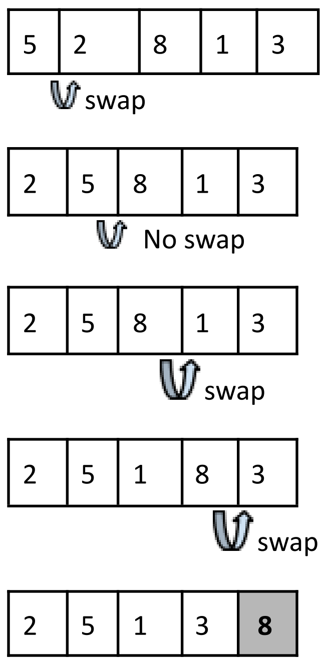
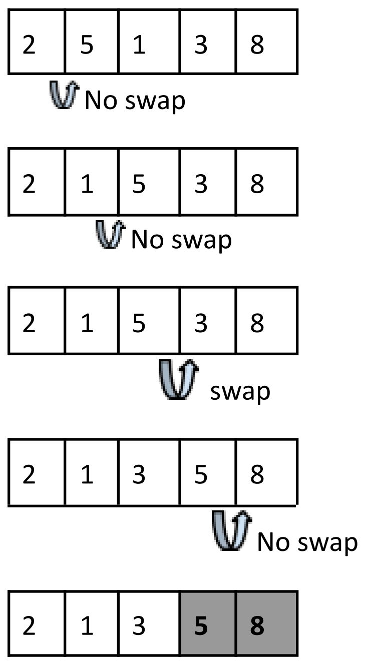
Name - Parepalli Manohar
Student ID - 0938792
Course - Data Structures

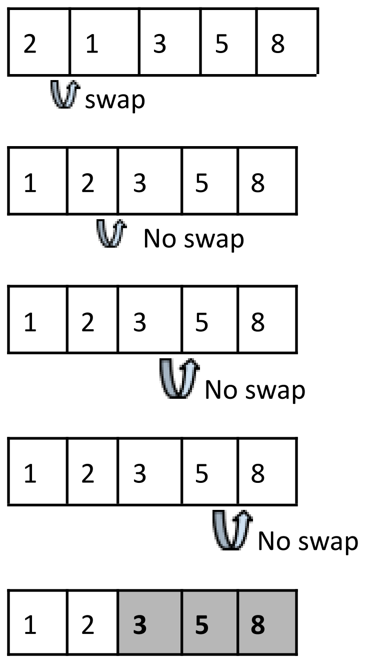
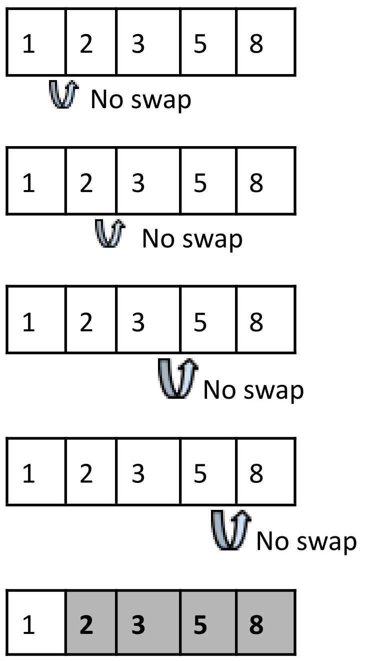
1.Manually sort the array ` [5, 2, 8, 1, 3] ` using the bubble sort technique. write down each step of the process.

Given Array:

5	2	8	1	3

Bubble Sort: According to my observation, In bubble sort after every step the large number comes to the end of the array.

Pass 1: Take the given array as input compare the numbers if one number is greater than the other then swap.	Pass 2: Take the array that came output from the first step and do the same process
 5 2 8 1 3 swap 2 5 8 1 3 No swap 2 5 8 1 3 swap 2 5 1 8 3 swap 2 5 1 3 8	 2 5 1 3 8 No swap 2 1 5 3 8 No swap 2 1 5 3 8 swap 2 1 3 5 8 No swap 2 1 3 5 8

Pass 3: Take the array that came output from the step2 and do the same process	Pass 4: Take the array that came output from the step3 and do the same process
 2 1 3 5 8 swap 1 2 3 5 8 No swap 1 2 3 5 8 No swap 1 2 3 5 8 No swap 1 2 3 5 8	 1 2 3 5 8 No swap 1 2 3 5 8 No swap 1 2 3 5 8 No swap 1 2 3 5 8 No swap 1 2 3 5 8

So the final Output is

1	**2**	**3**	**5**	**8**

In this sorting technique , if we have ‘n’ elements then we get ‘n-1’ steps.

2.Trace the Bubble Sort

Provided the unsorted array ` [7, 4, 2, 9, 1] ` and trace the bubble sort algorithm step by step, showing the changes in the array after each pass.

Given array: [7, 4, 2, 9, 1]

Pass 1:

- Compare 7 and 4.
  - swap them.
  - Array →[4, 7, 2, 9, 1].
- Compare 7 and 2.
  - Swap them.
  - Array → [4, 2, 7, 9, 1].
- Compare 7 and 9.
  - No swap needed.
  - Array→ [4, 2, 7, 9, 1].
- Compare 9 and 1.
  - Swap them.
  - Array → [4, 2, 7, 1, 9].
- After Pass 1: **[4, 2, 7, 1, 9]**

Pass 2:

- Compare 4 and 2.
  - Swap them.
  - Array → [2, 4, 7, 1, 9].
- Compare 4 and 7.
  - No swap needed.
  - Array→ [2, 4, 7, 1, 9].
- Compare 7 and 1.
  - Swap them.
  - Array →[2, 4, 1, 7, 9].

- Compare 7 and 9.
    - No swap needed.
    - Array → [2, 4, 1, 7, 9].
- After Pass 2: **[2, 4, 1, 7, 9]**

Pass 3:

- Compare 2 and 4.
    - No swap needed.
    - Array → [2, 4, 1, 7, 9].
- Compare 4 and 1.
    - Swap them.
    - Array →[2, 1, 4, 7, 9].
- Compare 4 and 7 .
    - No swap needed.
    - Array→ [2, 1, 4, 7, 9].
- Compare 7 and 9.
    - No swap needed.
    - Array → [2, 1, 4, 7, 9].
- After Pass 3: **[2, 1, 4, 7, 9]**

Pass 4:

- Compare 2 and 1.
    - Swap them.
    - Array→[1, 2, 4, 7, 9].
- Compare 2 and 4.
    - No swap needed.
    - Array →[1, 2, 4, 7, 9].
- Compare 4 and 7.
    - No swap needed.
    - Array →[1, 2, 4, 7, 9].
- Compare 7 and 9.
    - No swap needed.
    - Array →[1, 2, 4, 7, 9].
- After Pass 4: **[1, 2, 4, 7, 9]**

Now, the array is sorted, and no more passes are needed. The final sorted array is [1, 2, 4, 7, 9].

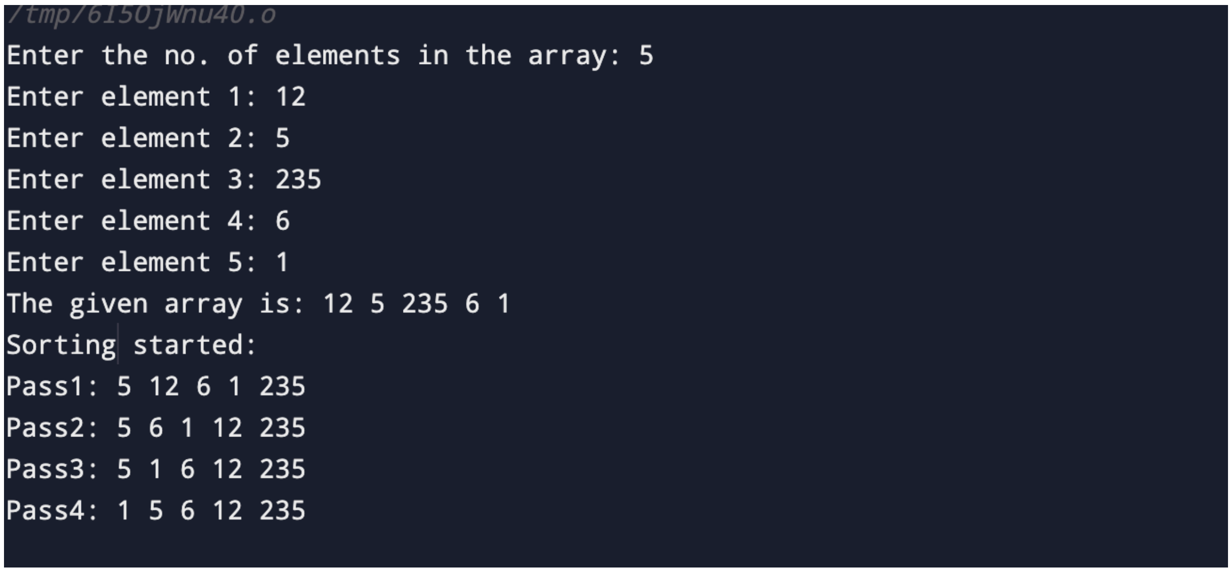
3.Code Implementation
Implement the bubble sort algorithm in C++. Provide them with the following unsorted array: ` [3, 6, 1, 8, 2] `. Code from scratch and test it to ensure it works correctly.

<u>CODE FOR BUBBLE SORT:</u>

```
#include <iostream>
using namespace std;
void print_array(int a[],int n){
    for (int i = 0; i < n; i++) {
    cout << a[i] << " ";
  }
}
void bubblesort(int a[],int n)
{
  int c;
  for (int i=0;i<n-1;i++){
    for(int j=0;j<n-i-1;j++)
    {
      if(a[j]>a[j+1])
      {
        c=a[j];
        a[j]=a[j+1];
        a[j+1]=c;
      }
    }
    cout<<"\nPass"<<i+1<<": ";
    print_array(a,n);
  }
}
int main()
{
  const int maxsize = 100;
  int a[maxsize]; // declaring an array
  int n;
  cout << "Enter the no. of elements in the array: ";
  cin >> n;
  // taking the elements dynamically into the array
```

```
    for (int i = 0; i < n; i++) {
        cout << "Enter element " << i + 1 << ": ";
        cin >> a[i];
    }
    cout << "The given array is: ";
    print_array(a,n);
    cout<<"\nSorting started:";
    bubblesort(a,n);
    return 0;
}
```

OUTPUT:



```
/tmp/615OjWnu40.o
Enter the no. of elements in the array: 5
Enter element 1: 12
Enter element 2: 5
Enter element 3: 235
Enter element 4: 6
Enter element 5: 1
The given array is: 12 5 235 6 1
Sorting started:
Pass1: 5 12 6 1 235
Pass2: 5 6 1 12 235
Pass3: 5 1 6 12 235
Pass4: 1 5 6 12 235
```

4.Advanced: Optimization Challenge.
Challenge yourself to optimize the bubble sort algorithm. Provided with the partially sorted array ` [1, 2, 3, 4, 5, 10, 9, 8, 7, 6] `. Optimize the algorithm to reduce the number of comparisons or swaps, making the sorting process more efficient.

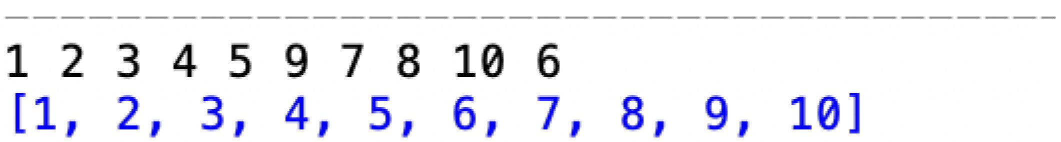
<u>CODE:</u>
```
def is_sort(a):
   for i in range(len(a)-1):
      if(a[i]>a[i+1]):
         return False
      else:
         return True

def bubble_sort(a,c):
   n=len(a)
   for i in range(n):
      for j in range(0,n-i-1):
         if(a[j]>a[j+1]):
            a[j],a[j+1]=a[j+1],a[j]
   print(a+c)

a=list(map(int,input().split()))
b=0
c=[]
#checking how many elemnts in an array is sorted
for i in range(len(a)):
   if(len(a)<2):
      if(a[0]<a[1]):
         c.append(a[0])
         c.append(a[1])
      else:
         bubble_sort(a,c)
         break
   if(len(a)>2):#by seeing the array is sorted or nit we decreassed no of iterations in main function
      if(is_sort(a[0:i])):
         #print("issort")
         c.append(i)
```

```
else:
   bubble_sort(a[i::],c)
   break
```

OUTPUT:



```
1 2 3 4 5 9 7 8 10 6
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

5. Advanced: Comparison with Other Sorting Algorithms
Compare the bubble sort algorithm with quicksort and mergesort. Discuss the advantages and disadvantages of bubble sort in different scenarios. Additionally, analyze when it might be preferable to use other sorting algorithms.

Bubble Sort	**Quick Sort**	**Merge Sort**
● Compare with neighboring element and performs swap accordingly. ● No.fo comparisons: n*(n-1)/2 ● For largest arrays it is not efficient ● Bubble sort is easy to perform ● Bubble sort is the iterative approach	● Compare the whole array with a fixed pivot element and swaps accordingly ● No.of comparisons: n*log(n) in best case n^2 in wort case ● For largest arrays it is efficient ● Quick sort is little complex compare to quick sort ● Quick sort is Recursive approach	● It follows divide and conquer rule for sorting ● No.of comparisons: nlog(n) ● It is suitable for larger datasets ● Time complexity of merge sort: O(n logn) ● Merge sort is a recursive approach

ADVANTAGES OF BUBBLE SORT:

1.It is easy to implement and easy to perform
2. Suitable for smaller data sets
3. While sorting it does not needed the additional memory
4.As bubble sort is easy to implement, it is easily integrated in a code.
5. It is a stable sorting algorithm

DIS ADVANTAGES OF BUBBLE SORT:

1. Not suitable for the larger data sets as its time complexity is in quadratic nature.
2. It performs poorly for the random and unpredictable ordered arrays.
3. It is not more efficient than quick and merge sort as it is not suitable for the arrays which are having unpredictable order.
4. We cannot use bubble sort for the real world applications.
5. Even though bubble sort is stable, when optimizations applied it is not.

Scenarios to prefers other sorting techniques rather than bubble sort:

1. Whenever there are larger data sets prefer to use quick and merge sorts
2. Whenever there are random or unpredictable arrays to sort then it prefres to use quick and merge sorts rather than bubble sort
3. When we need to perform the sort quickly and optimized it prefers to use quick sort.
4. For real world applications it is better not to use bubble sort .Instead use quick or merge sorts.
5. Whenever the additional memory is not a constraint, then we can use quick or merge sort rather than bubble sort
6. In the data sets where we are having predictable performance , there we can make use of ,merge sort as it is more efficient.