AOA Lab 1

Name: Prakash P. Biswas
Roll No: 9947
Branch: SE Comps B
Batch: A

Bubble Sort:

Code:

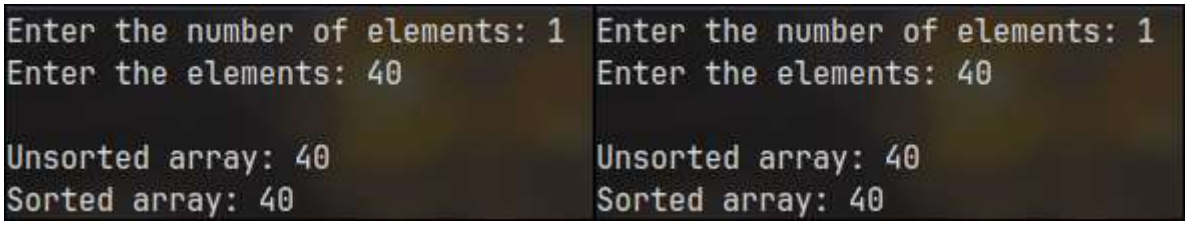
```cpp
#include <iostream>

// swaps a and b
void swap(int &a, int &b) {
 // here a and b do not contain addresses but the references to the original paased values c doesn't have references like cpp so we use pointers there
 int temp = a;
 a = b;
 b = temp;
}

// uses bubble sort to bubble up the lowest element to start
void bubbleSort(int arr[], int n) {
 for (int i = 0; i < n - 1; ++i) { //n
   for (int j = i + 1; j < n; ++j) {
     if (arr[j] < arr[i]) {
       swap(arr[j], arr[i]);
     }
   }
 }
}


// Takes user input for the given array
void getArray(int arr[], int n){
 std::cout << "Enter the elements: ";
 for (int i = 0; i < n; ++i) {
```

```
    std::cin >> arr[i];
  }
}

// Prints the elements of the array to the standard out
void printArray(int arr[], int n){
  for (int i = 0; i < n; ++i) {
    std::cout << arr[i] << " ";
  }
}

int main() {
  int n;
  std::cout << "Enter the number of elements: ";
  std::cin >> n;
  int arr[n];

  getArray(arr, n);

  std::cout << "Unsorted array: ";
  printArray(arr, n);

  bubbleSort(arr, n);

  std::cout << "Sorted array: ";
  printArray(arr, n);

  return 0;
}
```

OUTPUT:

```
Enter the number of elements: 5
Enter the elements: 1 9 3 5 3

Unsorted array: 1 9 3 5 3
Sorted array: 1 3 3 5 9
```

```
Enter the number of elements: 5
Enter the elements: 1 5 2 -4 2

Unsorted array: 1 5 2 -4 2
Sorted array: -4 1 2 2 5
```



COMPLEXITY(worst case):
for first iteration:
No. of comparisons: n-1
No. of swaps: n -1
for second iteration:
No. of comparisons: n-2
No. of swaps: n -1
for last(n-1) iteration:
No. of comparison: 1
No. of swaps: 1

$T(n) = (n - 1) + (n - 2) + (n - 3) \ldots\ldots + 2 + 1$

$= \frac{(n-1)(n-2)}{2}$

$= \frac{(N)(N-1)}{2}$

$T(n) = O(n^2)$

Modified Bubble Sort:

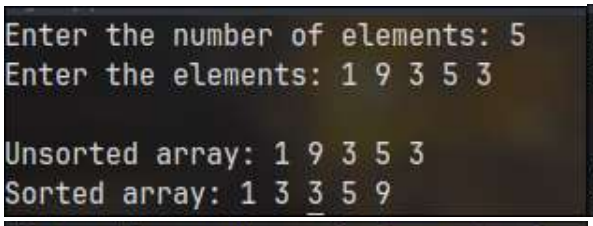
Code:

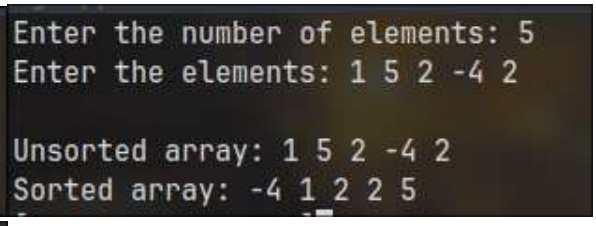
```
#include <iostream>

// swaps a and b
void swap(int &a, int &b) {
 // here a and b do not contain addresses but the references to the original
 // paased values c doesn't have references like cpp so we use pointers there
 int temp = a;
 a = b;
 b = temp;
}
```

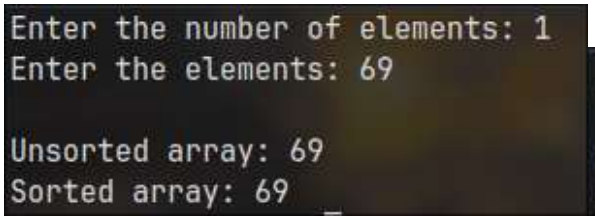
```cpp
// uses modified bubble sort to bubble up the highest element to
end
void modBubbleSort(int arr[], int n) {
 for (int i = 0; i < n - 1; i++) {
   int count = 0;
   for (int j = 0; j < n - i - 1; j++) {
     if (arr[j + 1] < arr[j]) {
       swap(arr[j + 1], arr[j]);
       count = 1;
     }
   }
   if (count == 0)
     break;
 }
}

// Takes user input for the given array
void getArray(int arr[], int n) {
 std::cout << "Enter the elements: ";
 for (int i = 0; i < n; ++i) {
   std::cin >> arr[i];
 }
}

// Prints the elements of the array to the standard out
void printArray(int arr[], int n) {
 for (int i = 0; i < n; ++i) {
   std::cout << arr[i] << " ";
 }
}

int main() {
 int n;
 std::cout << "Enter the number of elements: ";
 std::cin >> n;
```

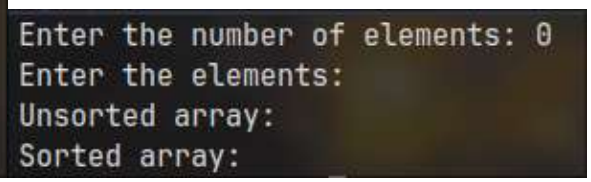
```
 int arr[n];

 getArray(arr, n);

 std::cout << "\nUnsorted array: ";
 printArray(arr, n);

 modBubbleSort(arr, n);

 std::cout << "\nSorted array: ";
 printArray(arr, n);

 return 0;
}
```

Output:









COMPLEXITY(worst case):
for first iteration:
No. of comparisons: n-1
No. of swaps: n -1
for second iteration:
No. of comparisons: n-2
No. of swaps: n -1
for last(n-1) iteration:
No. of comparison: 1
No. of swaps: 1

T(n) = (n - 1) + (n -2) + (n - 3) ….. + 2 + 1
= $\underline{(n-1)(n-2)}$

$$= \frac{(N)(N-1)}{2}$$

$$T(n) = O(n^2)$$

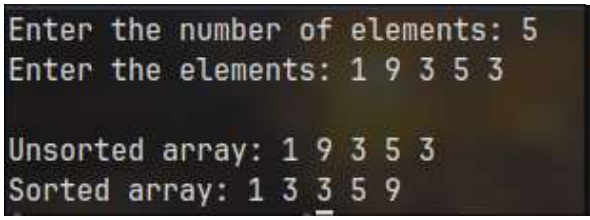
Selection Sort:

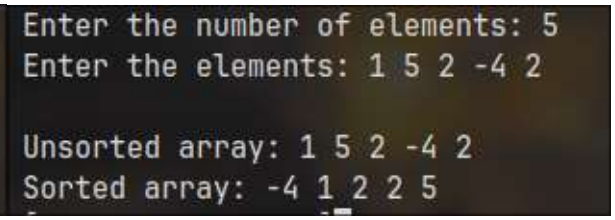
CODE:

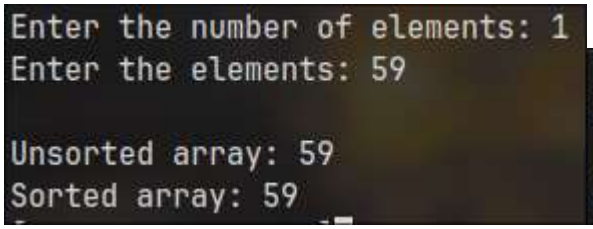
```
#include <iostream>

// swaps a and b
void swap(int &a, int &b) {
 // here a and b do not contain addresses but the references to
the original
 // paased values c doesn't have references like cpp so we use
pointers there
 int temp = a;
 a = b;
 b = temp;
}

// Selection sort to select the smallest element and place it at
the beginning
void selectionSort(int arr[], int n) {
 for (int i = 0; i < n - 1; ++i) {
   int minIndex = i;
   for (int j = i + 1; j < n; ++j) {
     if (arr[j] < arr[minIndex]) {
       minIndex = j;
     }
   }
   swap(arr[i], arr[minIndex]);
 }
}

// Takes user input for the given array
void getArray(int arr[], int n) {
```

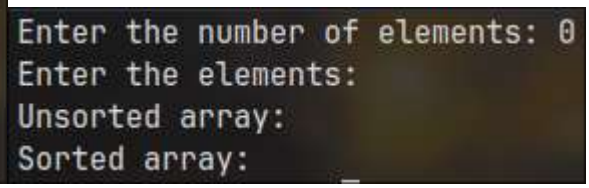
```
 std::cout << "Enter the elements: ";
 for (int i = 0; i < n; ++i) {
   std::cin >> arr[i];
 }
}

// Prints the elements of the array to the standard out
void printArray(int arr[], int n) {
 for (int i = 0; i < n; ++i) {
   std::cout << arr[i] << " ";
 }
}

int main() {
 int n;
 std::cout << "Enter the number of elements: ";
 std::cin >> n;
 int arr[n];

 getArray(arr, n);

 std::cout << "\nUnsorted array: ";
 printArray(arr, n);

 selectionSort(arr, n);

 std::cout << "\nSorted array: ";
 printArray(arr, n);

 return 0;
}
```

OUTPUT:









COMPLEXITY(worst case):
for first iteration:
No. of comparisons: n-1
for second iteration:
No. of comparisons: n-2
for last(n-1) iteration:
No. of comparison: 1

$T(n) = (n - 1) + (n - 2) + (n - 3) \ldots\ldots + 2 + 1$

$= \frac{(n-1)(n-2)}{2}$

$= \frac{(N)(N-1)}{2}$

$T(n) = O(n^2)$

Insertion Sort:
CODE:

```
#include <iostream>

// swaps a and b
void swap(int &a, int &b) {
 // here a and b do not contain addresses but the references to the original
 // paased values c doesn't have references like cpp so we use pointers there
 int temp = a;
 a = b;
```

```
 b = temp;
}

// Insertion sort to insert elements in their correct positions
void insertionSort(int arr[], int n) {
 for (int i = 1; i < n; ++i) {
   int key = arr[i];
   int j = i - 1;

   // Move elements greater than key to one position ahead
   while (j >= 0 && arr[j] > key) {
     arr[j + 1] = arr[j];
     --j;
   }

   // Insert key in its correct position
   arr[j + 1] = key;
 }
}

// Takes user input for the given array
void getArray(int arr[], int n) {
 std::cout << "Enter the elements: ";
 for (int i = 0; i < n; ++i) {
   std::cin >> arr[i];
 }
}

// Prints the elements of the array to the standard out
void printArray(int arr[], int n) {
 for (int i = 0; i < n; ++i) {
   std::cout << arr[i] << " ";
 }
}
```

```cpp
int main() {
 int n;
 std::cout << "Enter the number of elements: ";
 std::cin >> n;
 int arr[n];

 getArray(arr, n);

 std::cout << "\nUnsorted array: ";
 printArray(arr, n);

 insertionSort(arr, n);

 std::cout << "\nSorted array: ";
 printArray(arr, n);

 return 0;
}
```

OUTPUT:

```
Enter the number of elements: 5
Enter the elements: 1 3 2 5 4

Unsorted array: 1 3 2 5 4
Sorted array: 1 2 3 4 5
```

```
Enter the number of elements: 5
Enter the elements: 1 2 -4 2 9

Unsorted array: 1 2 -4 2 9
Sorted array: -4 1 2 2 9
```

```
Enter the number of elements: 1
Enter the elements: 39

Unsorted array: 39
Sorted array: 39
```

```
Enter the number of elements: 0
Enter the elements:
Unsorted array:
Sorted array:
```

COMPLEXITY(worst case):

Total no of comparison = 1 + 2 + 3 + 4 + 5

= 1 + 2 + 3 + 4 … + (n -1) ——-------(Generalising the above equation)

$= \frac{n\,(n-1)}{2}$

$T(n) = O(n^2)$

Counting Sort:

CODE:

```
#include <iostream>

// Counting sort to sort elements based on their count
int getMax(int arr[], int n) {
 int max = arr[0];
 for (int i = 1; i < n; i++) {
   if (arr[i] > max)
     max = arr[i];
 }
 return max; // maximum element from the array
}

// sorts the array based on the count of each element
void countingSort(int arr[], int n) {
 int output[n + 1];
 int max = getMax(arr, n);
 int count[max + 1]; // create count array with size [max+1]

 for (int i = 0; i <= max; ++i) {
   count[i] = 0; // Initialize count array with all zeros
 }

 for (int i = 0; i < n; i++) // Store the count of each element
 {
   count[arr[i]]++;
 }

 for (int i = 1; i <= max; i++)
   count[i] += count[i - 1]; // find cumulative frequency

 /* This loop will find the index of each element of the original
array in
```

```cpp
  count array, and place the elements in output array*/
 for (int i = n - 1; i >= 0; i--) {
   output[count[arr[i]] - 1] = arr[i];
   count[arr[i]]--; // decrease count for same numbers
 }

 for (int i = 0; i < n; i++) {
   arr[i] = output[i]; // store the sorted elements into main
array
 }
}

// Takes user input for the given array
void getArray(int arr[], int n) {
 std::cout << "Enter the elements: ";
 for (int i = 0; i < n; ++i) {
   std::cin >> arr[i];
 }
}

// Prints the elements of the array to the standard out
void printArray(int arr[], int n) {
 for (int i = 0; i < n; ++i) {
   std::cout << arr[i] << " ";
 }
}

int main() {
 int n;
 std::cout << "Enter the number of elements: ";
 std::cin >> n;
 int arr[n];

 getArray(arr, n);
```

```
 std::cout << "\nUnsorted array: ";
 printArray(arr, n);

 countingSort(arr, n);

 std::cout << "\nSorted array: ";
 printArray(arr, n);

 return 0;
}
```

Complexity(worst case):
Function getMax = O(n)
Function countingSort = O(n)
Function getArray = O(n)
Function printArray = O(n)

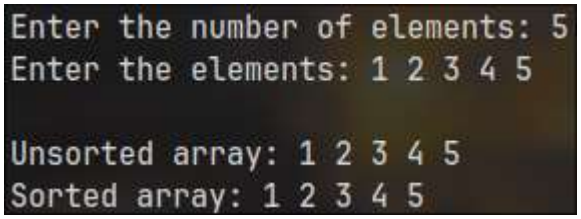
Therefore the time complexity is O(n)
However if we take into consideration the time taken to create an array let say k, The time time complexity will increase as k is much larger in worst case hence we can't ignore it
Therefore the time complexity is o(n + k)

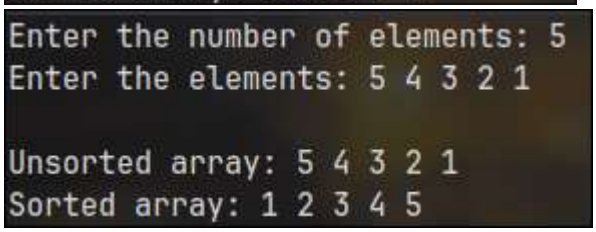
OUTPUT:

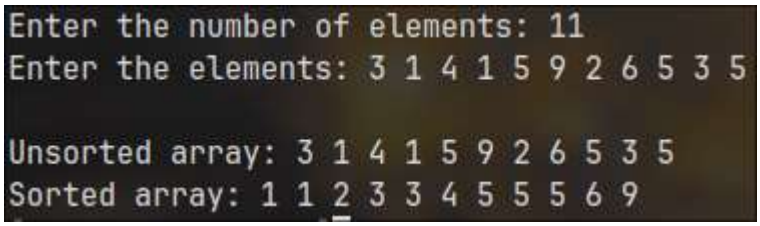
```
Enter the number of elements: 10
Enter the elements: 1 9 0 23 9 5 9 2 1 0

Unsorted array: 1 9 0 23 9 5 9 2 1 0
Sorted array: 0 0 1 1 2 5 9 9 9 23
```

```
Enter the number of elements: 4
Enter the elements: 4 2 3 1

Unsorted array: 4 2 3 1
Sorted array: 1 2 3 4
```

```
Enter the number of elements: 4
Enter the elements: 5 5 5 5

Unsorted array: 5 5 5 5
Sorted array: 5 5 5 5
```


Enter the number of elements: 5
Enter the elements: 1 2 3 4 5
Unsorted array: 1 2 3 4 5
Sorted array: 1 2 3 4 5


Enter the number of elements: 5
Enter the elements: 5 4 3 2 1
Unsorted array: 5 4 3 2 1
Sorted array: 1 2 3 4 5


Enter the number of elements: 11
Enter the elements: 3 1 4 1 5 9 2 6 5 3 5
Unsorted array: 3 1 4 1 5 9 2 6 5 3 5
Sorted array: 1 1 2 3 3 4 5 5 5 6 9