Name: Sarvagnya H. Purohit **Registration Number**: 201070056
Date: 02-01-2022 **Branch:** Computer Engineering
Course: DSA **Course Instructor:** Dr. Mahesh Shirole

Lab Assignment 6

Aim: Implement the following sorting techniques for increasing and decreasing order of elements: Heap Sort, Merge Sort, Quick Sort, Bucket Sort, Radix Sort. Use student information as a data which has the following attributes: registrationID, name, address, grade.

Theory:

Heap Sort:

Heap Sort is a popular and efficient sorting algorithm in computer programming. Learning how to write the heap sort algorithm requires knowledge of two types of data structures - arrays and trees. The initial set of numbers that we want to sort is stored in an array e.g. [12, 5, 90, 34, 26, 32] and after sorting, we get a sorted array [5,12,26,32,34,90]. Heap sort works by visualizing the elements of the array as a special kind of complete binary tree called a heap.

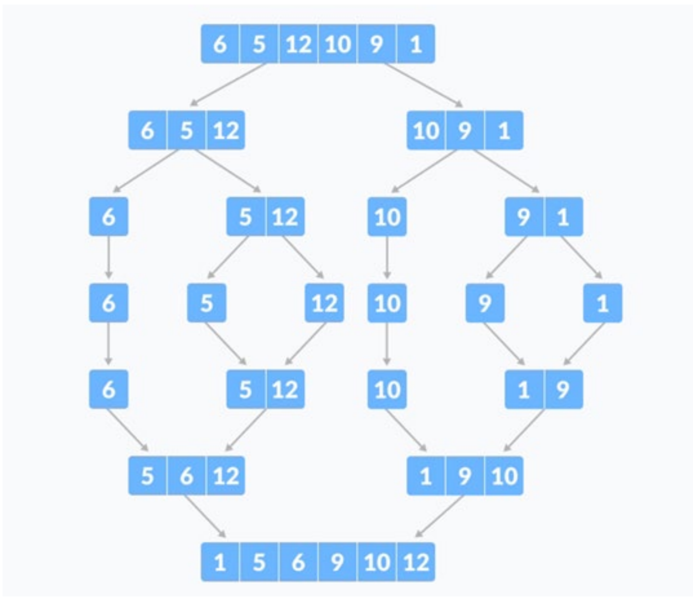
- Relationship between Array Indexes and Tree Elements: A complete binary tree has an interesting property that we can use to find the children and parents of any node. If the index of any element in the array is i, the element in the index 2i+1 will become the left child and element in 2i+2 index will become the right child. Also, the parent of any element at index i is given by the lower bound of (i-1)/2.
- Working of Heap Sort: Since the tree satisfies Max-Heap property, then the largest item is stored at the root node.
  - Swap: Remove the root element and put at the end of the array (nth position). Put the last item of the tree (heap) at the vacant place.
  - Remove: Reduce the size of the heap by 1.
  - Heapify: Heapify the root element again so that we have the highest element at root.
  - The process is repeated until all the items of the list are sorted.

Merge Sort:

Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm. Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are combined to form the final solution.

Using the Divide and Conquer technique, we divide a problem into subproblems. When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem. Suppose we had to sort an array A. A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p..r].

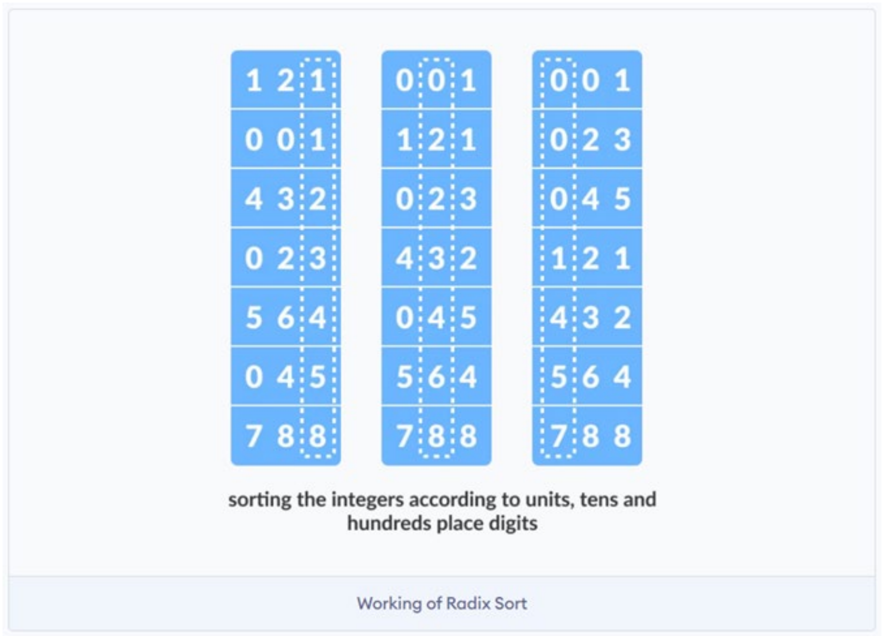
- Divide: If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays A[p..q] and A[q+1, r].
- Conquer: In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.
- Combine: When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1, r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1, r].



Quick Sort:
Quicksort is a sorting algorithm based on the divide and conquer approach where:

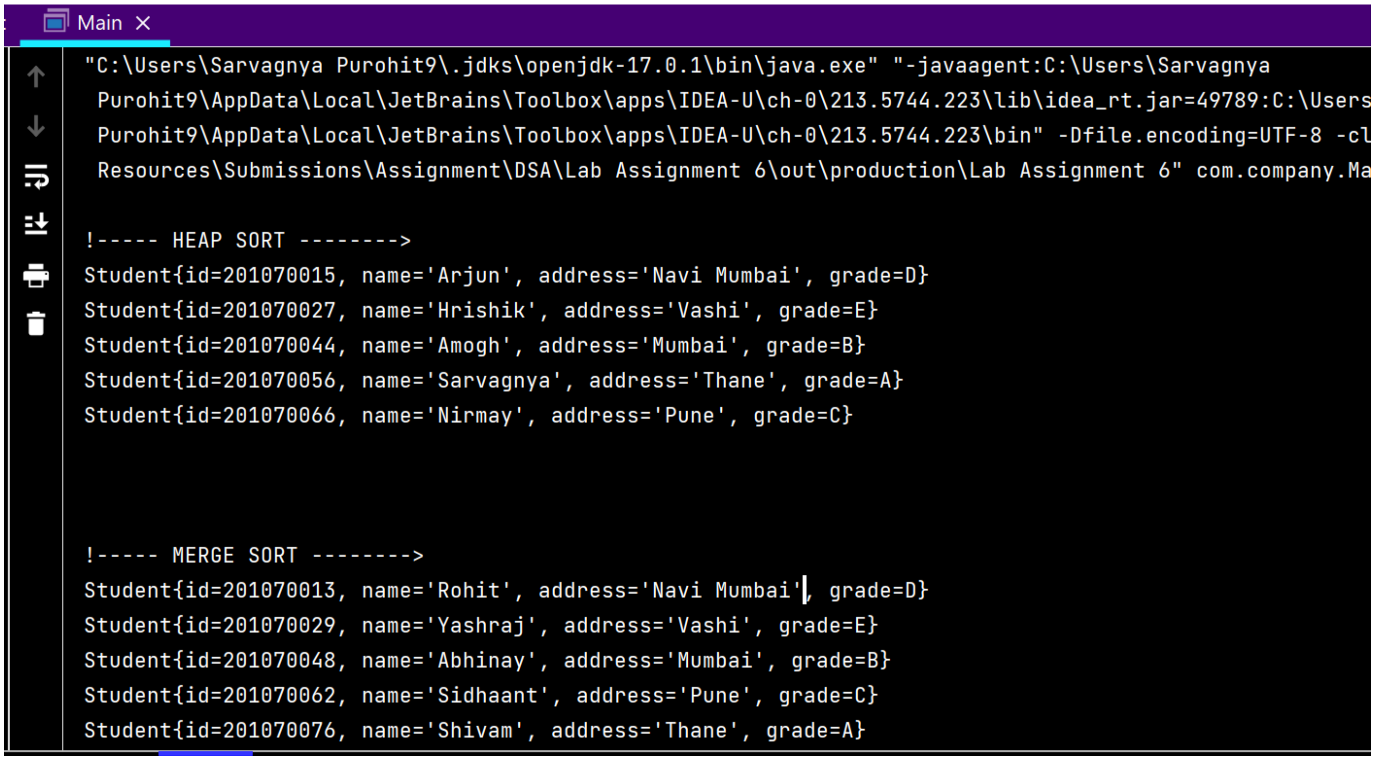
1. An array is divided into subarrays by selecting a pivot element (element selected from the array). While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.
2. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
3. At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

Radix Sort:
Radix sort is a sorting algorithm that sorts the elements by first grouping the individual digits of the same place value. Then, sort the elements according to their increasing/decreasing order. Suppose, we have an array of 8 elements. First, we will sort elements based on the value of the unit place. Then, we will sort elements based on the value of the tenth place. This process goes on until the last significant place. Let the initial array be [121, 432, 564, 23, 1, 45, 788]. It is sorted according to radix sort as shown in the figure below:



Working of Radix Sort

Test Data and Output:

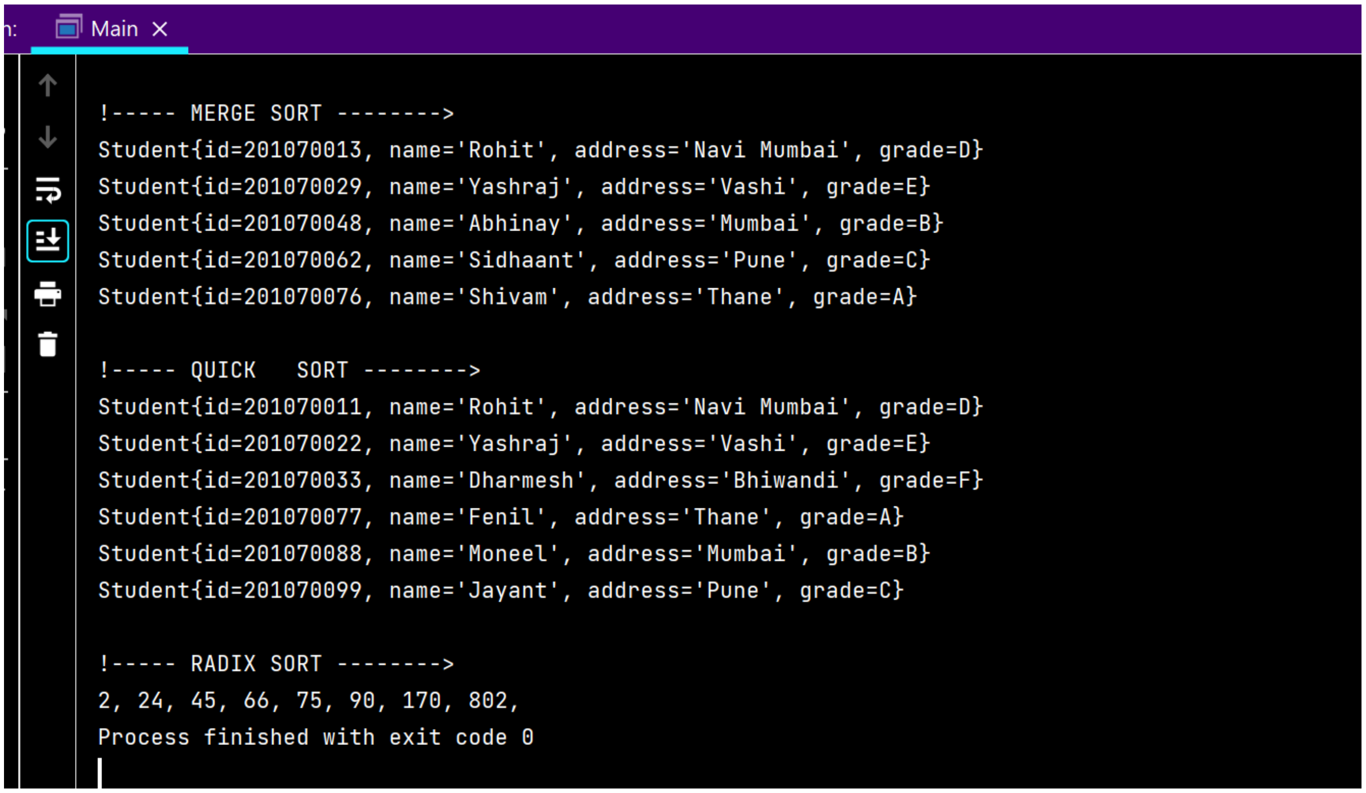


```
"C:\Users\Sarvagnya Purohit9\.jdks\openjdk-17.0.1\bin\java.exe" "-javaagent:C:\Users\Sarvagnya
 Purohit9\AppData\Local\JetBrains\Toolbox\apps\IDEA-U\ch-0\213.5744.223\lib\idea_rt.jar=49789:C:\Users
 Purohit9\AppData\Local\JetBrains\Toolbox\apps\IDEA-U\ch-0\213.5744.223\bin" -Dfile.encoding=UTF-8 -cl
 Resources\Submissions\Assignment\DSA\Lab Assignment 6\out\production\Lab Assignment 6" com.company.Ma

!----- HEAP SORT -------->
Student{id=201070015, name='Arjun', address='Navi Mumbai', grade=D}
Student{id=201070027, name='Hrishik', address='Vashi', grade=E}
Student{id=201070044, name='Amogh', address='Mumbai', grade=B}
Student{id=201070056, name='Sarvagnya', address='Thane', grade=A}
Student{id=201070066, name='Nirmay', address='Pune', grade=C}



!----- MERGE SORT -------->
Student{id=201070013, name='Rohit', address='Navi Mumbai', grade=D}
Student{id=201070029, name='Yashraj', address='Vashi', grade=E}
Student{id=201070048, name='Abhinay', address='Mumbai', grade=B}
Student{id=201070062, name='Sidhaant', address='Pune', grade=C}
Student{id=201070076, name='Shivam', address='Thane', grade=A}
```



```
!----- MERGE SORT -------->
Student{id=201070013, name='Rohit', address='Navi Mumbai', grade=D}
Student{id=201070029, name='Yashraj', address='Vashi', grade=E}
Student{id=201070048, name='Abhinay', address='Mumbai', grade=B}
Student{id=201070062, name='Sidhaant', address='Pune', grade=C}
Student{id=201070076, name='Shivam', address='Thane', grade=A}

!----- QUICK   SORT -------->
Student{id=201070011, name='Rohit', address='Navi Mumbai', grade=D}
Student{id=201070022, name='Yashraj', address='Vashi', grade=E}
Student{id=201070033, name='Dharmesh', address='Bhiwandi', grade=F}
Student{id=201070077, name='Fenil', address='Thane', grade=A}
Student{id=201070088, name='Moneel', address='Mumbai', grade=B}
Student{id=201070099, name='Jayant', address='Pune', grade=C}

!----- RADIX SORT -------->
2, 24, 45, 66, 75, 90, 170, 802,
Process finished with exit code 0
```

Conclusion:

We learnt about various sorting methods such as heap sort, merge sort, quick sort and radix sort and implemented them to sort our data of Student objects – we sorted the Student objects in an ascending order of their registration numbers with the help of ***Comparator*** in Java.
In case of Heap sort, we used up-heap and where the minimum value is present at the root position and then we extracted the value at root, stored it in a list, removed the node and rearranged the heap. Continuing in this similar fashion, we were able to sort the elements of the heap.
In Merge Sort we saw the principle of Divide and Conquer Algorithm. Here we divided a problem into multiple sub-problems, solved each sub-problem individually and finally combined them (or merged them) to get the final solution.
Quick Sort is also based on the divide and conquer approach where we divided an array into subarrays by selecting a pivot element (element selected from the array which is be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot). Proceeding with the similar approach for the left and right subarrays until each subarray contains a single element. At this point, elements are already sorted. Finally, we combined the elements to form a sorted array.
In case of Radix sort we first grouped the individual digits of the same place value , then sorted the elements according to their increasing order until the last significant place and resultant array is sorted