****

**LAB EXERCISE 4**

Name: Jayannthan P T

Dept: CSE ‘A’

Roll No.: 205001049

1. Implement Merge Sort and take a snapshot of the function calling stack and recursive depth
   1. **Code:**

*// Implement Merge Sort and take a snapshot of the function calling stack and recursive*

*depth*

#include <iostream>

using **namespace** std;

**void** display(**int** **\***array, **int** size)

{

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

    {

        cout << array[i] << " ";

    }

    cout << endl;

}

**void** display\_specific(**int** **\***array, **int** b, **int** l)

{

    for (**int** i = b; i < l; i++)

        cout << array[i] << " ";

    cout << endl;

}

**void** merge(**int** **\***array, **int** l, **int** m, **int** r)

{

**int** size\_left, size\_right;

    size\_left = m - l + 1;

    size\_right = r - m;

**int** left\_array[size\_left], right\_array[size\_right];

    for (**int** i = 0; i < size\_left; i++)

        left\_array[i] = array[l + i];

    for (**int** j = 0; j < size\_right; j++)

        right\_array[j] = array[m + 1 + j];

**int** i = 0;

**int** j = 0;

**int** k = l;

    while (i < size\_left && j < size\_right)

    {

        if (left\_array[i] <= right\_array[j])

        {

            array[k] = left\_array[i];

            i++;

        }

        else

        {

            array[k] = right\_array[j];

            j++;

        }

        k++;

    }

    while (i < size\_left)

    {

        array[k] = left\_array[i];

        i++;

        k++;

    }

    while (j < size\_right)

    {

        array[k] = right\_array[j];

        j++;

        k++;

    }

    cout << endl;

    cout << "Left:";

    display(left\_array, size\_left);

    cout << "Right:";

    display(right\_array, size\_right);

    cout << "Merged:";

    display\_specific(array, l, k);

}

**void** mergeSort(**int** **\***array, **int** l, **int** r)

{

**int** m;

    if (l < r)

    {

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

        cout << "\nDividing array:";

        display\_specific(array, l, r+1);

        cout << "into ";

        display\_specific(array, l, m+1);

        cout << "and ";

        display\_specific(array, m + 1, r+1);

        mergeSort(array, l, m);

        mergeSort(array, m + 1, r);

        merge(array, l, m, r);

    }

}

**int** main()

{

**int** n;

    cout << "Enter the number of elements: ";

    cin >> n;

**int** arr[n];

    cout << "Enter elements:" << endl;

    for (**int** i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    cout << "Array before Sorting: ";

    display(arr, n);

    mergeSort(arr, 0, n - 1);

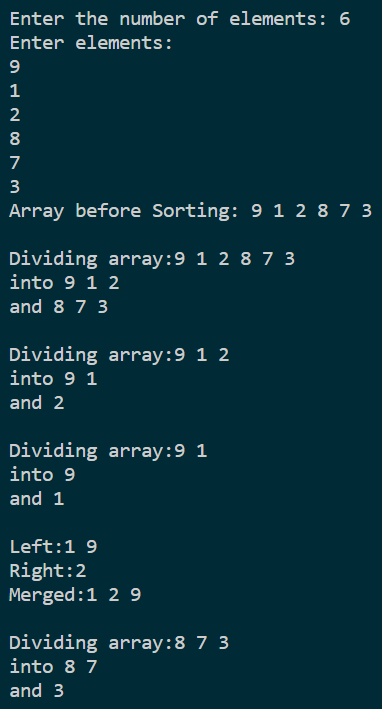
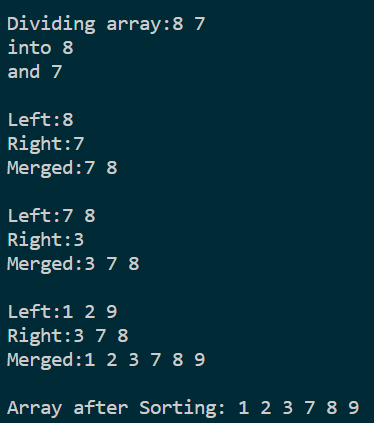
    cout << endl;

    cout << "Array after Sorting: ";

    display(arr, n);

}

* 1. **Output:**

1. Implement Merge Sort and call insertion sort for n=12, instead of recursive calls.
   1. **Code:**

*// Implement Merge Sort and call insertion sort for n=12, instead of recursive calls.*

#include <iostream>

using **namespace** std;

**void** display(**int** **\***array, **int** size)

{

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

    {

        cout << array[i] << " ";

    }

    cout << endl;

}

**void** display\_specific(**int** **\***array, **int** b, **int** l)

{

    for (**int** i = b; i < l; i++)

        cout << array[i] << " ";

    cout << endl;

}

**void** merge(**int** **\***array, **int** l, **int** m, **int** r)

{

    for (**int** i = l; i <= r; i++)

    {

**int** big = array[i];

**int** j = i - 1;

        while (big < array[j] && j >= l)

        {

            array[j + 1] = array[j];

            --j;

        }

        array[j + 1] = big;

    }

    cout << endl;

    cout << "Left:";

    display\_specific(array, l, m + 1);

    cout << "Right:";

    display\_specific(array, m + 1, r + 1);

    cout << "Merged:";

    display\_specific(array, l, r + 1);

}

**void** mergeSort(**int** **\***array, **int** l, **int** r)

{

**int** m;

    if (l < r)

    {

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

        cout << "\nDividing array:";

        display\_specific(array, l, r + 1);

        cout << "into ";

        display\_specific(array, l, m + 1);

        cout << "and ";

        display\_specific(array, m + 1, r + 1);

        mergeSort(array, l, m);

        mergeSort(array, m + 1, r);

        merge(array, l, m, r);

    }

}

**int** main()

{

**int** n;

    cout << "Enter the number of elements: ";

    cin >> n;

**int** arr[n];

    cout << "Enter elements:" << endl;

    for (**int** i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    cout << "Array before Sorting: ";

    display(arr, n);

    mergeSort(arr, 0, n - 1);

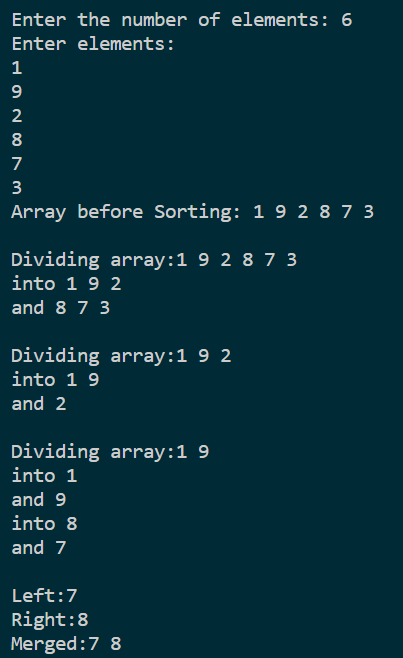
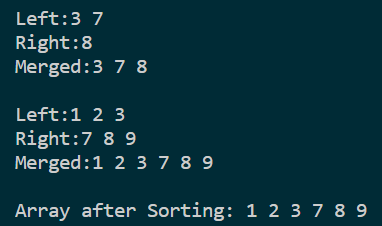
    cout << endl;

    cout << "Array after Sorting: ";

    display(arr, n);

}

* 1. **Output:**

1. Implement QuickSort
   1. **Code:**

*// Implement Quicksort*

#include <iostream>

using **namespace** std;

**void** swap(**int** **\***a, **int** **\***b)

{

**int** temp = \*a;

    \*a = \*b;

    \*b = temp;

}

**void** display(**int** **\***array, **int** size)

{

    for (**int** 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** n;

    cout << "Enter the number of elements: ";

    cin >> n;

**int** arr[n];

    cout << "Enter elements:" << endl;

    for (**int** i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    cout << "Array before Sorting: ";

    display(arr, n);

    quickSort(arr, 0, n - 1);

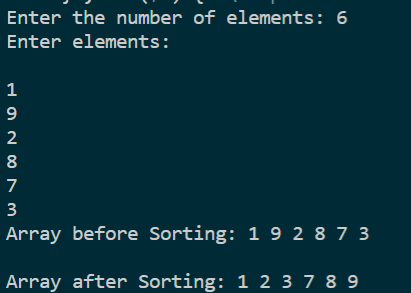
    cout << endl;

    cout << "Array after Sorting: ";

    display(arr, n);

}

* 1. **Output:**



1. Find the Kth Smallest/Largest Element in Unsorted Array.
   1. **Code:**

*// Implement QuickSelection*

#include <iostream>

using **namespace** std;

**void** swap(**int** **\***a, **int** **\***b)

{

**int** temp = \*a;

    \*a = \*b;

    \*b = temp;

}

**void** display(**int** **\***array, **int** size)

{

    for (**int** 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);

}

**int** quickSelect(**int** array[], **int** low, **int** high, **int** k)

{

    if ((low < high) and (k > 0) and (k < high - low + 1))

    {

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

        if ((pi - low) == (k - 1))

        {

            return array[pi];

        }

        if (pi - low > k - 1)

        {

            return quickSelect(array, low, pi - 1, k);

        }

        else

        {

            return quickSelect(array, pi + 1, high, k - pi + low - 1);

        }

    }

}

**int** main()

{

**int** n;

    cout << "Enter the number of elements: ";

    cin >> n;

**int** arr[n];

    cout << "Enter elements:" << endl;

    for (**int** i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

**int** k;

**int** choice;

    cout << "Menu:\n\t1.Kth Smallest Element\n\t2.Kth Largest Element\nEnter Choice:";

    cin >> choice;

    switch (choice)

    {

    case 1:

        cout << "Enter k to find kth smallest element in array: ";

        cin >> k;

        cout << "K-th Smallest Element is " << quickSelect(arr, 0, n - 1, k);

        break;

    case 2:

        cout << "Enter k to find kth largest element in array: ";

        cin >> k;

        cout << "K-th Largest Element is " << quickSelect(arr, 0, n - 1, n - k + 1);

        break;

    default:

        return 0;

        break;

    }

}

* 1. **Output:**

