Sorting Techniques

1. Binary Search

**package** Sorting;

**public** **class** BinarySearch {

**int** binarySearch(**int** arr[], **int** low, **int** high, **int** searchElement) {

**if**(high >= low) {

**int** mid = (low + (high - 1) )/2;

**if**(arr[mid]==searchElement)

**return** mid;

**else** **if**(arr[mid] > searchElement) { //search the element in the right side of array

**return** binarySearch(arr, low, mid - 1, searchElement);

}

**else** **if**(arr[mid] < searchElement) { //search the element in the left side of array

**return** binarySearch(arr, mid + 1, high, searchElement);

}

**return** -1;

}

**return** -1;

}

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

BinarySearch bs = **new** BinarySearch();

**int** arr[] = {1,2,3,8,9,10}; //given array

**int** n = arr.length;

**int** searchElement = 9; //element to be searched

**int** result = bs.binarySearch(arr, 0 , n - 1, searchElement); //creating customized method and passing values

**if**(result == -1) {

System.***out***.println("Element not present");

}

**else** {

System.***out***.println("Element found at index :" +result);

}

}

}

Output:  
Element found at index :4

1. Bubble Sort

**package** Sorting;

**public** **class** Bubblesort {

**static** **void** print (**int** a[]) //function to print array elements

{

**int** n = a.length;

**int** i;

**for** (i = 0; i < n; i++)

{

System.***out***.print(a[i] + " ");

}

}

**static** **void** bubbleSort (**int** a[]) // function to implement bubblesort

{

**int** n = a.length;

**int** i, j, temp;

**for** (i = 0; i < n; i++)

{

**for** (j = i + 1; j < n; j++)

{

**if** (a[j] < a[i])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

}

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

**int** a[] = {35, 10, 31, 11, 26};

Bubblesort b1 = **new** Bubblesort();

System.***out***.println("Before sorting array elements are - ");

b1.*print*(a);

b1.*bubbleSort*(a);

System.***out***.println();

System.***out***.println("After sorting array elements are - ");

b1.*print*(a);

}

}

Output:   
Before sorting array elements are -

35 10 31 11 26

After sorting array elements are -

10 11 26 31 35

1. Merge Sort

**package** Sorting;

**public** **class** MergeSort {

**void** mergesort(**int** arr[], **int** l , **int** m, **int** r) {

**int** n1 = m - l + 1;

**int** n2 = r - m;

**int** L[] = **new** **int**[n1];

**int** M[] = **new** **int**[n2];

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

L[i] = arr[l + i];

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

M[j] = arr[m + 1 + j];

**int** i, j, k;

i = 0; //starting index of i

j = 0;

k = l;

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

**if** (L[i] <= M[j]) { // if i in the left array is less than j in the right array

arr[k] = L[i]; //put i value in the k array

i++; //then increase i

} **else** {

arr[k] = M[j]; // then j is placed in the k

j++; //then increase j

}

k++; //increase k in both cases

}

**while** (i < n1) { //while i less then mid value

arr[k] = L[i]; //then put i value in k

i++;

k++;

}

**while** (j < n2) { //check if j is less than n2

arr[k] = M[j]; //place j in k array

j++;

k++;

}

}

**void** mergeSort(**int** arr[], **int** l, **int** r) {

**if**(l < r) {

**int** m = (l + r) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r); //recursive method

mergesort(arr, l, m, r); //merge both methods

}

}

**static** **void** printArray(**int** arr[]) {

**int** n = arr.length;

**for** (**int** i = 0; i < n; ++i) //post increment

System.***out***.print(arr[i] + ", ");

System.***out***.println();

}

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

**int** arr[] = { 15, 5, 24, 8, 1, 3, 16, 10, 20 }; //given array of elements

MergeSort ob = **new** MergeSort();

ob.mergeSort(arr, 0, arr.length - 1); //combine array from 0 index to last index

System.***out***.println("Sorted array:"); //print sorted array

*printArray*(arr);

}

}

Output:  
Sorted array:

1, 3, 5, 8, 10, 15, 16, 20, 24,

1. Quick Sort

**package** Sorting;

**public** **class** QuickSort {

**int** partition (**int** a[], **int** start, **int** end) {

**int** pivot = a[end]; // pivot element

**int** i = (start - 1);

**for** (**int** j = start; j <= end - 1; j++) {

// If current element is smaller than the pivot

**if** (a[j] < pivot) {

i++; // increment index of smaller element

**int** t = a[i];

a[i] = a[j];

a[j] = t;

}

}

**int** t = a[i+1];

a[i+1] = a[end];

a[end] = t;

**return** (i + 1);

}

**void** quick(**int** a[], **int** start, **int** end) { /\* a[] = array to be sorted, start = Starting index, end = Ending index \*/

**if** (start < end) {

**int** p = partition(a, start, end); //p is partitioning index

quick(a, start, p - 1);

quick(a, p + 1, end);

}

}

**void** printArr(**int** a[], **int** n) {

**int** i;

**for** (i = 0; i < n; i++)

System.***out***.print(a[i] + " ");

}

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

**int** data[] = { 8, 7, 2, 1, 0, 9, 6 };

**int** n = data.length;

System.***out***.println("\nUnsorted Array");

QuickSort a1 = **new** QuickSort();

a1.printArr(data, n);

a1.quick(data, 0, n - 1);

System.***out***.println("\nAfter sorting array elements are - ");

a1.printArr(data, n);

System.***out***.println();

}

}

Output:  
Unsorted Array

8 7 2 1 0 9 6

After sorting array elements are -

0 1 2 6 7 8 9