Name: Kashaf Khan
Roll Number: 2023F-BSE-248
Section: A

LAB # 05

Sorting on Linear Arrays in JAVA

OBJECTIVE:

To sort a linear array using Selection Sort, Bubble Sort and Merge Sort..

LAB TASKS:

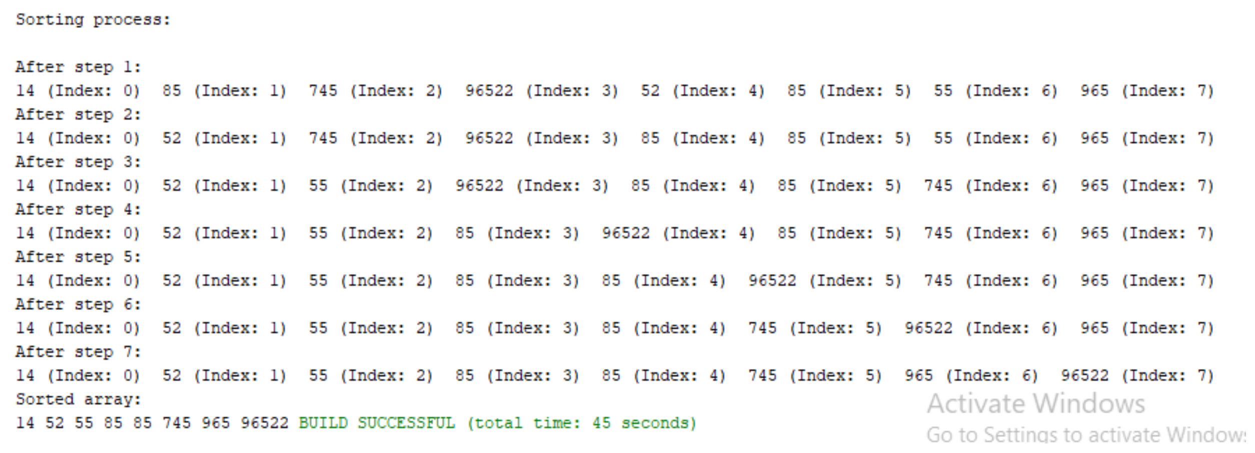
1. Write a program for Selection sort that sorts an array containing numbers, prints all the sort values of array each followed by its location.

CODE:

```
 public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     System.out.print("Enter the size of the array: ");
     int n = scanner.nextInt();
     int[] array = new int[n];
     System.out.println("Enter the elements of the array:");
     for (int i = 0; i < n; i++) {
        System.out.print("Element " + (i + 1) + ": ");
        array[i] = scanner.nextInt();
     System.out.println("\nSorting process:");
     selectionSort(array);
     System.out.println("\nSorted array:");
     for (int num : array) {
        System.out.print(num + " ");
  public static void selectionSort(int[] array) {
     int n = array.length;
```

```
    for (int i = 0; i < n - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < n; j++) {
        if (array[j] < array[minIndex]) {
      if (minIndex != i) {
        int temp = array[i];
        array[i] = array[minIndex];
        array[minIndex] = temp;
      System.out.println("\nAfter step " + (i + 1) + ":");
      for (int k = 0; k < array.length; k++) {
        System.out.print(array[k] + " (Index: " + k + ")  ");
```

OUTPUT



```
Sorting process:

After step 1:
14 (Index: 0)  85 (Index: 1)  745 (Index: 2)  96522 (Index: 3)  52 (Index: 4)  85 (Index: 5)  55 (Index: 6)  965 (Index: 7)
After step 2:
14 (Index: 0)  52 (Index: 1)  745 (Index: 2)  96522 (Index: 3)  85 (Index: 4)  85 (Index: 5)  55 (Index: 6)  965 (Index: 7)
After step 3:
14 (Index: 0)  52 (Index: 1)  55 (Index: 2)  96522 (Index: 3)  85 (Index: 4)  85 (Index: 5)  745 (Index: 6)  965 (Index: 7)
After step 4:
14 (Index: 0)  52 (Index: 1)  55 (Index: 2)  85 (Index: 3)  96522 (Index: 4)  85 (Index: 5)  745 (Index: 6)  965 (Index: 7)
After step 5:
14 (Index: 0)  52 (Index: 1)  55 (Index: 2)  85 (Index: 3)  85 (Index: 4)  96522 (Index: 5)  745 (Index: 6)  965 (Index: 7)
After step 6:
14 (Index: 0)  52 (Index: 1)  55 (Index: 2)  85 (Index: 3)  85 (Index: 4)  745 (Index: 5)  96522 (Index: 6)  965 (Index: 7)
After step 7:
14 (Index: 0)  52 (Index: 1)  55 (Index: 2)  85 (Index: 3)  85 (Index: 4)  745 (Index: 5)  965 (Index: 6)  96522 (Index: 7)
Sorted array:
14 52 55 85 85 745 965 96522 BUILD SUCCESSFUL (total time: 45 seconds)
```

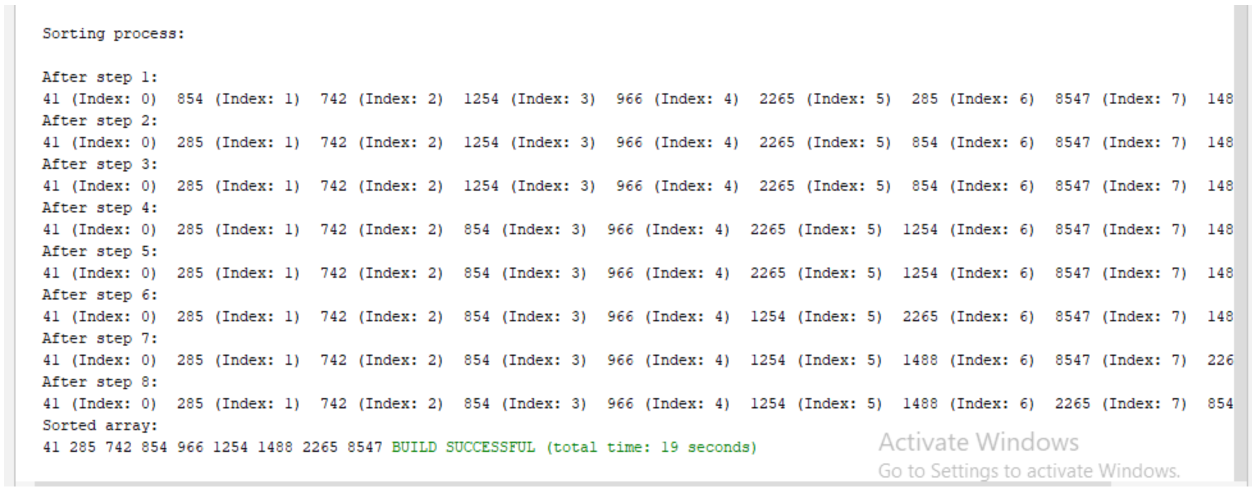
2. Write a program that takes 10 numbers as input in an array. Sort the elements of array by using Bubble sort. Print each iteration of the sorting process.

CODE:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the size of the array: ");
    int n = scanner.nextInt();
    int[] array = new int[n];
    System.out.println("Enter the elements of the array:");
    for (int i = 0; i < n; i++) {
      System.out.print("Element " + (i + 1) + ": ");
      array[i] = scanner.nextInt();
    System.out.println("\nSorting process:");
    selectionSort(array);
    System.out.println("\nSorted array:");
    for (int num : array) {
      System.out.print(num + " ");
  public static void selectionSort(int[] array) {
    int n = array.length;
    for (int i = 0; i < n - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < n; j++) {
        if (array[j] < array[minIndex]) {
          minIndex = j;
      if (minIndex != i) {
     int temp = array[i];
```

```
        array[i] = array[minIndex];

        array[minIndex] = temp;

      System.out.println("\nAfter step " + (i + 1) + ":");

      for (int k = 0; k < array.length; k++) {

        System.out.print(array[k] + " (Index: " + k + ")  ");
```

OUTPUT:



```
Sorting process:

After step 1:
41 (Index: 0)  854 (Index: 1)  742 (Index: 2)  1254 (Index: 3)  966 (Index: 4)  2265 (Index: 5)  285 (Index: 6)  8547 (Index: 7)  148
After step 2:
41 (Index: 0)  285 (Index: 1)  742 (Index: 2)  1254 (Index: 3)  966 (Index: 4)  2265 (Index: 5)  854 (Index: 6)  8547 (Index: 7)  148
After step 3:
41 (Index: 0)  285 (Index: 1)  742 (Index: 2)  1254 (Index: 3)  966 (Index: 4)  2265 (Index: 5)  854 (Index: 6)  8547 (Index: 7)  148
After step 4:
41 (Index: 0)  285 (Index: 1)  742 (Index: 2)  854 (Index: 3)  966 (Index: 4)  2265 (Index: 5)  1254 (Index: 6)  8547 (Index: 7)  148
After step 5:
41 (Index: 0)  285 (Index: 1)  742 (Index: 2)  854 (Index: 3)  966 (Index: 4)  2265 (Index: 5)  1254 (Index: 6)  8547 (Index: 7)  148
After step 6:
41 (Index: 0)  285 (Index: 1)  742 (Index: 2)  854 (Index: 3)  966 (Index: 4)  1254 (Index: 5)  2265 (Index: 6)  8547 (Index: 7)  148
After step 7:
41 (Index: 0)  285 (Index: 1)  742 (Index: 2)  854 (Index: 3)  966 (Index: 4)  1254 (Index: 5)  1488 (Index: 6)  8547 (Index: 7)  226
After step 8:
41 (Index: 0)  285 (Index: 1)  742 (Index: 2)  854 (Index: 3)  966 (Index: 4)  1254 (Index: 5)  1488 (Index: 6)  2265 (Index: 7)  854
Sorted array:
41 285 742 854 966 1254 1488 2265 8547 BUILD SUCCESSFUL (total time: 19 seconds)
```

3. Write a program that takes 10 random numbers in an array. Sort the elements of array

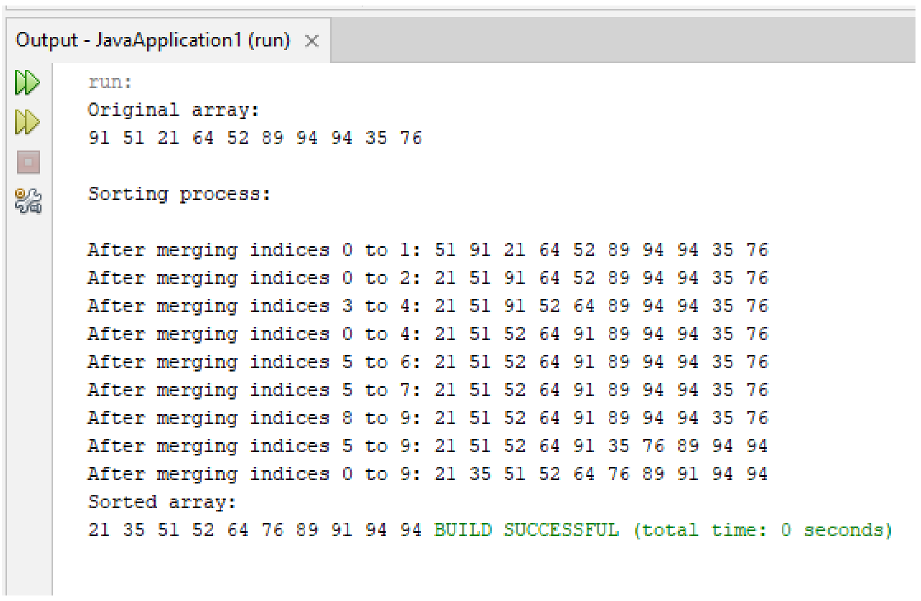
by using Merge sort applying recursive technique. Print each iteration of the sorting process.
CODE:

```
 public static void main(String[] args) {

     int[] array = new int[10];

     Random random = new Random();

     System.out.println("Original array:");

     for (int i = 0; i < 10; i++) {

       array[i] = random.nextInt(100);

       System.out.print(array[i] + " ");
```

```
    System.out.println("\n\nSorting process:");
    mergeSort(array, 0, array.length - 1);
    System.out.println("\nSorted array:");
    for (int num : array) {
      System.out.print(num + " ");
  public static void mergeSort(int[] array, int left, int right) {
    if (left < right) {
      int mid = left + (right - left) / 2;
      mergeSort(array, left, mid);
      mergeSort(array, mid + 1, right);
      merge(array, left, mid, right);
      System.out.print("\nAfter merging indices " + left + " to " + right + ": ");
      for (int num : array) {
        System.out.print(num + " ");
  public static void merge(int[] array, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int[] leftArray = new int[n1];
    int[] rightArray = new int[n2];
    for (int i = 0; i < n1; i++) {
      leftArray[i] = array[left + i];
    for (int j = 0; j < n2; j++) {
      rightArray[j] = array[mid + 1 + j];
    int i = 0, j = 0;
    int k = left;
```

```
while (i < n1 && j < n2) {
  if (leftArray[i] <= rightArray[j]) {
    array[k] = leftArray[i];
    i++;
   else {
    array[k] = rightArray[j];
    j++;
  k++;
while (i < n1) {
  array[k] = leftArray[i];
  i++;
  k++;
while (j < n2) {
  array[k] = rightArray[j];
  j++;
  k++;
```

OUTPUT:



Output - JavaApplication1 (run)

```
run:
Original array:
91 51 21 64 52 89 94 94 35 76

Sorting process:

After merging indices 0 to 1: 51 91 21 64 52 89 94 94 35 76
After merging indices 0 to 2: 21 51 91 64 52 89 94 94 35 76
After merging indices 3 to 4: 21 51 91 52 64 89 94 94 35 76
After merging indices 0 to 4: 21 51 52 64 91 89 94 94 35 76
After merging indices 5 to 6: 21 51 52 64 91 89 94 94 35 76
After merging indices 5 to 7: 21 51 52 64 91 89 94 94 35 76
After merging indices 8 to 9: 21 51 52 64 91 89 94 94 35 76
After merging indices 5 to 9: 21 51 52 64 91 35 76 89 94 94
After merging indices 0 to 9: 21 35 51 52 64 76 89 91 94 94
Sorted array:
21 35 51 52 64 76 89 91 94 94 BUILD SUCCESSFUL (total time: 0 seconds)
```

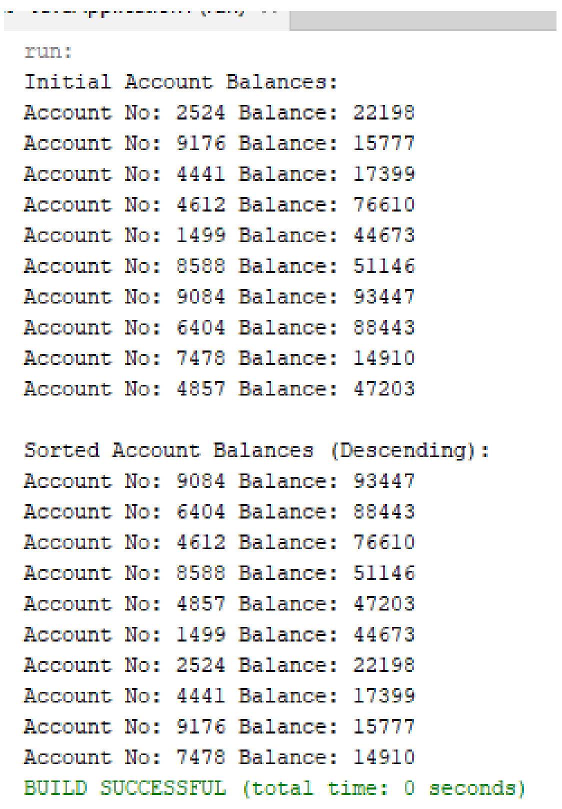
HOME TASKS:

Declare an array of size n to store account balances. Initialize with values 0 to 100000 and sort Account No's according to highest balance values by using Quick sort, For e.g.:

CODE:

```
 public static void main(String[] args) {
     int n = 10;
     int[] accountNos = new int[n];
     int[] balances = new int[n];
     Random random = new Random();
     System.out.println("Initial Account Balances:");
     for (int i = 0; i < n; i++) {
       accountNos[i] = 1000 + random.nextInt(9000);
       balances[i] = random.nextInt(100001);
       System.out.println("Account No: " + accountNos[i] + " Balance: " + balances[i]);
     quickSort(accountNos, balances, 0, n - 1);
     System.out.println("\nSorted Account Balances (Descending):");
     for (int i = 0; i < n; i++) {
       System.out.println("Account No: " + accountNos[i] + " Balance: " + balances[i]);
  public static void quickSort(int[] accountNos, int[] balances, int low, int high) {
     if (low < high) {
       int pi = partition(accountNos, balances, low, high);
       quickSort(accountNos, balances, low, pi - 1);
       quickSort(accountNos, balances, pi + 1, high);
  public static int partition(int[] accountNos, int[] balances, int low, int high) {
     int pivot = balances[high];
     int i = low - 1;
     for (int j = low; j < high; j++) {
       if (balances[j] > pivot) {
          i++;
          int tempBalance = balances[i];
          balances[i] = balances[j];
          balances[j] = tempBalance;
          int tempAccount = accountNos[i];
          accountNos[i] = accountNos[j];
     int tempBalance = balances[i + 1];
     balances[i + 1] = balances[high];
     balances[high] = tempBalance;
     int tempAccount = accountNos[i + 1];
     accountNos[i + 1] = accountNos[high];
     accountNos[high] = tempAccount;
     return i + 1
```

OUTPUT:



```
run:
Initial Account Balances:
Account No: 2524 Balance: 22198
Account No: 9176 Balance: 15777
Account No: 4441 Balance: 17399
Account No: 4612 Balance: 76610
Account No: 1499 Balance: 44673
Account No: 8588 Balance: 51146
Account No: 9084 Balance: 93447
Account No: 6404 Balance: 88443
Account No: 7478 Balance: 14910
Account No: 4857 Balance: 47203

Sorted Account Balances (Descending):
Account No: 9084 Balance: 93447
Account No: 6404 Balance: 88443
Account No: 4612 Balance: 76610
Account No: 8588 Balance: 51146
Account No: 4857 Balance: 47203
Account No: 1499 Balance: 44673
Account No: 2524 Balance: 22198
Account No: 4441 Balance: 17399
Account No: 9176 Balance: 15777
Account No: 7478 Balance: 14910
BUILD SUCCESSFUL (total time: 0 seconds)
```

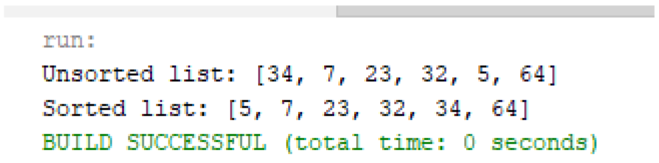
2. Write a program which takes an unordered list of integers (or any other objects e.g. String), you have to rearrange the list in their natural order using merge sort.

CODE:

```
public static void main(String[] args) {
    ArrayList<Integer> list = new ArrayList<>();
    list.add(34);  list.add(7);   list.add(23);
    list.add(32);  list.add(5);list.add(64);
    System.out.println("Unsorted list: " + list);
    mergeSort(list, 0, list.size() - 1);
    System.out.println("Sorted list: " + list);
  public static <T extends Comparable<T>> void mergeSort(ArrayList<T> list, int left, int right) {
    if (left < right) {
      int mid = left + (right - left) / 2;
```

```
        mergeSort(list, left, mid);

        mergeSort(list, mid + 1, right);

        merge(list, left, mid, right);

  public static <T extends Comparable<T>> void merge(ArrayList<T> list, int left, int mid, int right)
{

      int n1 = mid - left + 1;

      int n2 = right - mid;

      ArrayList<T> leftList = new ArrayList<>();

      ArrayList<T> rightList = new ArrayList<>();

      for (int i = 0; i < n1; i++) {

        leftList.add(list.get(left + i));

      for (int j = 0; j < n2; j++) {

        rightList.add(list.get(mid + 1 + j));

      int i = 0, j = 0, k = left;

      while (i < n1 && j < n2) {

        if (leftList.get(i).compareTo(rightList.get(j)) <= 0) {

          list.set(k, leftList.get(i));

          i++;

         else {

          list.set(k, rightList.get(j));

      while (i < n1) {

        list.set(k, leftList.get(i));

      while (j < n2) {

        list.set(k, rightList.get(j));

        j++;

        k++;
```

OUTPUT:



```
run:
Unsorted list: [34, 7, 23, 32, 5, 64]
Sorted list: [5, 7, 23, 32, 34, 64]
BUILD SUCCESSFUL (total time: 0 seconds)
```

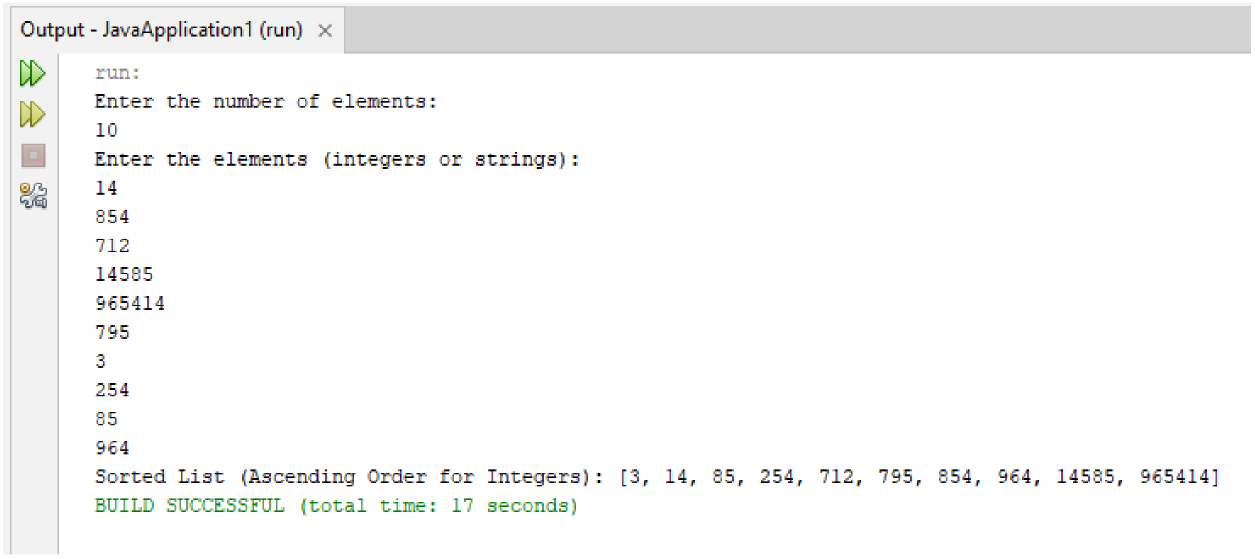
3. You are given an unordered list of integers or strings. Write a program to Take this list as input. Sort it in **natural order** using Merge Sort. For integers, this means ascending order. For strings, this means alphabetical order. Print the sorted list.

CODE:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of elements:");
    int n = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    ArrayList<String> list = new ArrayList<>();
    System.out.println("Enter the elements (integers or strings):");
    for (int i = 0; i < n; i++) {
      list.add(scanner.nextLine());
    try {
      ArrayList<Integer> intList = new ArrayList<>();
      for (String element : list) {
        intList.add(Integer.parseInt(element));
      mergeSort(intList, 0, intList.size() - 1);
      System.out.println("Sorted List (Ascending Order for Integers): " + intList);
     catch (NumberFormatException e) {
      mergeSort(list, 0, list.size() - 1);
```

```
        System.out.println("Sorted List (Alphabetical Order for Strings): " + list);
  public static <T extends Comparable<T>> void mergeSort(ArrayList<T> list, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(list, left, mid);
        mergeSort(list, mid + 1, right);
        merge(list, left, mid, right);
  public static <T extends Comparable<T>> void merge(ArrayList<T> list, int left, int mid, int right)
{
    int n1 = mid - left + 1;
    int n2 = right - mid;
    ArrayList<T> leftList = new ArrayList<>();
    ArrayList<T> rightList = new ArrayList<>();
    for (int i = 0; i < n1; i++) {
        leftList.add(list.get(left + i));
    for (int j = 0; j < n2; j++) {
        rightList.add(list.get(mid + 1 + j));
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftList.get(i).compareTo(rightList.get(j)) <= 0) {
            list.set(k, leftList.get(i));
            i++;
         else {
            list.set(k, rightList.get(j));
    while (i < n1) {
        list.set(k, leftList.get(i));
    while (j < n2) {
        list.set(k, rightList.get(j));
```

OUTPUT:



Output - JavaApplication1 (run)

```
run:
Enter the number of elements:
10
Enter the elements (integers or strings):
14
854
712
14585
965414
795
3
254
85
964
Sorted List (Ascending Order for Integers): [3, 14, 85, 254, 712, 795, 854, 964, 14585, 965414]
BUILD SUCCESSFUL (total time: 17 seconds)
```

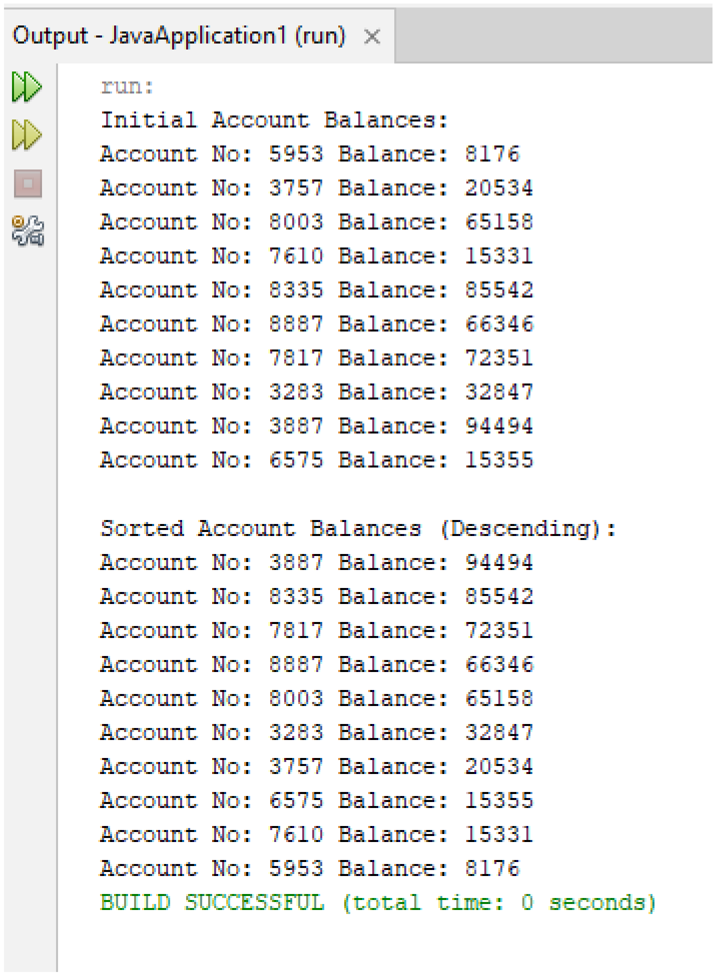
4. You are given a set of bank accounts, each with a unique account number and a balance. Write a Java program to Declare an array of size n to store account balances. Initialize each balance randomly with values between 0 and 100,000. Sort the accounts in **descending order** of their balances using Quick Sort. Print the sorted list in the format

CODE:

```
public static void main(String[] args) {
    int n = 10;  // Number of bank accounts
    int[] accountNos = new int[n];
    int[] balances = new int[n];
    Random random = new Random();
    System.out.println("Initial Account Balances:");
    for (int i = 0; i < n; i++) {
        accountNos[i] = 1000 + random.nextInt(9000); // Random account number between 1000
and 9999
        balances[i] = random.nextInt(100001); // Random balance between 0 and 100000
        System.out.println("Account No: " + accountNos[i] + " Balance: " + balances[i]);
    quickSort(accountNos, balances, 0, n - 1);
    System.out.println("\nSorted Account Balances (Descending):");
    for (int i = 0; i < n; i++) {
      System.out.println("Account No: " + accountNos[i] + " Balance: " + balances[i]);
  public static void quickSort(int[] accountNos, int[] balances, int low, int high) {
    if (low < high) {
      int pi = partition(accountNos, balances, low, high);
      quickSort(accountNos, balances, low, pi - 1);
      quickSort(accountNos, balances, pi + 1, high);
  public static int partition(int[] accountNos, int[] balances, int low, int high) {
    int pivot = balances[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
      if (balances[j] > pivot) {
        i++;            int tempBalance = balances[i];
        balances[i] = balances[j];
        balances[j] = tempBalance;
        int tempAccount = accountNos[i];
        accountNos[i] = accountNos[j];
        accountNos[j] = tempAccount;
    int tempBalance = balances[i + 1];
    balances[i + 1] = balances[high];
    balances[high] = tempBalance;
    int tempAccount = accountNos[i + 1];
    accountNos[i + 1] = accountNos[high];
    accountNos[high] = tempAccount;
```

```
return i + 1;
```

OUTPUT:



```
run:
Initial Account Balances:
Account No: 5953 Balance: 8176
Account No: 3757 Balance: 20534
Account No: 8003 Balance: 65158
Account No: 7610 Balance: 15331
Account No: 8335 Balance: 85542
Account No: 8887 Balance: 66346
Account No: 7817 Balance: 72351
Account No: 3283 Balance: 32847
Account No: 3887 Balance: 94494
Account No: 6575 Balance: 15355

Sorted Account Balances (Descending):
Account No: 3887 Balance: 94494
Account No: 8335 Balance: 85542
Account No: 7817 Balance: 72351
Account No: 8887 Balance: 66346
Account No: 8003 Balance: 65158
Account No: 3283 Balance: 32847
Account No: 3757 Balance: 20534
Account No: 6575 Balance: 15355
Account No: 7610 Balance: 15331
Account No: 5953 Balance: 8176
BUILD SUCCESSFUL (total time: 0 seconds)
```