

Experiment No. 01

Student Name: **Rishav Kumar**

Branch: **MCA - CCD**

Semester: **I**

Subject Name: **Design and analysis of algorithm**

UID: **22MCC20039**

Section/Group: **MCD-1/ Grp B**

Date of Performance: **26th Sept 22**

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;

        // recursive 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

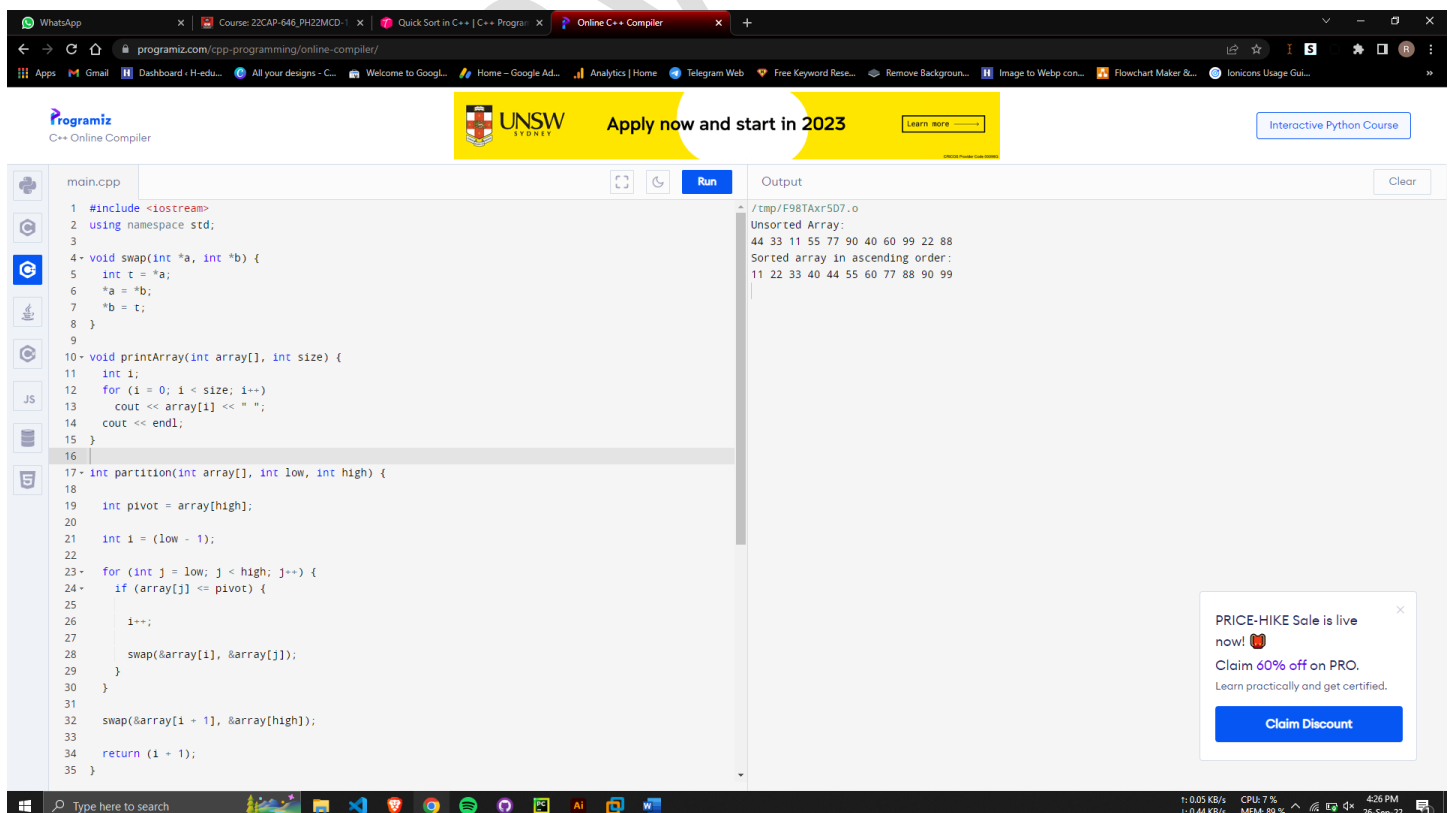
    cout << "After Sorting" << endl;
    for (int i = 0; i < 5; i++) {
        cout << myarray[i] << " ";
    }

    return 0;
}

```

4. Output:

A)



The screenshot shows a web browser window with the URL `programiz.com/cpp-programming/online-compiler/`. The page header includes the Programiz logo and a banner for UNSW Sydney with the text "Apply now and start in 2023". The main content area displays the C++ code for a merge sort algorithm. The output window shows the following results:

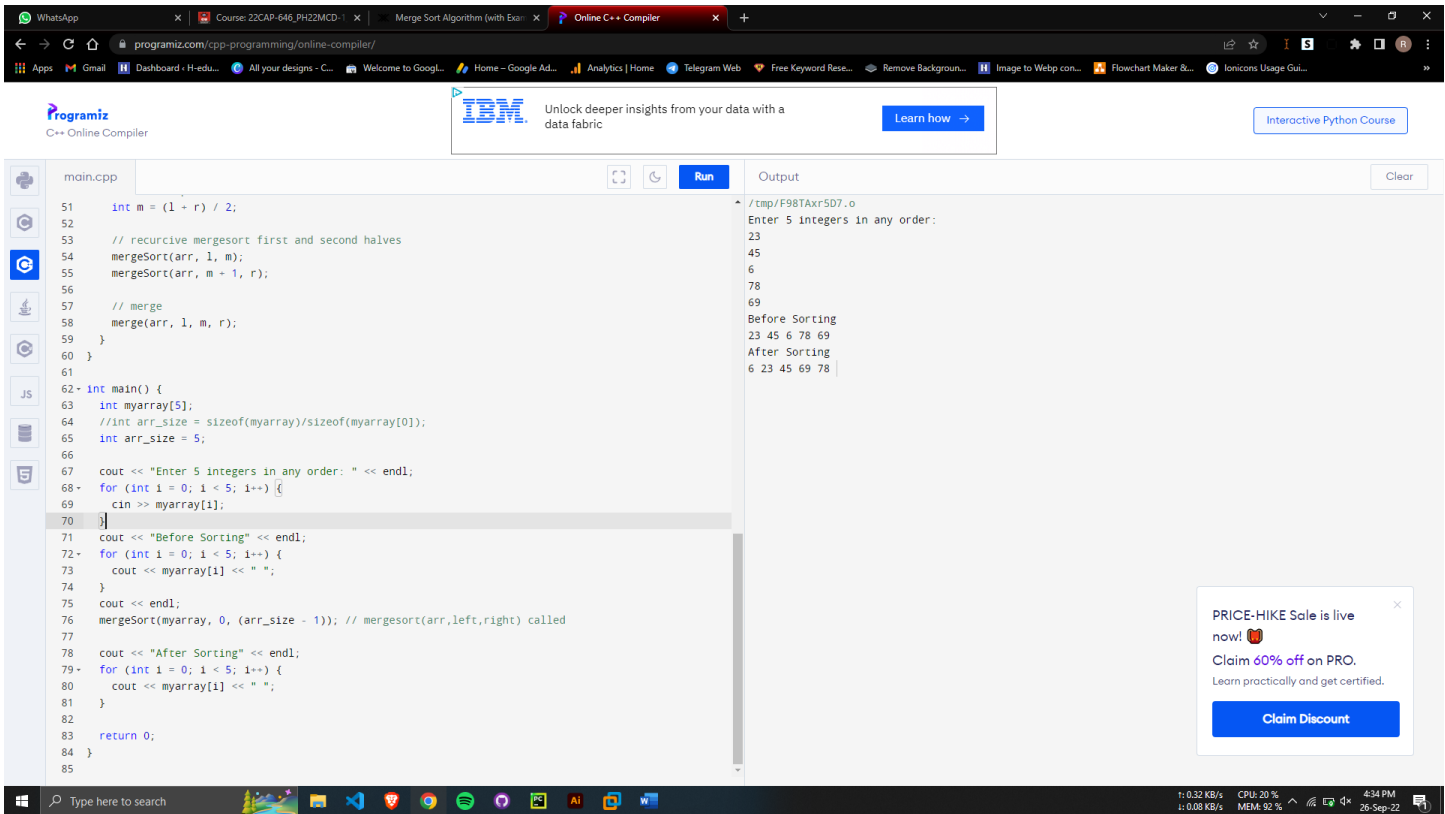
```

/tmp/F98TAxr5D7.o
Unsorted Array:
44 33 11 55 77 90 40 60 99 22 88
Sorted array in ascending order:
11 22 33 40 44 55 60 77 88 90 99

```

A promotional banner for a "PRICE-HIKE Sale" is visible in the bottom right corner, offering a 60% discount on PRO courses.

B)



The screenshot shows a web browser with the URL `programiz.com/cpp-programming/online-compiler/`. The page displays a C++ code editor with a merge sort algorithm implementation. The code is as follows:

```

1  int m = (l + r) / 2;
2
3  // recursive mergesort first and second halves
4  mergeSort(arr, l, m);
5  mergeSort(arr, m + 1, r);
6
7  // merge
8  merge(arr, l, m, r);
9
10 }
11
12 int main() {
13     int myarray[5];
14     //int arr_size = sizeof(myarray)/sizeof(myarray[0]);
15     int arr_size = 5;
16
17     cout << "Enter 5 integers in any order: " << endl;
18     for (int i = 0; i < 5; i++) {
19         cin >> myarray[i];
20     }
21     cout << "Before Sorting" << endl;
22     for (int i = 0; i < 5; i++) {
23         cout << myarray[i] << " ";
24     }
25     cout << endl;
26     mergeSort(myarray, 0, (arr_size - 1)); // mergesort(arr,left,right) called
27
28     cout << "After Sorting" << endl;
29     for (int i = 0; i < 5; i++) {
30         cout << myarray[i] << " ";
31     }
32     return 0;
33 }

```

The output of the program is shown on the right side of the editor:

```

/cmp/F98TAxr5D7.o
Enter 5 integers in any order:
23
45
6
78
69
Before Sorting
23 45 6 78 69
After Sorting
6 23 45 69 78

```

A promotional banner for a "PRICE-HIKE Sale" is visible in the bottom right corner of the browser window.

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 *****