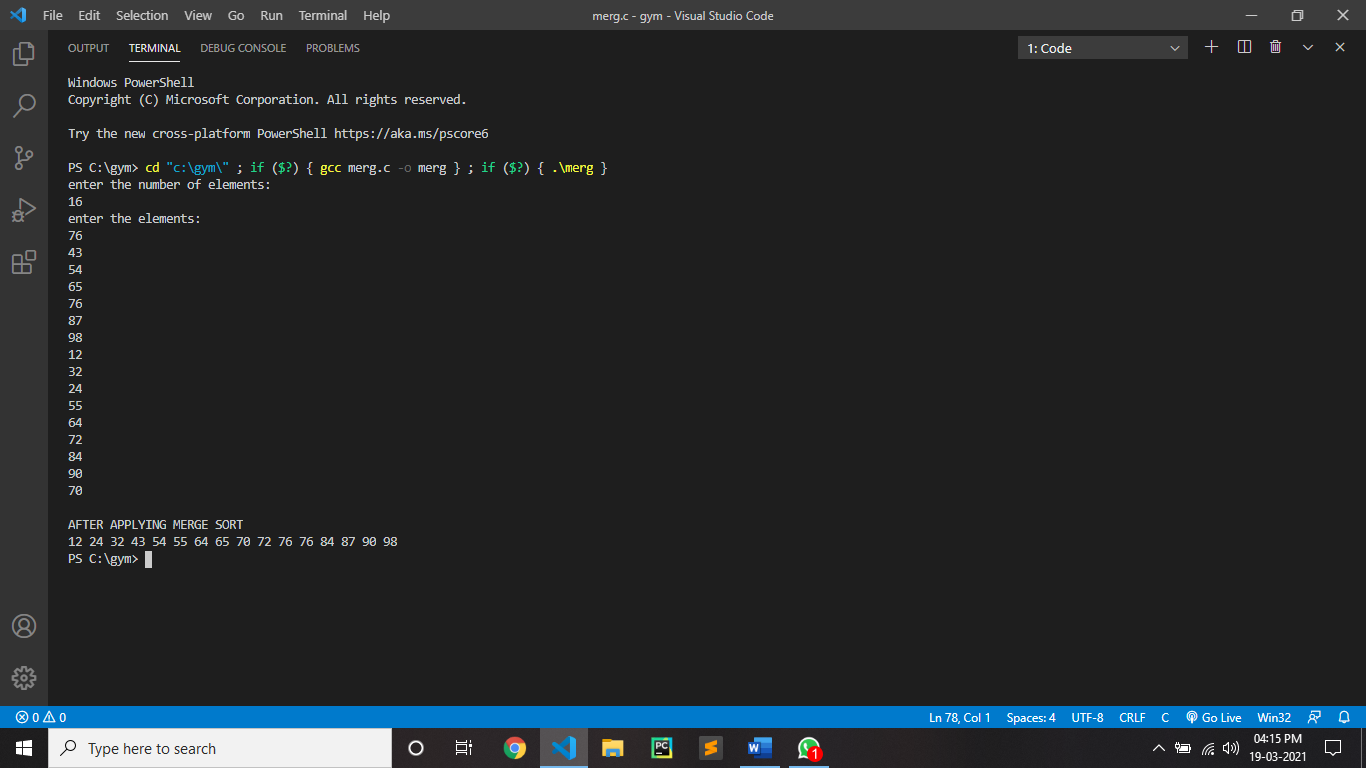
{

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

}

}

**OUTPUT:**



CONCLUSION:

By performing above practical we can conclude that Merge sort is more efficient and works faster than quick sort in case of larger array size , Quick sort is more efficient and works faster than merge sort in case of smaller array size. Quick sort is not stable. Merge sort is stable. For quick sort the best case for time complexity is O(nlogn) and the average case is O(nlogn) and the worst case is O(n^2) .For insertion sort the best case for time complexity is O(nlogn) and the average case is O(nlogn) and the worst case is O(nlogn).Space complexity of merge sort is O(n) and for space complexity quick sort is O(nlogn)