UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Experiment No. 01

Student Name: Rishav Kumar
UID: 22MCC20039
Branch: MCA - CCD
Section/Group: MCD-1/ Grp B
Semester: I
Date of Performance: 26th Sept 22
Subject Name: Design and analysis of algorithm
Subject Code: 22CAP-646

1. **Task to be done:**

   A) Sort the list by quick sort and write an algorithm for quick sort.

   B) Explain divide and conquer and write an algorithm for merge sort.

2. **Algorithm for experiment/practical:**

   A)

```
//partition function which rearranges the array such that
// all the smaller elements are left to pivot element
// and all the larger elements are right to pivot element.

partition(arr[], lo, hi)
    pivot = arr[hi]
    PIndex = lo     // place for swapping
    for i := lo to hi - 1 do
        if arr[i] <= pivot then
            swap arr[PIndex] with arr[i]
            PIndex = PIndex + 1
    swap arr[PIndex] with arr[hi]
    return PIndex

// we divide the array into two subarrays around
// the pivot and recursively call for them separately.

quicksort(arr[], lo, hi)
    if lo < hi
        PIndex = partition(arr, lo, hi)
        quicksort(arr, lo , p-1)
        quicksort(arr, p+1, hi)
```

B)

```
mergeSort(arr[],l,r)  //arr is array, l is left, r is right
{
   if(l<r)
   {
      midpoint = (l+r)/2
      mergeSort(arr,l,m)
      mergeSort(arr,m+1,r)
      merge(arr,l,m,r)
   }
}
```

3. Code for experiment/practical:

A)

```
# include <iostream>
using
namespace
std;

void
swap(int * a, int * b)
{
    int
t = *a;
*a = *b;
*b = t;
}

void
printArray(int
array[], int
size) {
    int
i;
for (i = 0; i < size; i++)
cout << array[i] << " ";
cout << endl;
}

int
partition(int
array[], int
low, int
high) {

    int
pivot = array[high];

int
i = (low - 1);

for (int j = low; j < high; j++)
{
if (array[j] <= pivot)
{
```

```
    i + +;

swap( & array[i], & array[j]);
}
}

swap( & array[i + 1], & array[high]);

return (i + 1);
}

void
quickSort(int
array[], int
low, int
high) {
if (low < high) {

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

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);
}
}

int
main()
{
int
data[] = {44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88};
int
n = sizeof(data) / sizeof(data[0]);

cout << "Unsorted Array: \n";
printArray(data, n);

quickSort(data, 0, n - 1);

cout << "Sorted array in ascending order: \n";
printArray(data, n);
}
```

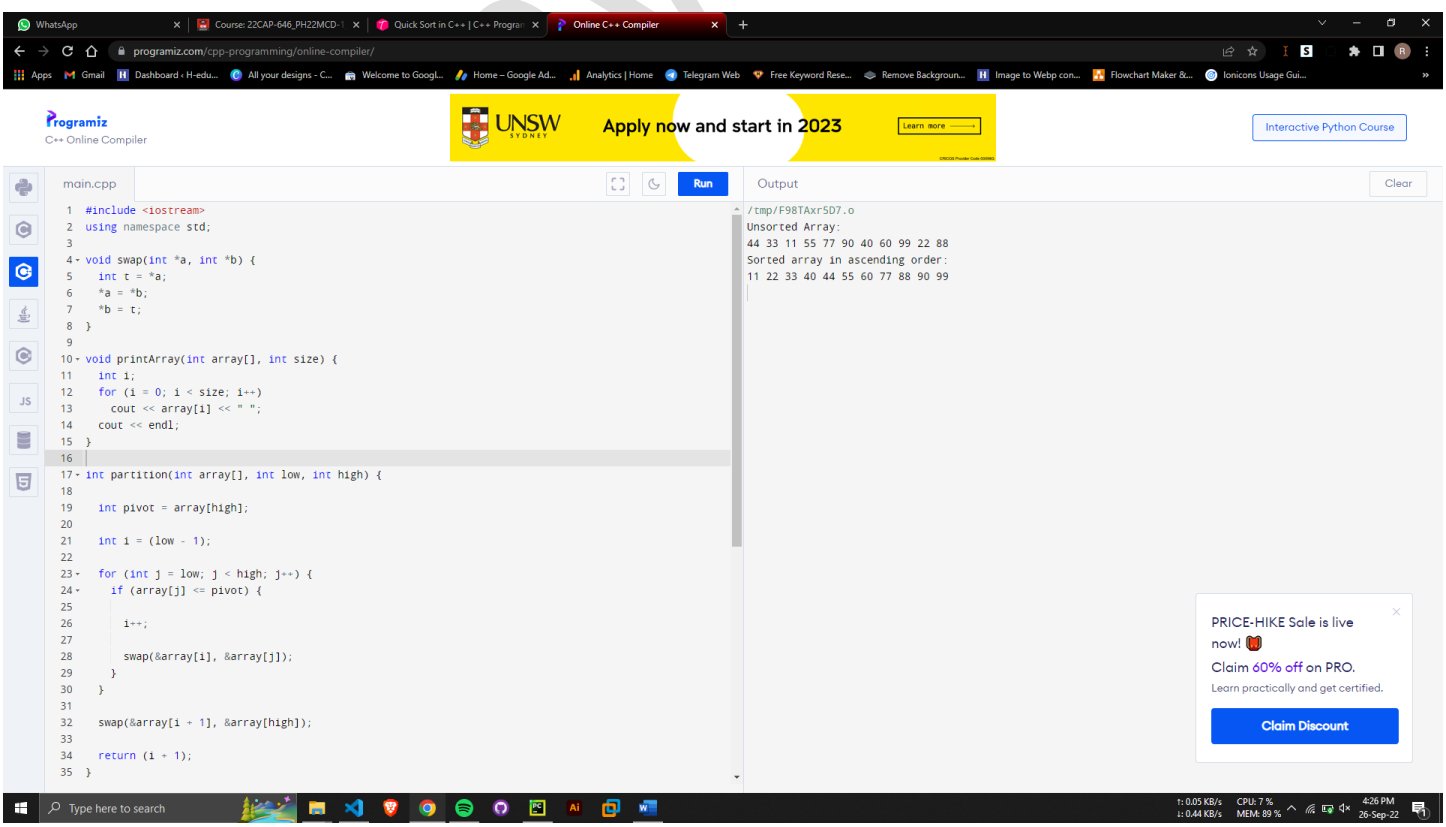
B)

```
#include <iostream>
  using namespace std;

void merge(int arr[], int l, int m, int r) {
  int i = l;
  int j = m + 1;
  int k = l;

  /* create temp array */
  int temp[5];

  while (i <= m && j <= r) {
    if (arr[i] <= arr[j]) {
      temp[k] = arr[i];
      i++;
      k++;
    } else {
      temp[k] = arr[j];
      j++;
      k++;
    }

  }

  /* Copy the remaining elements of first half, if there are any */
  while (i <= m) {
    temp[k] = arr[i];
    i++;
    k++;

  }

  /* Copy the remaining elements of second half, if there are any */
  while (j <= r) {
    temp[k] = arr[j];
    j++;
    k++;
  }

  /* Copy the temp array to original array */
  for (int p = l; p <= r; p++) {
    arr[p] = temp[p];
  }
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r) {
  if (l < r) {
    // find midpoint
    int m = (l + r) / 2;

    // recurcive mergesort first and second halves
    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);

    // merge
    merge(arr, l, m, r);
```

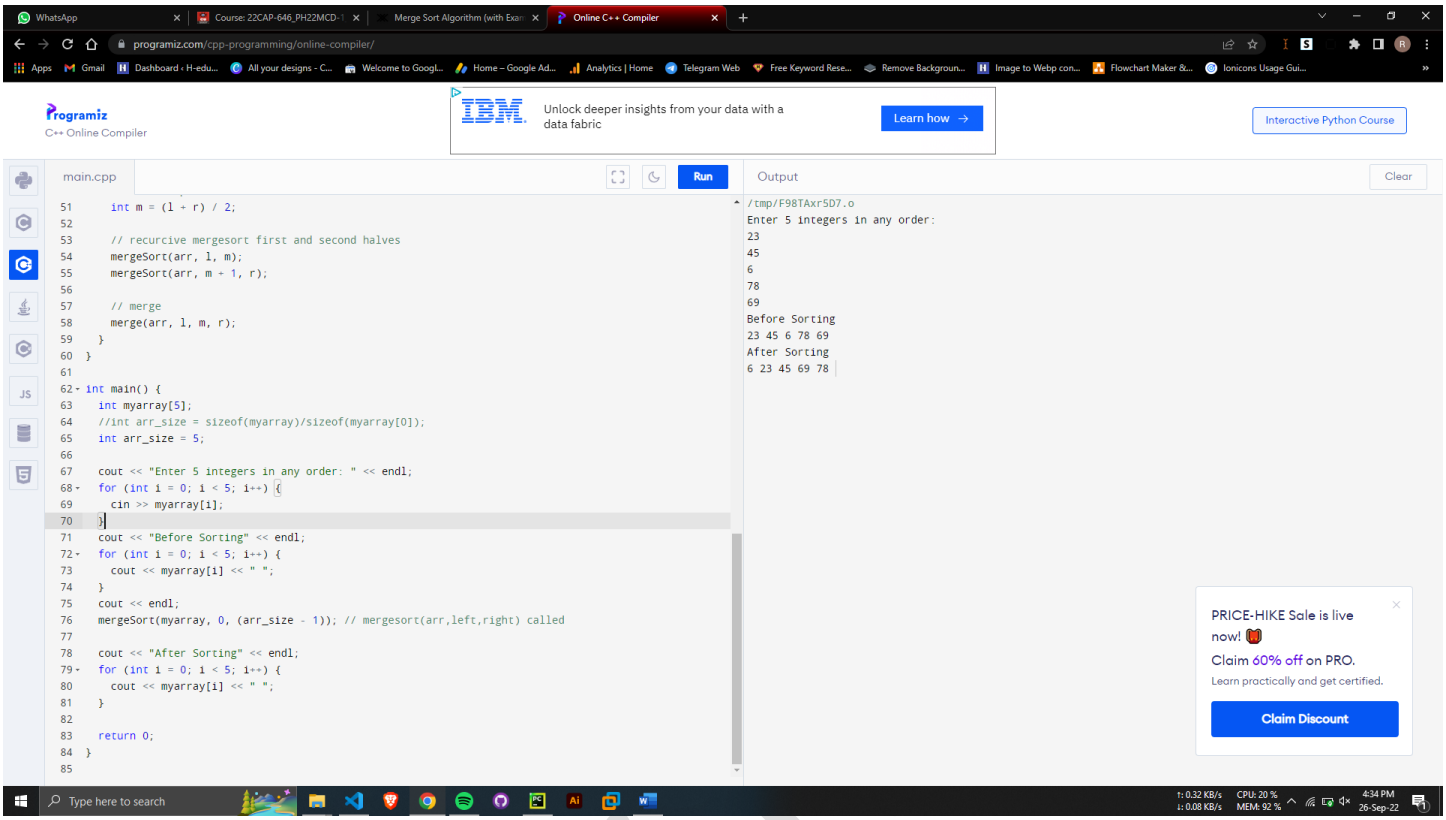
```
  }
}

int main() {
  int myarray[5];
  //int arr_size = sizeof(myarray)/sizeof(myarray[0]);
  int arr_size = 5;

  cout << "Enter 5 integers in any order: " << endl;
  for (int i = 0; i < 5; i++) {
    cin >> myarray[i];
  }
  cout << "Before Sorting" << endl;
  for (int i = 0; i < 5; i++) {
    cout << myarray[i] << " ";
  }
  cout << endl;
  mergeSort(myarray, 0, (arr_size - 1)); // mergesort(arr,left,right) called

  cout << "After Sorting" << endl;
  for (int i = 0; i < 5; i++) {
    cout << myarray[i] << " ";
  }

  return 0;
}
```

4. Output:

A)



B)



Learning outcomes (What I have learned):

1. Learned to implement the quick sort and algorithm.

2. Learned to implement the merge sort and algorithm.

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Demonstration and Performance		22

********** THE END **********