Assignment – 4

18. Create a function to sort n array using bubble sort.

Source Code: main()

```
#include <iostream>
using namespace std;
void swap(int *, int *);
void bubble_sort(int *, int);
int main()
{
    int arr[], size = sizeof(arr) / sizeof(arr[0]) - 1;
    cout << "How many element you want to insert: ";
    cin >> size;
    if (size <= 0) {cout << "Invalid input"; exit(0);}
    cout << "Enter the array elements: ";
    for (int i = 0; i < size; i++)
        cin >> arr[i];
    cout << "The unsorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];
    cout << endl;
    bubble_sort(arr, size);
    cout << "The sorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];
    cout << endl;

    return 0;
}
```

Source Code: swap(a,b)

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

Source Code: bubbleSort(arr,size)

```
void bubble_sort(int *arr, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size - i; j++)
        {
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
        }
    }
}
```

Output

```
How many elements you want to insert: 5
Enter the array elements: 5 4 3 2 1
The unsorted array is:  5 4 3 2 1
The sorted array is:  1 2 3 4 5
```

19. Write a program to perform the quick sort.

Source Code: main()

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

void swap(int *, int *);
void quick_sort(int *, int, int);
int partition(int *arr, int start, int end);
int main()
{
    int arr[10], size;
    cout << "How many element you want to insert: ";
    cin >> size;
    if (size <= 0){cout << "Invalid input";exit(0);}
    cout << "Enter the array elements: ";
    for (int i = 0; i < size; i++)
        cin >> arr[i];
    cout << "The unsorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];
    cout << endl;
    quick_sort(arr, 0, size);
    cout << "The sorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];
    cout << endl;
    return 0;
}
```

Source Code: swap(a,b)

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

Source Code: quickSort(arr,start,end)

```
void quick_sort(int *arr, int start, int end)
{
    if (start < end)
    {
        int pivot = partition(arr, start, end);
        quick_sort(arr, start, pivot - 1);
        quick_sort(arr, pivot + 1, end);
    }
}
```

Source Code: partition(arr,start,end)

```
int partition(int *arr, int start, int end)
{
    int pivot = arr[start], i = start, j = end;
    while (i < j)
    {
        while (arr[i] <= pivot)
            i++;
        while (arr[j] > pivot)
            j--;
        if (j > i)
            swap(&arr[i], &arr[j]);
    }
   swap(&arr[start], &arr[j]);
    return j;
}
```

Output

```
How many element you want to insert: 5
Enter the array elements: 3 1 9 4 6
The unsorted array is:  3 1 9 4 6
The sorted array is:  1 3 4 6 9
```

20. Implement the heapsort algorithm.

Source Code: main()

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

void swap(int *, int *);
void heapSort(int *arr, int);
void build_max_heap(int *, int size);
void heapify(int *, int, int);

int main()
{
    int size;
    cout << "How many element you want to insert: ";
    cin >> size;
    if (size <= 0)
    {
        cout << "Invalid input";
        exit(0);
    }

    int arr[size]; // Dynamically allocate based on the size input
    cout << "Enter the array elements: ";
    for (int i = 0; i < size; i++)
        cin >> arr[i];

    cout << "The unsorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];
    cout << endl;

    heapSort(arr, size);

    cout << "The sorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];
    cout << endl;

    return 0;
}
```

Source Code: swap(a,b)

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

Source Code: heapSort(arr,size)

```
void heapSort(int *arr, int size)
{
    build_max_heap(arr, size); // Build the initial max heap
    // Perform heap sort
    for (int i = size - 1; i > 0; i--) // Start from the end of the array
    {
        swap(&arr[0], &arr[i]); // Move the largest element to the end
        heapify(arr, i, 0);     // Heapify the reduced heap
    }
}
```

Source Code: heapify(arr,size,i)

```
void heapify(int *arr, int size, int i)
{
    int largest = i;       // Assume the current node is the largest
    int left = 2 * i + 1;  // Left child
    int right = 2 * i + 2; // Right child
    // Check if the left child exists and is greater than the current largest
    if (left < size && arr[left] > arr[largest])
        largest = left;
    // Check if the right child exists and is greater than the current largest
    if (right < size && arr[right] > arr[largest])
        largest = right;
    // If the largest element is not the current node
    if (largest != i)
    {
        swap(&arr[i], &arr[largest]); // Swap the current node with the largest
        heapify(arr, size, largest);  // Recursively heapify the affected subtree
    }
}
```

Source Code: build_max_heap(arr,size)

```
void build_max_heap(int *arr, int size)
{
   for (int i = size / 2 - 1; i >= 0; i--) // Start heapifying from the last non-leaf
node
       heapify(arr, size, i);
}
```

Output

```
How many element you want to insert: 5
Enter the array elements: 3 1 9 4 6
The unsorted array is:  3 1 9 4 6
The sorted array is:  1 3 4 6 9
```

21. Implement the merge algorithm.

Source Code: main()

```
#include <iostream>
#include <climits>
using namespace std;
void merge(int *, int, int, int);
void mergeSort(int *, int, int);

int main()
{
    int size;
    cout << "How many element you want to insert: ";
    cin >> size;
    if (size <= 0)
    {
        cout << "Invalid input";
        exit(0);
    }

    int arr[size]; // Dynamically allocate based on the size input
    int start = 0, end = size;
    cout << "Enter the array elements: ";
    for (int i = 0; i < size; i++)
        cin >> arr[i];

    cout << "Before sorting the array: ";
    for (int i = start; i <= end; i++)
        cout << " " << arr[i];

    cout << endl;

    mergeSort(arr, start, end);

    cout << "The sorted array is: ";
    for (int i = start; i <= end; i++)
        cout << " " << arr[i];

    cout << endl;
    return 0;
}
```

Source Code: swap(a,b)

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

Source Code: mergeSort(arr,start,end)

```
void mergeSort(int *arr, int start, int end)
{
   if (start < end)
   {
      int mid = (start + end) / 2;
      mergeSort(arr, start, mid);
      mergeSort(arr, mid + 1, end);
      merge(arr, start, mid, end);
   }
}
```

Source Code: merge(arr,start,mid,end)

```
void merge(int *arr, int start, int mid, int end)
{
   int temp[end];
   int i = start, j = mid + 1, k = 0;
   while (i <= mid && j <= end)
   {
      if (arr[i] < arr[j])
         temp[k++] = arr[i++];
      else
         temp[k++] = arr[j++];
   }
   while (i <= mid)
      temp[k++] = arr[i++];
   while (j <= end)
      temp[k++] = arr[j++];
   for (int i = start; i <= end; i++)
   {
      arr[i] = temp[i - start];
   }
}
```

Output

```
How many element you want to insert: 5
Enter the array elements: 3 1 9 4 6
The unsorted array is:  3 1 9 4 6
The sorted array is:  1 3 4 6 9
```

22. Implement the radix sort algorithm for integers.

Source Code: main()

```
#include <iostream>
#include <climits>
#define N 10
using namespace std;

int get_digit(int, int);
int find_max(int *, int);
void radix_sort(int *, int);
void count_sort(int *, int, int);

int main()
{
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int size = sizeof(arr) / sizeof(arr[0]) - 1;

    cout << "The unsorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];

    cout << endl;

    radix_sort(arr, size);

    cout << "The sorted array is: ";
    for (int i = 0; i < size; i++)
        cout << " " << arr[i];

    cout << endl;

    return 0;
}
```

Source Code: swap(a,b)

```
void radix_sort(int *arr, int size)
{
    for (int place = 1; find_max(arr, size) / place > 0; place *= 10)
        count_sort(arr, size, place);
}
```

Source Code: count_sort(arr,size,place)

```
void count_sort(int *arr, int size, int place)
{
    int output[size], count[N] = {0};

    for (int i = 0; i < size; i++)
    {
        int digit = get_digit(arr[i], place);
        count[digit]++;
    }

    for (int i = 1; i < N; i++)
        count[i] += count[i - 1];

    for (int i = size - 1; i >= 0; i--)
    {
        int digit = get_digit(arr[i], place);
        output[--count[digit]] = arr[i];
    }

    for (int i = 0; i < size; i++)
        arr[i] = output[i];
}
```

Source Code: get_digit(num,palce)

```
int get_digit(int num, int place)
{
    return (num / place) % 10;
}
```

Source Code: find_max(arr,size)

```
int find_max(int *arr, int size)
{
    int max = INT_MIN;

    for (int i = 0; i < size; i++)
    {
        if (max < arr[i])
            max = arr[i];
    }

    return max;
}
```

Output

```
The unsorted array is:  170 45 75 90 802 24 2

The sorted array is:  2 24 45 75 90 170 802
```