SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

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

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);
}
```

Output:

```
Enter the number of elements: 6
Enter elements:
9
1
2
8
7
3
Array before Sorting: 9 1 2 8 7 3

Dividing array:9 1 2 8 7 3
into 9 1 2
and 8 7 3

Dividing array:9 1 2
into 9 1
and 2

Dividing array:9 1
into 9
and 1

Left:1 9
Right:2
Merged:1 2 9

Dividing array:8 7 3
into 8 7
and 3

Dividing array:8 7
into 8
and 7

Left:8
Right:7
Merged:7 8

Left:7 8
Right:3
Merged:3 7 8

Left:1 2 9
Right:3 7 8
Merged:1 2 3 7 8 9

Array after Sorting: 1 2 3 7 8 9
```

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

Code:

```cpp
// 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);
}
```

Output:

```
Enter the number of elements: 6
Enter elements:
1
9
2
8
7
3
Array before Sorting: 1 9 2 8 7 3

Dividing array:1 9 2 8 7 3
into 1 9 2
and 8 7 3

Dividing array:1 9 2
into 1 9
and 2

Dividing array:1 9
into 1
and 9
into 8
and 7

Left:7
Right:8
Merged:7 8
```

```
Left:3 7
Right:8
Merged:3 7 8

Left:1 2 3
Right:7 8 9
Merged:1 2 3 7 8 9

Array after Sorting: 1 2 3 7 8 9
```

3. Implement QuickSort

Code:

```cpp
// 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);
}
```

Output:

```
Enter the number of elements: 6
Enter elements:

1
9
2
8
7
3
Array before Sorting: 1 9 2 8 7 3

Array after Sorting: 1 2 3 7 8 9
```

4. Find the Kth Smallest/Largest Element in Unsorted Array.

Code:

```cpp
// 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;
    }
}
```

Output:

```
Enter the number of elements: 9
Enter elements:
1
9
8
2
3
7
6
4
5
```

```
Menu:
        1.Kth Smallest Element
        2.Kth Largest Element
Enter Choice:1
Enter k to find kth smallest element in array: 5
K-th Smallest Element is 5
```

```
Menu:
        1.Kth Smallest Element
        2.Kth Largest Element
Enter Choice:2
Enter k to find kth largest element in array: 5
K-th Largest Element is 5
```