

**Q. 1) Write a C program to implement the Bubble sort algorithm.**

**Answer:**

**Algorithm:**

Step 1: Start.

Step 2: Input the array of numbers to be sorted.

Step 3: Begin at the first element of the array.

Step 4: Compare each pair of adjacent elements.

Step 5: If the pair is in the wrong order, swap them.

Step 6: Continue comparing and swapping until reaching the end of the array.

Step 7: After each pass, the largest unsorted element is placed in its correct position.

Step 8: Repeat steps 3 to 7 for the remaining unsorted elements.

Step 9: If no swaps occur during a pass, the array is sorted.

Step 10: Stop.

**Example:**

Initial Array: [5, 4, 1, 0, 3]

**First Pass:**

- Compare the first two elements (5 and 4). Since  $5 > 4$ , swap them.
  - Array becomes [4, 5, 1, 0, 3].
- Compare the next two elements (5 and 1). Since  $5 > 1$ , swap them.
  - Array becomes [4, 1, 5, 0, 3].
- Compare the next two elements (5 and 0). Since  $5 > 0$ , swap them.
  - Array becomes [4, 1, 0, 5, 3].
- Compare the next two elements (5 and 3). Since  $5 > 3$ , swap them.
  - Array becomes [4, 1, 0, 3, 5].
- After the first pass, the largest element 5 is in its correct position.

So, Array after first pass: [4, 1, 0, 3, 5]. similarly,

Array after second pass: [1, 0, 3, 4, 5]

Array after third pass: [0, 1, 3, 4, 5]

Since, in the fourth pass there is no swap needed. So the sorted array is [0, 1, 3, 4, 5].

**Program:**

```
#include <stdio.h>
```

```
int bubbleSort(int A[], int);
```

```
void display(int A[], int);
```

```
int main(){
```

```
    int terms;
```

```
    printf("\t*****Bubble sort Algorithm*****\n");
```

```
    printf("Enter the no. of terms to be sorted: ");
```

```
    scanf("%d", &terms);
```

```
    int arr[terms];
```

```
    printf("Enter the numbers for sorting: ");
```

```
    for(int i = 0; i < terms; i++){
```

```
        scanf("%d",&arr[i]);
```

```
    }
```

```
    bubbleSort(arr, terms);
```

```
    display(arr, terms);
```

```
    return 0;
```

```
}
```

```
int bubbleSort(int A[], int n){
```

```
    for (int i = 0; i < n - 1; i++){
```

```
        int swapped = 0;
```

```
        printf("\n");
```

```
        //Bubble sort logic
```

```
for (int j = 0; j < n-i-1; j++){  
    if (A[j] > A[j+1]){  
        int temp = A[j];  
        A[j] = A[j+1];  
        A[j+1] = temp;  
        swapped = 1;  
    }  
}  
printf("Pass %d:\n", i+1);  
for (int i = 0; i < n; i++){  
    printf("%d\t", A[i]);  
}  
if(!swapped)  
    break;  
}  
}  
  
void display(int A[], int n){  
    printf("\nThe sorted data in Ascending order: \n");  
    for (int i = 0; i < n; i++){  
        printf("%d\t", A[i]);  
    }  
    printf("\n");  
}
```

**Output:**

```

user@DolindraBahadurRaut:~/DSA/Sorting$ gcc bubbleSort.c
user@DolindraBahadurRaut:~/DSA/Sorting$ ./a.out
*****Bubble sort Algorithm*****
Enter the no. of terms to be sorted: 5
Enter the numbers for sorting: 5 4 1 0 3

Pass 1:
4      1      0      3      5
Pass 2:
1      0      3      4      5
Pass 3:
0      1      3      4      5
Pass 4:
0      1      3      4      5
The sorted data in Ascending order:
0      1      3      4      5
user@DolindraBahadurRaut:~/DSA/Sorting$

```

**Conclusion:**

Therefore, the bubble sort algorithm has been successfully implemented in C. The program correctly sorts the array as expected.

**Q.2) Write a program in C to sort the array (in ascending order) using insertion sort algorithm.**

**Answer:****Algorithm:**

Step 1: Start.

Step 2: Input the array to be sorted.

Step 3: Consider the first element as already sorted and start from the second element.

Step 4: Iterate through the unsorted elements (starting from the second element):

- Pick the current element (called key).
- Compare the key with the element to its left.
  
- If the key is smaller, shift the left element to the right.
- Continue comparing and shifting leftward until the correct position for the key is found.

Step 5: Insert the key at the correct position.

Step 6: Repeat step 4 to 5 for all elements until the entire array is sorted.

Step 7: Print the sorted array.

Step 8: Stop.

### **Example:**

Let's apply Insertion Sort on the array of size 5 with elements [5, -9, 12, 2, -1].

Compare -9 and 5. Hence,  $-9 < 5$  shift 5 to the right, and insert -9 at the first position. [-9, 5, 12, 2, -1]

Compare 12 and 5. No shifting needed, as 12 is already in the correct position.

Move to the fourth element (2). Compare with 12, shift 12 to the right, compare with 5, shift 5 to the right, and insert 2 at the second position.

Array: [-9, 2, 5, 12, -1]

Move to the fifth element (-1). Compare with 12, shift 12 to the right, compare with 5, shift 5 to the right, compare with 2, shift 2 to the right, and insert -1 at the second position.

Array: [-9, -1, 2, 5, 12]

Final sorted array: [-9, -1, 2, 5, 12].

### **Program:**

```
#include <stdio.h>
```

```
void displayArray(int arr[], int n) {
```

```
    for (int i = 0; i < n; i++) {
```

```
        printf("%d ", arr[i]);
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
void insertionSort(int arr[], int n) {
```

```

int i, key, j;

// Step 2: Iterate through the remaining unsorted elements
for (i = 1; i < n; i++) {
    key = arr[i]; // Step 3: Pick the current element
    j = i - 1;
    // Step 4: Compare and shift elements to the right
    while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j]; // Shift element to the right
        j = j - 1;
    }
    arr[j + 1] = key; // Step 5: Insert the key at its correct position
}
}

int main() {
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Array before Insertion Sort: "); // Display the array before sorting
    displayArray(arr, n);
    insertionSort(arr, n); // Perform Insertion Sort

    printf("Array after Insertion Sort: "); // Display the array after sorting

```

```

displayArray(arr, n);

return 0;

}

```

**Output:**

```

user@DolindraBahadurRaut:~/DSA/Sorting$ gcc insertionSort.c
user@DolindraBahadurRaut:~/DSA/Sorting$ ./a.out
Enter the number of elements in the array: 5
Enter 5 elements:
5 -9 12 2 -1
Array before Insertion Sort: 5 -9 12 2 -1
Array after Insertion Sort: -9 -1 2 5 12
user@DolindraBahadurRaut:~/DSA/Sorting$

```

**Conclusion:**

Therefore, we have successfully implemented a C program to sort an array using the Insertion Sort algorithm.

**Q.3) Write a C program to sort an array of integers using the selection sort algorithm.****Answer:****Algorithm:**

Step 1: Start.

Step 2: Input the array and assume it is unsorted.

Step 3: Start with the first element of the unsorted array and assume it is the smallest.

Step 4: Scan the unsorted array to find the actual smallest element.

Step 5: Swap the smallest element found with the first element of the unsorted array.

Step 6: Move to the next position and repeat steps 3 to 5 for the remaining unsorted part of the array except for the last element.

Step 7: Continue until the array is fully sorted.

Step 8: Print the array after sorting.

Step 9: Stop.

**Example:**

Initial Array:[14, 52, 16, -5, 11]

Step 1: First Iteration ( $i = 0$ )

- The unsorted array: [14, 52, 16, -5, 11]
- Smallest element is -5.
- Swap -5 with 14.
- New array: [-5, 52, 16, 14, 11]

Step 2: Second Iteration ( $i = 1$ )

- The unsorted array: [52, 16, 14, 11]
- Smallest element is 11.
- Swap 11 with 52.
- New array: [-5, 11, 16, 14, 52]

Step 3: Third Iteration ( $i = 2$ )

- The unsorted array: [16, 14, 52]
- Smallest element is 14.
- Swap 14 with 16.
- New array: [-5, 11, 14, 16, 52]

Step 4: Fourth Iteration ( $i = 3$ )

- The unsorted array: [16, 52]
- Smallest element is 16.
- No swap needed as 16 is already in the correct position.
- New array: [-5, 11, 14, 16, 52]

Step 5: Fifth Iteration ( $i = 4$ )

- Only one element left, no sorting needed.
- Final sorted array: [-5, 11, 14, 16, 52]

**Program:**

```
#include <stdio.h>
```



```

void selectionSort(int array[], int size) {
    int i, j, imin;
    for(i = 0; i < size - 1; i++) {
        imin = i; // Get index of minimum element
        for(j = i + 1; j < size; j++) {
            if(array[j] < array[imin]) {
                imin = j; // Update the index of the minimum element
            }
        }
        if (imin != i) { // Swap only if necessary
            int temp = array[i];
            array[i] = array[imin];
            array[imin] = temp;
        }
    }
}

int main() {
    int n;
    printf("Enter the size of array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array:\n");
    for(int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("\nArray before Sorting: ");
    for(int i = 0; i < n; i++) {

```

```

    printf("%d ", arr[i]);
}
printf("\n");
selectionSort(arr, n);
printf("Array after Sorting: ");
for(int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");
return 0;
}

```

**Output:**

```

user@DolindraBahadurRaut:~/DSA$ gcc selectionSort.c
user@DolindraBahadurRaut:~/DSA$ ./a.out
Enter the size of array: 5
Enter the elements of the array:
14 52 16 -5 11

Array before Sorting: 14 52 16 -5 11
Array after Sorting: -5 11 14 16 52

```

**Conclusion:**

Hence, we successfully implemented a C program to sort an array of integers using the selection sort algorithm.

**Q.4) Write a program to implement the quick sort algorithm.****Answer:****Algorithm:**

Step 1: Start.

Step 2: Input the array for sorting.

Step 3: Make the right-most index value as a pivot element.

Step 4: Partition the array into two subarrays using pivot element such that

- Left subarray contains elements less than the pivot element.
- Right subarray contains elements greater than the pivot element.

Step 5. Recursively apply quick sort for left and right subarrays until each subarray is formed of a single element

Step 5: Display the sorted array.

Step 6: Stop.

### **Example:**

Let's assume, Array is [15, -2, 25, 0, 5], and we need to sort it using the quick sort algorithm.

Step 1: Choose Pivot Element: Pivot = 5 (last element of the array).

Step 2: Partition into Left and Right Sub-arrays

Rearrange the array such that:

- Left Sub-array: Elements smaller than or equal to 5 are [-2, 0, 5].
- Right Sub-array: Elements greater than 5 are [15, 25].
- After partitioning, the array becomes: [-2, 0, 5, 15, 25].

Step 3: Recursive Call to Left and Right sub-array:

- Left sub-array: [-2, 0]
  - Pivot = 0 (last element of the sub-array).
  - Partitioning results in: [-2, 0]
- Right sub-array: [15, 25]
  - Pivot = 25 (last element of the sub-array).
  - Partitioning results in: [15, 25]

After all recursive calls, the final sorted array is: [-2, 0, 5, 15, 25]

### **Program:**

```
#include <stdio.h>

void swap(int *a, int *b) {
```

```

int t = *a;

*a = *b;

*b = t;
}

// function to find the partition position
int partition(int array[], int low, int high) {
    int pivot = array[high]; // select the rightmost element as pivot
    int i = (low - 1);
    for (int j = low; j < high; j++) {
        if (array[j] <= pivot) {
            i++;
            swap(&array[i], &array[j]);    // swap element at i with element at j
        }
    }

    swap(&array[i + 1], &array[high]); // swap the pivot element with the greater element at i
    return (i + 1); // return the partition point
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1); // recursive call on the left of pivot
        quickSort(arr, pi + 1, high); // recursive call on the right of pivot
    }
}

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

```

```

    printf("%d ", array[i]);
}
printf("\n");
}
int main() {
    int array[] = {15,-2,25,0,5};

    int n = sizeof(array) / sizeof(array[0]);

    printf("Unsorted Array\n");
    printArray(array, n);

    quickSort(array, 0, n - 1); // perform quicksort

    printf("Sorted array in ascending order: \n");
    printArray(array, n);

    return 0;
}

```

**Output:**

```

user@DolindraBahadurRaut:~/DSA$ gcc quickSort.c
user@DolindraBahadurRaut:~/DSA$ ./a.out
Unsorted Array
15 -2 25 0 5
Sorted array in ascending order:
-2 0 5 15 25

```

**Conclusion:**

Hence, we successfully implemented a C program to sort an array using the quick sort algorithm.

**Q.5) Write a C program to implement the merge sort algorithm .**

**Answer:****Algorithm:**

Step 1: Start.

Step 2: Accept the array that needs to be sorted.

Step 3: If the array has more than one element, divide the array into two halves (left and right).

Step 4: Recursively apply merge sort to the left half and the right half of the array until each subarray contains only one element (base case).

Step 5: Once all subarrays are reduced to a single element, start merging the sorted halves into a single sorted array.

Step 6: Continue merging the subarrays back together until the entire array is sorted.

Step 7: Once all subarrays are merged, the entire array is sorted.

Step 8: Print the sorted array.

Step 9: Stop.

### **Example:**

Initial Array: [12, 10, 8, 11]

Step 1: Divide the array into two halves: Left half: [12, 10] and Right half: [8, 11]

Step 2: Recursively divide the halves:

- [12, 10] becomes [12] and [10].
- [8, 11] becomes [8] and [11].

Step 3: Merge [12] and [10]:

- Compare 12 and 10.
- Result after merging: [10, 12]

Step 4: Merge [8] and [11]:

- Compare 8 and 11.
- Result after merging: [8, 11]

Step 5: Merge [10, 12] and [8, 11]:

- Compare 10 and 8:  $8 < 10$ , so place 8 first.

- Compare 10 and 11:  $10 < 11$ , so place 10 next.
- Compare 12 and 11:  $11 < 12$ , so place 11 next.
- Finally, place 12.
- Result after merging: [8, 10, 11, 12]

Final Sorted Array: [8, 10, 11, 12]

**Program:**

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
void merge(int arr[], int l, int m, int r){
```

```
    int i, j, k;
```

```
    int n1 = m - l + 1;
```

```
    int n2 = r - m;
```

```
    int L[n1], R[n2]; // Create temp arrays
```

```
    // Copy data to temp arrays L[] and R[]
```

```
    for (i = 0; i < n1; i++)
```

```
        L[i] = arr[l + i];
```

```
    for (j = 0; j < n2; j++)
```

```
        R[j] = arr[m + 1 + j];
```

```
    // Merge the temp arrays back into arr[l..r
```

```
    i = 0;
```

```
    j = 0;
```

```
    k = l;
```

```
    while (i < n1 && j < n2) {
```

```
        if (L[i] <= R[j]) {
```

```
            arr[k] = L[i];
```

```
            i++;
```

```

    }
    else {
        arr[k] = R[j];
        j++;
    }
    k++;
}

while (i < n1) { // Copy the remaining elements of L[] if there are any
    arr[k] = L[i];
    i++;
    k++;
}

while (j < n2) { // Copy the remaining elements of R[] if there are any
    arr[k] = R[j];
    j++;
    k++;
}
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void printArray(int A[], int size) { // Function to print an array

```



```

int i;

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

    printf("%d ", A[i]);

printf("\n");
}

int main(){

    int arr[] = {12,10,8,11};

    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");

    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

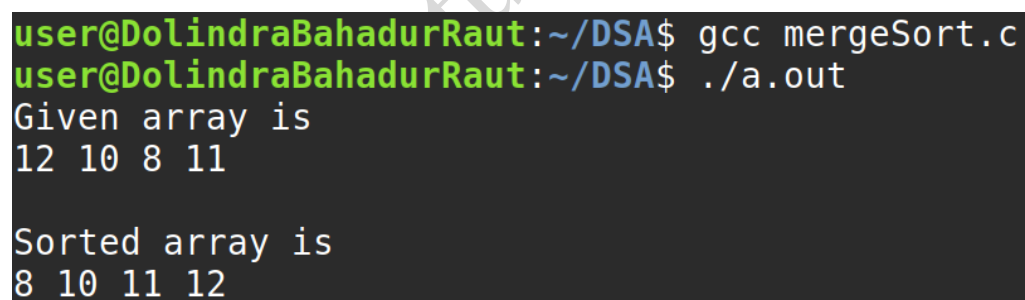
    printf("\nSorted array is \n");

    printArray(arr, arr_size);

    return 0;

}

```

**Output:**


```

user@DolindraBahadurRaut:~/DSA$ gcc mergeSort.c
user@DolindraBahadurRaut:~/DSA$ ./a.out
Given array is
12 10 8 11

Sorted array is
8 10 11 12

```

**Conclusion:**

Hence, we successfully implemented the Merge Sort algorithm in C, which recursively divides the array into smaller subarrays and merges them back in sorted order.