# Department of Computing

**Name: Mahum Samar**

**CMS ID: 290647**

**CS250: Data Structure and Algorithms**

**Class: BSCS 9B**

# lab 10

# Task 01

## Code:

|  |
| --- |
| #include <iostream> #include <chrono> #include <algorithm> using namespace std; using namespace std::chrono;  class Sorting { public:   int count = 0;   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 RandomArrayGenerator(int arr[], int n)  {  *//method to generate random array* for(int i = 0 ; i < n ; i++)  {  arr[i] = (rand() % 1000) + 1;  }  }   int partition(int a[], int left, int right)  {  *///method to sort the array portion that have been partitioned.  //it sort by taking the pivot as left index value.* int pivot = left;  while (left <= right)  {  while (a[left] <= a[pivot] && left <= right)  {  *//while left value is less then the pivot value left index is incremented.* left++;  }  while (a[right] > a[pivot] && left <= right)  {  *//while right index ix greater then the pivot index value* right--;  }  if(left < right)  {  *//if left is less than the right index then the values are swapped.* Swap(&a[left], &a[right]);  }  *//right indexed value is swapped with the pivot value.* Swap(&a[right], &a[pivot]);  }  *//this functions return the right one.* return right;  }   void QuickSort(int a[], int left, int right)  {  *//quick sort using the recursion.* if(left < right)  {  *//base case while the left index is less than the right index.* int pivot = partition(a, left, right);  *//for the left half* QuickSort(a, left, pivot - 1);  *//for right half* QuickSort(a, pivot + 1, right);  }   }  };   int main() {  int RandomArray100[100];  int RandomArray1000[1000];  int RandomArray5000[5000];   Sorting \*sorting = new Sorting();  *//initializing the arrays with the random numbers using rand() function.* sorting -> RandomArrayGenerator(RandomArray100, 100);  sorting -> RandomArrayGenerator(RandomArray1000, 1000);  sorting -> RandomArrayGenerator(RandomArray5000, 5000);   *//selection sort* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t\tQuick sort\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";   auto start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray100, 0, 99);  auto stop = high\_resolution\_clock::now();  auto duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 100 size: " << duration.count() << " microsecond.\n"<< endl;   start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray1000, 0, 999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond.\n"<< endl;    start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray5000, 0, 4999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Time for array of 1000 size: " << duration.count() << " microsecond.\n"<< endl;    return 0; } |

## Output:

# Task 02

## Code:

|  |
| --- |
| #include <iostream> #include <chrono> #include <algorithm> using namespace std; using namespace std::chrono;  class Sorting { public:   int count = 0;   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 RandomArrayGenerator(int arr[], int n)  {  *//method to generate random array* for(int i = 0 ; i < n ; i++)  {  arr[i] = (rand() % 1000) + 1;  }  }   int partition(int a[], int left, int right)  {  *///method to sort the array portion that have been partitioned.  //it sort by taking the pivot as left index value.* int pivot = left;  while (left <= right)  {  while (a[left] <= a[pivot] && left <= right)  {  *//while left value is less then the pivot value left index is incremented.* left++;  }  while (a[right] > a[pivot] && left <= right)  {  *//while right index ix greater then the pivot index value* right--;  }  if(left < right)  {  *//if left is less than the right index then the values are swapped.* Swap(&a[left], &a[right]);  }  *//right indexed value is swapped with the pivot value.* Swap(&a[right], &a[pivot]);  }  *//this functions return the right one.* return right;  }   void QuickSort(int a[], int left, int right)  {  *//quick sort using the recursion.* if(left < right)  {  *//base case while the left index is less than the right index.* count++;*//variable to count the number of partitions* int pivot = partition(a, left, right);  *//for the left half* QuickSort(a, left, pivot - 1);  *//for right half* QuickSort(a, pivot + 1, right);  }   }  };   int main() {  int RandomArray100[100];  int RandomArray1000[1000];  int RandomArray5000[5000];   Sorting \*sorting = new Sorting();  *//initializing the arrays with the random numbers using rand() function.* sorting -> RandomArrayGenerator(RandomArray100, 100);  sorting -> RandomArrayGenerator(RandomArray1000, 1000);  sorting -> RandomArrayGenerator(RandomArray5000, 5000);   *//selection sort* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t\tQuick sort\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";   sorting -> count = 0;   auto start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray100, 0, 99);  auto stop = high\_resolution\_clock::now();  auto duration = duration\_cast<microseconds>(stop - start);  cout << "Number of Partitions for array of size 100 : " << sorting -> count << endl;  cout << "Time for array of 100 size: " << duration.count() << " microsecond.\n"<< endl;    sorting -> count = 0;  start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray1000, 0, 999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Number of Partitions for array of size 1000 : " << sorting -> count << endl;  cout << "Time for array of 1000 size: " << duration.count() << " microsecond.\n"<< endl;    sorting -> count = 0;  start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray5000, 0, 4999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Number of Partitions for array of size 5000 : " << sorting -> count << endl;  cout << "Time for array of 1000 size: " << duration.count() << " microsecond.\n"<< endl;    return 0; } |

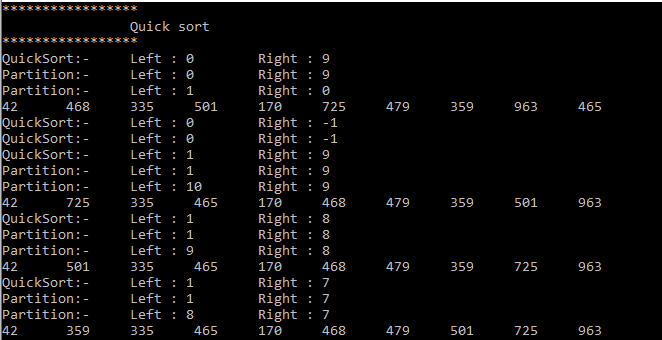
## Output:

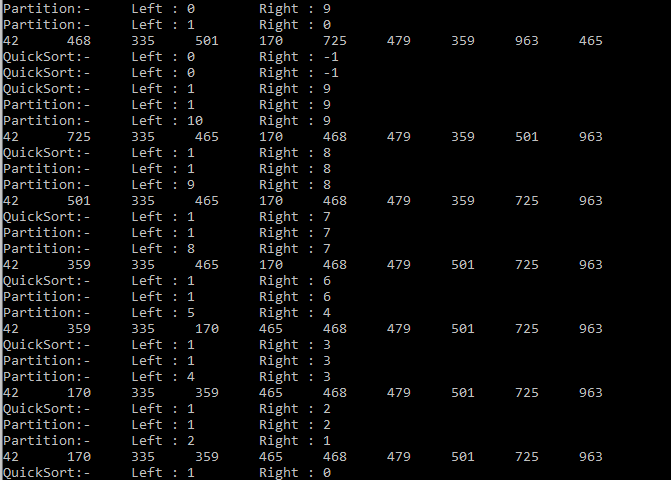
# Task 03

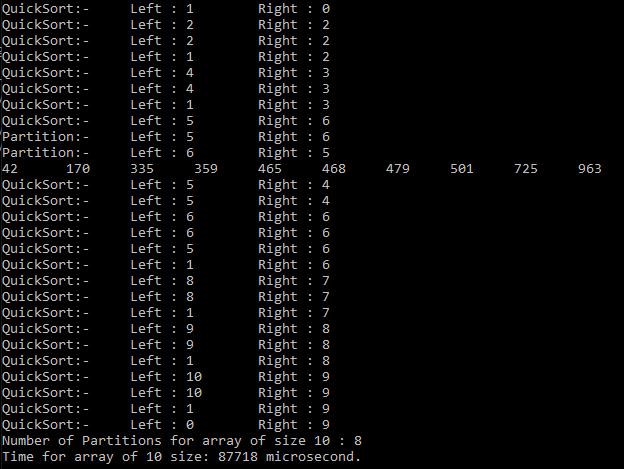
## Code:

|  |
| --- |
| #include <iostream> #include <chrono> #include <algorithm> using namespace std; using namespace std::chrono;  int RandomArray10[10];  int length;  class Sorting { public:  int count = 0;   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 RandomArrayGenerator(int arr[], int n)  {  *//method to generate random array* for(int i = 0 ; i < n ; i++)  {  arr[i] = (rand() % 1000) + 1;  }  }   int partition(int a[], int left, int right)  {  cout << "Partition:-\tLeft : " << left << "\t" << "Right : " << right << endl;  int pivot = left;  while (left <= right)  {  *//while left is less than right index* while (a[left] <= a[pivot] && left <= right)  {  *//while left value is less than or equal to the pivot index.* left++;  }  while (a[right] > a[pivot] && left <= right)  {  *//while the right index value is greater than pivot* right--;  }  if(left < right)  {  *//if left index is less than the right index swap the values.* Swap(&a[left], &a[right]);  }  Swap(&a[right], &a[pivot]);  }  cout << "Partition:-\tLeft : " << left << "\t" << "Right : " << right << endl;  return right;  }   void QuickSort(int a[], int left, int right)  {   cout << "QuickSort:-\tLeft : " << left << "\t" << "Right : " << right << endl;  if(left < right)  {  count++; *//to count number of partitions* int pivot = partition(a, left, right);  PrintArray(a, length);  cout << endl;  *//for left part of the array* QuickSort(a, left, pivot - 1);  *//for right part of the array* QuickSort(a, pivot + 1, right);  }  cout << "QuickSort:-\tLeft : " << left << "\t" << "Right : " << right << endl;   }  };   int main() {    Sorting \*sorting = new Sorting();  *//initializing the arrays with the random numbers using rand() function.* sorting -> RandomArrayGenerator(RandomArray10, 10);    *//selection sort* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t\tQuick sort\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";   sorting -> count = 0;   auto start = high\_resolution\_clock::now();  length = 10;  sorting -> QuickSort(RandomArray10, 0, 9);  auto stop = high\_resolution\_clock::now();  auto duration = duration\_cast<microseconds>(stop - start);  cout << "Number of Partitions for array of size 10 : " << sorting -> count << endl;  cout << "Time for array of 10 size: " << duration.count() << " microsecond.\n"<< endl;   return 0; } |

## Output:







# Task 04

## Code:

|  |
| --- |
| #include <iostream> #include <chrono> #include <algorithm> using namespace std; using namespace std::chrono;  class Sorting { public:   int count = 0;   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 RandomArrayGenerator(int arr[], int n)  {  *//method to generate random array* for(int i = 0 ; i < n ; i++)  {  arr[i] = (rand() % 1000) + 1;  }  }   int partition(int a[], int left, int right)  {  *///method to sort the array portion that have been partitioned.  //it sort by taking the pivot as left index value.* int pivot;  *//finding the median and then taking it as the pivot value.* int middle=(left+right)/2;  if(a[left]>a[middle])  {  if(a[middle]>a[right])  {  pivot=middle;  }  else if(a[right]>a[left])  {  pivot=left;  }  else  {  pivot=right;  }  }  else  {  if(a[middle]<a[right])  {  pivot=middle;  }  else if(a[left]>a[right])  {  pivot=left;  }  else  {  pivot=right;  }  }      while (left <= right)  {  while (a[left] <= a[pivot] && left <= right)  {  *//while left value is less then the pivot value left index is incremented.* left++;  }  while (a[right] > a[pivot] && left <= right)  {  *//while right index ix greater then the pivot index value* right--;  }  if(left < right)  {  *//if left is less than the right index then the values are swapped.* Swap(&a[left], &a[right]);  }  Swap(&a[right], &a[pivot]);  }  return right;  }   void QuickSort(int a[], int left, int right)  {  if(left < right)  {  count++;  int pivot = partition(a, left, right);  QuickSort(a, left, pivot - 1);  QuickSort(a, pivot + 1, right);  }   }  };   int main() {  int RandomArray100[100];  int RandomArray1000[1000];  int RandomArray5000[5000];   Sorting \*sorting = new Sorting();  *//initializing the arrays with the random numbers using rand() function.* sorting -> RandomArrayGenerator(RandomArray100, 100);  sorting -> RandomArrayGenerator(RandomArray1000, 1000);  sorting -> RandomArrayGenerator(RandomArray5000, 5000);   *//selection sort* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t\tQuick sort\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";   sorting -> count = 0;   auto start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray100, 0, 99);  auto stop = high\_resolution\_clock::now();  auto duration = duration\_cast<microseconds>(stop - start);  cout << "Number of Partitions for array of size 100 : " << sorting -> count << endl;  cout << "Time for array of 100 size: " << duration.count() << " microsecond.\n"<< endl;    sorting -> count = 0;  start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray1000, 0, 999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Number of Partitions for array of size 1000 : " << sorting -> count << endl;  cout << "Time for array of 1000 size: " << duration.count() << " microsecond.\n"<< endl;    sorting -> count = 0;  start = high\_resolution\_clock::now();  sorting -> QuickSort(RandomArray5000, 0, 4999);  stop = high\_resolution\_clock::now();  duration = duration\_cast<microseconds>(stop - start);  cout << "Number of Partitions for array of size 5000 : " << sorting -> count << endl;  cout << "Time for array of 1000 size: " << duration.count() << " microsecond.\n"<< endl;    return 0; } |

## Output: