# Department of Computing

**Name: Mahum Samar**

**CMS ID : 290647**

**CS250: Data Structure and Algorithms**

**Class: BSCS 9B**

# lab 09

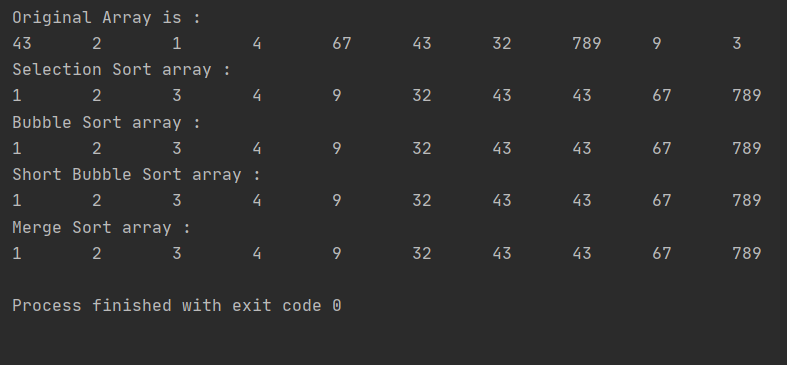
# Task 01

Implement Selection sort, Bubble sort, Short Bubble sort and Merge sort algorithms in C++.

## Code:

|  |
| --- |
| #include <iostream> #include <cmath>   using namespace std;   class Sorting { public:  void Swap(int \*x, int \*y)  {  *//method to swap the two values of the array by pointing to the address of the array index* int temp = \*x;  \*x = \*y;  \*y = temp;  }   void PrintArray(int array[], int n)  {  *//method ot print the contents of the array on the screen.* for(int i = 0 ; i < n ; i++)  {  cout << array[i] << "\t";  }  }   void SelectionSort(int array[], int n)  {  *//method for selection sort algorithm implementation.* int current; *//variable to store the current index to which is pointed and being compared with other values.* int i; *//to store the index while looping through the array.* int minIndex; *//to store the min index   //loop until n-1 because last value will be automatically sorted.* for(current = 0 ; current < n-1 ; current++)  {  *//loop for swapping the array element to sort it.* minIndex = current;  for(i = current +1 ; i < n ; i++)  {  if(array[i] < array[minIndex])  {  *//condition that checks the order.* minIndex = i;  }  }   *//swap method called.* Swap(&array[minIndex], &array[current]);  *//PrintArray(array, n);* }  }   void ShortBubbleSort(int array[], int n)  {  int i, current;  bool valSwap; *//variable to end the loop if the array is already sorted.* for(current = 0 ; current < n-1 ; current++)  {  *//loop to check the sorting of the array* valSwap = false;  *//cout << "Current : " << current << endl;* for(i = 0 ; i < n-1 ; i++)  {  *//values are swapped if needed. and valSwap is true so that indicating that the value has been swapped and loop needs to continue again.* if(array[i] > array[i+1])  {  Swap(&array[i], &array[i+1]);  valSwap = true;  }  *//PrintArray(array, n);* }  *//cout << endl;  //if the values are not swapped in hte iteration of the array so the list has become sorted and not needed to continue the loop.* if(!valSwap)  {  return;  }  }  }   void BubbleSort(int array[], int n)  {  *//cout << "Size of input list is : " << n << endl;* int i, current;  for(current = 0 ; current < n-1 ; current++)  {  *//loop to swap the values.  //cout << "Current : " << current << endl;* for(i =0 ; i < n-current-1 ; i++)  {  *//cout << "i : " << i << "\t";* if(array[i] > array[i+1])  {  *//value is swapped whenever the condition becomes true. and check for each consecutive index.* Swap(&array[i], &array[i+1]);  }  }  }  }   void Merge(int arr[], int leftFirst, int leftLast, int rightFirst, int rightLast)  {  *//merge method used in mergeSort. after the recursion call ends the lists are combined in which this is used.* int saveFirst = leftFirst; *//left first is saved so that when the contents of the temp array are transferred to the original array it can be used.* int index = leftFirst; *//to keep track which part of the temp array is to be used for storing the sorted parts of the sublist  //int temp[5];* int temp[rightLast+1]; *//creating temp array according to the size that is needed.* while(leftFirst <= leftLast && rightFirst <= rightLast)  {  *//loop that compares the values in the sublist that are to be arranged.* if(arr[leftFirst] < arr[rightFirst])  {  *//if left first is smaller.* temp[index] = arr[leftFirst];  leftFirst++;  }  else  {  *//if right first is smaller* temp[index] = arr[rightFirst];  rightFirst++;  }  index++;  }   while(leftFirst <= leftLast)  {  *//to store the remaining elements of the left list in the temp after there are no more elements in the right list to be compared.* temp[index] = arr[leftFirst];  leftFirst++;  index++;  }   while(rightFirst <= rightLast)  {  *//to store the remaining elements of the right list in the temp after there are no more elements in the left list to be compared.* temp[index] = arr[rightFirst];  rightFirst++;  index++;  }   while(saveFirst <= rightLast)  {  *//transfering the contents from the temp array into the ariginal array.* arr[saveFirst] = temp[saveFirst];  saveFirst++;  }   }   void MergeSort(int arr[], int leftFirst, int rightLast)  {  if(leftFirst < rightLast)  {  *//base case  //cout << leftFirst << "\t" << rightLast << "\n";  //ceil function applied.* int middle = ceil((leftFirst + rightLast -1 )/2);  *//recursive call* MergeSort(arr, leftFirst, middle);  MergeSort(arr, middle + 1, rightLast);  *//merging the arrays.* Merge(arr, leftFirst, middle, middle + 1, rightLast);  }  *//PrintArray(arr, 5);* }   void RandomArrayGenerator(int arr[], int n)  {  *//method to generate random array* for(int i = 0 ; i < n ; i++)  {  arr[i] = (rand() % 1000) + 1;  }  }   void CopyArray(int arr1[], int arr2[], int n)  {  *//method to coy the contents of one array into the other.* for(int i = 0 ; i < n ; i++)  {  arr2[i] = arr1[i];  }  }  };    int main() {  Sorting \*sorting = new Sorting();  int array[10] = {43,2,1,4,67,43,32,789,9,3};  cout << "Original Array is :\n";  sorting -> PrintArray(array, 10);  cout << endl;   int array2[10];   sorting -> CopyArray(array,array2,10);  sorting -> SelectionSort(array2,10);  cout << "Selection Sort array : \n";  sorting -> PrintArray(array2, 10);  cout << endl;   sorting -> CopyArray(array,array2,10);  sorting -> BubbleSort(array2,10);  cout << "Bubble Sort array : \n";  sorting -> PrintArray(array2, 10);  cout << endl;   sorting -> CopyArray(array,array2,10);  sorting -> ShortBubbleSort(array2,10);  cout << "Short Bubble Sort array : \n";  sorting -> PrintArray(array2, 10);  cout << endl;   sorting -> CopyArray(array,array2,10);  sorting -> MergeSort(array2,0,9);  cout << "Merge Sort array : \n";  sorting -> PrintArray(array2, 10);  cout << endl;  return 0; } |

## Output:



# Task 02

The next step is to compare the running time of algorithms. Generate arrays of random numbers in the range 1 to 1000 with sizes 100, 1000, and 5000. Compare the running times of the four algorithms on each array. How do they compare? Are the results what you expected, and why? Answer the questions in at the end of the word file.

## Code:

|  |
| --- |
| *//declaring the random arrays* int RandomArray100[100]; int RandomArray1000[1000]; int RandomArray5000[5000];  Sorting \*sorting = new Sorting(); *//initializing th earrays with the random numbers using rand() function.* sorting -> RandomArrayGenerator(RandomArray100, 100); sorting -> RandomArrayGenerator(RandomArray1000, 1000); sorting -> RandomArrayGenerator(RandomArray5000, 5000);  *//selection sort* cout << "selection sort\n";  *//copy the orginal array into the separate array so that the original array is not sorted and it can be utilized for the other algorithms.* sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100); sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000); sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);  auto start = high\_resolution\_clock::now(); sorting -> SelectionSort(SortedRandomArray100, 100); auto stop = high\_resolution\_clock::now(); auto duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> SelectionSort(SortedRandomArray1000, 1000); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> SelectionSort(SortedRandomArray5000, 5000); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;  cout << "bubble sort\n";  sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100); sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000); sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);  start = high\_resolution\_clock::now(); sorting -> BubbleSort(SortedRandomArray100, 100); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> BubbleSort(SortedRandomArray1000, 1000); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> BubbleSort(SortedRandomArray5000, 5000); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;  cout << "short bubble sort\n";  sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100); sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000); sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);  *//* start = high\_resolution\_clock::now(); sorting -> ShortBubbleSort(SortedRandomArray100, 100); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> ShortBubbleSort(SortedRandomArray1000, 1000); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> ShortBubbleSort(SortedRandomArray5000, 5000); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;  cout << "merge sort\n"; sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100); sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000); sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);  *//* start = high\_resolution\_clock::now(); sorting -> MergeSort(SortedRandomArray100, 0, 99); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> MergeSort(SortedRandomArray1000, 0, 999); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;  start = high\_resolution\_clock::now(); sorting -> MergeSort(SortedRandomArray5000, 0, 4999); stop = high\_resolution\_clock::now(); duration = duration\_cast<microseconds>(stop - start); cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;   return 0; |

## Output:

Text

Description automatically generated

## explaination:

Selection Sort:

As the input size increases the time to sort the array also increases exponentially.

Bubble Sort:

As the input size increases the time to sort the array also increases exponentially.

Short Bubble Sort:

As the input size increases the time to sort the array also increases exponentially.

Merge sort:

As the input size increases the time to sort the array also increases but with the slow rate.

So, all the results are in accordance with the expected results. Because the array was randomly arranged and the algorithms have to sort them.

# Task 03

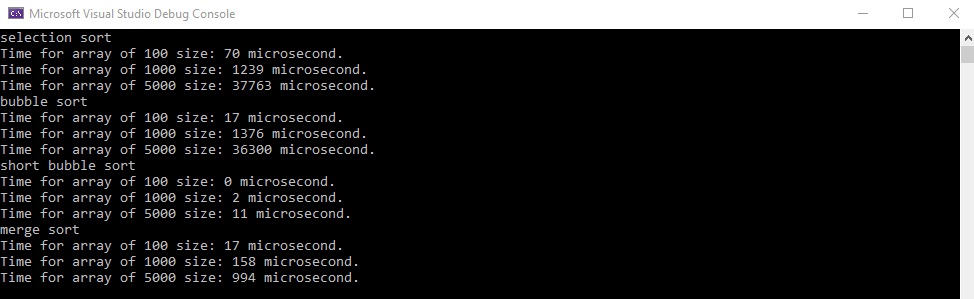
Now sort the arrays using stl::sort, once in ascending order and then in descending order. Given both sorted arrays as inputs to all the four algorithms and compute their running time. The running time of which algorithm shows most variations based on the structure of the input and why? Answer the questions in at the end of the word file.

## Code:

Ascending:

|  |
| --- |
| int main() {  *//declaring the random arrays* int RandomArray100[100];  int RandomArray1000[1000];  int RandomArray5000[5000];   Sorting \*sorting = new Sorting();  *//initializing th earrays with the random numbers using rand() function.* sorting -> RandomArrayGenerator(RandomArray100, 100);  sorting -> RandomArrayGenerator(RandomArray1000, 1000);  sorting -> RandomArrayGenerator(RandomArray5000, 5000);   sort(RandomArray100, RandomArray100 + 100);  sort(RandomArray1000, RandomArray100 + 1000);  sort(RandomArray5000, RandomArray5000 + 5000);   *//selection sort* cout << "selection sort\n";   *//copy the orginal array into the separate array so that the original array is not sorted and it can be utilized for the other algorithms.* sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   auto start = high\_resolution\_clock::now();  sorting -> SelectionSort(SortedRandomArray100, 100);  auto stop = high\_resolution\_clock::now();  auto duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> SelectionSort(SortedRandomArray1000, 1000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> SelectionSort(SortedRandomArray5000, 5000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;   cout << "bubble sort\n";   sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   start = high\_resolution\_clock::now();  sorting -> BubbleSort(SortedRandomArray100, 100);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> BubbleSort(SortedRandomArray1000, 1000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> BubbleSort(SortedRandomArray5000, 5000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;   cout << "short bubble sort\n";   sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   *//* start = high\_resolution\_clock::now();  sorting -> ShortBubbleSort(SortedRandomArray100, 100);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> ShortBubbleSort(SortedRandomArray1000, 1000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> ShortBubbleSort(SortedRandomArray5000, 5000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;   cout << "merge sort\n";  sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   *//* start = high\_resolution\_clock::now();  sorting -> MergeSort(SortedRandomArray100, 0, 99);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> MergeSort(SortedRandomArray1000, 0, 999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> MergeSort(SortedRandomArray5000, 0, 4999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;    return 0; } |

## Output:

Ascending:  


## explanation:

Selection Sort:

As the input size increases the time to sort the array also increases exponentially because there is no way to stop the selection sort algorithm if the list is already sorted.

Bubble Sort:

As the input size increases the time to sort the array also increases but not with the exponential rate this time because the array is sorted.

Short Bubble Sort:

As the input size increases the time to sort the array does not increases because if the array is found to be sorted the algorithm stops executing and returns.

Merge sort:

As the input size increases the time to sort the array also increases but with the slow rate.

So, all the results are in accordance with the expected results. Because the array is sorted in ascending order and the some algorithms have not to sort them like short bubble sort.

## Code:

Descending:

|  |
| --- |
| int main() {  *//declaring the random arrays* int RandomArray100[100];  int RandomArray1000[1000];  int RandomArray5000[5000];   Sorting \*sorting = new Sorting();  *//initializing th earrays with the random numbers using rand() function.* sorting -> RandomArrayGenerator(RandomArray100, 100);  sorting -> RandomArrayGenerator(RandomArray1000, 1000);  sorting -> RandomArrayGenerator(RandomArray5000, 5000);   sort(RandomArray100, RandomArray100 + 100, greater<int>());  sort(RandomArray1000, RandomArray1000 + 1000, greater<int>());  sort(RandomArray5000, RandomArray5000 + 5000, greater<int>());   *//selection sort* cout << "selection sort\n";   *//copy the orginal array into the separate array so that the original array is not sorted and it can be utilized for the other algorithms.* sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   auto start = high\_resolution\_clock::now();  sorting -> SelectionSort(SortedRandomArray100, 100);  auto stop = high\_resolution\_clock::now();  auto duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> SelectionSort(SortedRandomArray1000, 1000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> SelectionSort(SortedRandomArray5000, 5000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;   cout << "bubble sort\n";   sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   start = high\_resolution\_clock::now();  sorting -> BubbleSort(SortedRandomArray100, 100);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> BubbleSort(SortedRandomArray1000, 1000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> BubbleSort(SortedRandomArray5000, 5000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;   cout << "short bubble sort\n";   sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   *//* start = high\_resolution\_clock::now();  sorting -> ShortBubbleSort(SortedRandomArray100, 100);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> ShortBubbleSort(SortedRandomArray1000, 1000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> ShortBubbleSort(SortedRandomArray5000, 5000);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;   cout << "merge sort\n";  sorting -> CopyArray(RandomArray100, SortedRandomArray100, 100);  sorting -> CopyArray(RandomArray1000, SortedRandomArray1000, 1000);  sorting -> CopyArray(RandomArray5000, SortedRandomArray5000, 5000);   *//* start = high\_resolution\_clock::now();  sorting -> MergeSort(SortedRandomArray100, 0, 99);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> MergeSort(SortedRandomArray1000, 0, 999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond."<< endl;   start = high\_resolution\_clock::now();  sorting -> MergeSort(SortedRandomArray5000, 0, 4999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 5000 size: " << duration.count() << " microsecond."<< endl;    return 0; } |

## Output:

Descending:

Text

Description automatically generated

## explaination:

Selection Sort:

As the input size increases the time to sort the array also increases exponentially because the array is not sorted.

Bubble Sort:

As the input size increases the time to sort the array also increases exponentially.

Short Bubble Sort:

As the input size increases the time to sort the array also increases exponentially.

Merge sort:

As the input size increases the time to sort the array also increases but with the slow rate.

So, all the results are in accordance with the expected results. Because the array was in descending order and the algorithms have to sort them.