

Data Structures and Algorithms using Java
Professor. Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 55
Sorting Using JCF

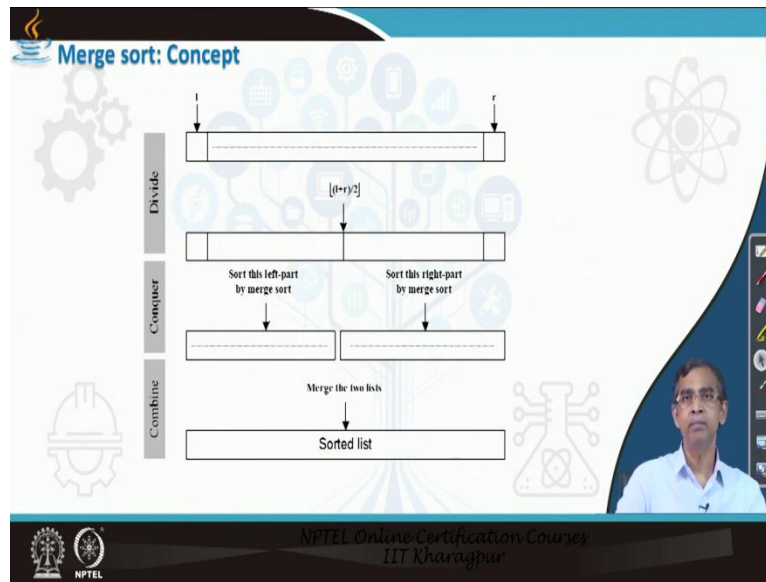
We have practiced many sorting algorithm, where we plan to write the code of our own, based on the algorithm, their logic and we have also studied Java collections framework, where we can store different type of data in the form of collection and sorting as I told or searching, this is the obvious, what is called the two operations in many applications, considering this, the Java developer includes sorting and searching algorithms as a part of, as a bundle of the different methods in order to deal with the collection.

(Refer Slide Time: 1:13)



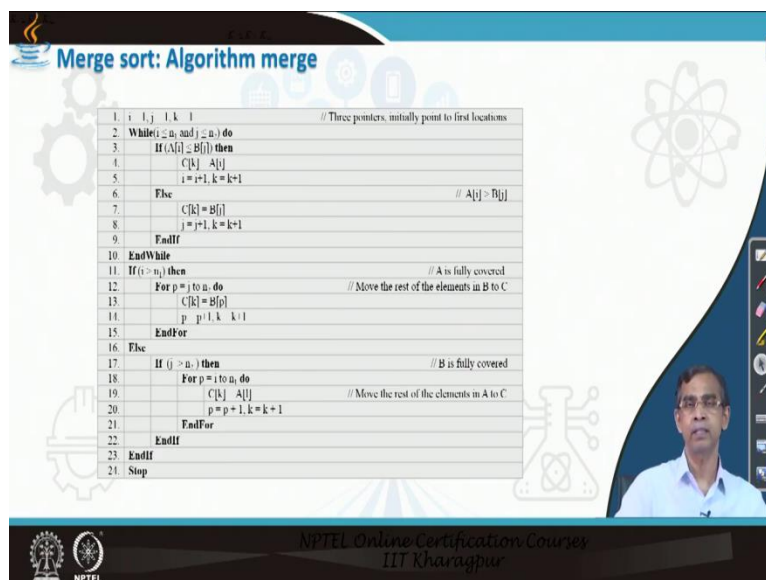
So, today in this discussion, we will discuss about how the sorting techniques supported by the Java collection framework, but before going to this Java collection framework learning, first we have to discuss about the Merge Sort, which is one very interesting sorting algorithm, which we did not discuss about it.

(Refer Slide Time: 1:48)



So, let us first discuss the merge sort and then finally, we will discuss about the sorting technique, which basically is supported by Java Collection Framework. So, first let us discuss the merge sort. As you know, merge sort algorithm is basically based on divide and conquer, where we have to divide that to least, almost equal to two parts, equal parts and then recursively call the merge operation and then finally, merging them. So, there are recursive definition of the merge sort is to be done, we have discussed about how quick sort technique also can be defined recursively, it is basically in the same way we can do it.

(Refer Slide Time: 2:28)



So, this is the algorithm that you can think for merge operation, it is very easy operation actually, two list at there, you have to just move the pointer from the two parts of the list and then finally, combining the results. So, this logic you can follow from the algorithm.

(Refer Slide Time: 2:46)

Merge sort: Algorithm

Input: An array $A[l..r]$ where l and r are the lower and upper indexes of A .
Output: Array $A[l..r]$ with all elements are arranged in ascending order.

Algorithm MergeSort($A[l..r]$)

```
1. If ( $r \leq l$ ) then
2.   Return // Termination of recursion
3. Else
4.    $mid = \lfloor \frac{l+r}{2} \rfloor$  // Divide: Find the index of the middle of the list
5.   MergeSort( $A[l..mid]$ ) // Conquer for the left-part
6.   MergeSort( $A[mid+1..r]$ ) // Conquer for the right-part
7.   Merge( $A$ ,  $mid$ ,  $r$ ) // Combine: Merging the sorted left- and right- parts
8. EndIf
9. Stop
```

NPTEL Online Certification Courses
IIT Kharagpur

Otherwise, you can write, this logic is important and then this is basically the, this is basically the technique that merge sort how recursively it can be defined. So, you have to find the mid calculation, how the mid calculation can be done using this floor and then you have to call recursively that to merge sort and finally, this is the merge operation you have to do. So, this is most important and algorithm we have already given in the last slides that you can follow, otherwise I can give you one logic, one implementation of these techniques first merging and then calling of merging towards the merge, toward implementation.

(Refer Slide Time: 3:29)

Example 55.1: Programming for Merge sort (defining a class for a collection)

```
/* This program defines a generic class to store a collection. */
class MergeSort<T extends Comparable<T>> {
    void merge(T arr[], int l, int m, int r, T d1[], T d2[]) {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        /* Create temp arrays */
        T l[] = d1;
        T r[] = d2;

        /* Copy data to temp arrays*/
        for (int i=0; i<n1; ++i)
            l[i] = arr[l + i];
        for (int j=0; j<n2; ++j)
            r[j] = arr[m + 1 + j];
    }
};
```

The diagram shows a horizontal array of length w (width) being split into two sub-arrays of lengths l and r . The middle element is labeled m . The sub-arrays are labeled d_1 and d_2 .

Let us see how I solved the program. So, this is one example that I want to give it. So, this is, generically I declared a class, these are all things it is there. Now, these are two lists, that we record it that means either array that we can declare into two parts. So, basically n_1 and n_2 are the two lists, usually they are of same size anyway, so, this basically is the part, so, m and r , m and r is basically, if that two list is there, so it is basically l , it is r , rightmost and middle m will be calculator, I will tell you how to find the mid value.

Then, so, these basically whenever you call these two, you have to pass it and so, this is the merge actually operation because we have to perform the merge operation on the same list. So, basically, this is the part of the list, and this is another part of the list. This is again sorted out by calling the merge sort recursively, finally when merge sort come back and then it will basically give sorted lists and everything.

So, this is basically the one part as I said here, so these is the, these are one part and this is our another part. So, this is our d_1 part we can say and this is our d_2 part. Now they are basically on the same array that is why I just made it this kind of arrangement. Anyway, so these part, this is basically the, we can store this in 2 different array also copy and then that is what the practice we usually do.

So, make them copy into two lists actually they are of type same t , then we can do it here basically, how these are just copy, the part of the list into two sub list that we have done,

otherwise you can do on the same list also that logic is little bit, you have to maintain it a little bit cleverly, otherwise you can make into some error.

(Refer Slide Time: 5:31)

Example 55.1: Programming for Merge sort (merge method)

```
// Continued on...  
  
/* Merge the temp arrays */  
  
// Initial indexes of first and second subarrays  
int i = 0, j = 0;  
  
// initial index of merged subarray array  
int k = 1;  
while (i < n1 && j < n2) {  
    if (L[i].compareTo(R[j]) <= 0) {  
        arr[k] = L[i];  
        i++;  
    }  
    else {  
        arr[k] = R[j];  
        j++;  
    }  
    k++;  
}  
  
// Continued to next...
```

```
// Continued on...  
  
/* Copy remaining elements of L[] if any */  
while (i < n1) {  
    arr[k] = L[i];  
    i++;  
    k++;  
}  
  
/* Copy remaining elements of R[] if any */  
while (j < n2) {  
    arr[k] = R[j];  
    j++;  
    k++;  
}  
  
// Continued to next...
```

NPTEL Online Certification Courses
IIT Kharagpur

And here is the code that I have given here, continuing the same procedure, how we can compare the two elements those are required to compare this there, because merging is based on comparison is there. So, here we just copy, here is, this is basically the merge operation, that we can take from one element to another element and then putting into the list there. And it is basically at the end, one list can be empty and other lists may contain some data. So, basically copy element we there. So, this basically the idea about merge operation, then we can think about how the merge sort technique can be implemented.

(Refer Slide Time: 6:10)

Example 55.1: Programming for Merge sort (recursive routine)

```
// Continued on...  
  
// Main function that sorts arr[] using merge()  
void mergeSort(int arr[], int l, int r, int d1[], int d2[]) {  
    if (l < r) {  
        // Find the middle point  
        int m = (l+r)/2;  
        // Sort first and second halves  
        mergeSort(arr, l, m, d1, d2);  
        mergeSort(arr, m+1, r, d1, d2);  
        // Merge the sorted halves  
        merge(arr, l, m, r, d1, d2);  
    }  
} // Continued to next...
```

```
// Continued on...  
  
/* A utility function to print array of size n */  
void printArray(int arr[]) {  
    int n = arr.length;  
    for (int i=0; i<n; ++i)  
        System.out.print(arr[i] + " ");  
    System.out.println();  
} // End of the program  
// Continued to next...
```

So, here is the code for writing the merge sort method here. And this is basically quick sort increments and these are termination condition, we call the merge sort for the two parts dividing this one, here is basically you see how we find the mid location. So, this is the simple logic. And then once it is there, we call this merge operation, this is the logic that we have discussed, the same procedure we follow here. And finally, this is an auxiliary method to print the elements.

So, that is all about the merge sort implementation, is a programming, logic, you have to follow it a little bit, take your own time, because for understanding others programs, sometimes it is very difficult. So, so, that may take some time anyway, you take your own time to understand the code.

(Refer Slide Time: 7:09)

Example 55.1: Programming for Merge sort (main method)

```
// Continued on...  
  
class MergeSortDemo {  
    public static void main(String args[]) {  
        Integer arr[] = {12, 11, 13, 5, 6, 7};  
        MergeSort ob = new MergeSort();  
        System.out.println("Given Array");  
        ob.printArray(arr);  
  
        Integer[] d1 = new Integer[]; +  
        Integer[] d2 = new Integer[];  
        ob.mergeSort(arr, 0, arr.length-1, d1, d2);  
  
        System.out.println("\nSorted array");  
        ob.printArray(arr);  
    }  
}
```

NPTEL Online Certification Courses
IIT Kharagpur

And here is the demo program. So, this is a demo program that how the merge sort works, we have considered this list as an input, and then we call the merge sort method. So, it return the elements, list and then we just print the element here. So here, we just create an object of this program, so that is why ob is basically calling the merge sort and then we can print the elements. And, so, there is another example this is basically to test some merge actually.

So, you can test that how the merge work or for some work also that you can think about it. So, these are different way that you can think about, how the merge operation can works. So, take your own time to understand the code, you can initially run the code that, you can it is working that, then you can finally practice it.

(Refer Slide Time: 8:26)

The `sort()` method sorts an array so that it is arranged in ascending order. The `sort()` method has two versions. The first version, shown here, sorts the entire array:

Method	Description
<code>static void sort(<T> array[])</code>	The methods sorts an array in ascending order.

```
static void sort(byte array[] )
static void sort(char array[] )
static void sort(double array[] )
static void sort(float array[] )
static void sort(int array[] )
static void sort(long array[] )
static void sort(short array[] )
static void sort(Object array[] )
static void sort(T array[] , Comparator<? super T> c)
```

Note: Array of type boolean is not applicable to sort a method.

NPTEL Online Certification Courses
IIT Kharagpur

Now, finally, let us come to the discussion about the utility, so far the sorting method is concerned in Java collection framework, there are, there is a collection, the name of the collection is arrays, in the, this collection array only implement the sorting method, there is no other collections which has its own sort implementation, it is, it may be a little bit peculiar, but it is actually it is, there is a purpose because any collection whether it is a array list or is a pass or a tree or any type you can convert into arrays. That already I have given example how a collection can be converted into other collection.

So, using this concept, you can just convert into an array then finally, you can call your what is called the sorting method there. Unlike other things that is, possible that for I do not know why, the different collection they did not include the sorting method in its framework there anyway, but this is the only thing is that, they wanted to make it one implementation and again which sorting algorithm that the Java collection framework implements that is also not known to us.

So, possibly it implement the quick sort technique, because quick sort is the most efficient sorting technique other than, if it is not sorted and assuming that list that needs to be sorted, usually rare to have in the ascending order, that is why. Now so far, this utility is concerned, as I told you it is declared in arrays and in the class arrays, there is one method called sort method, possibly we have, you can remember.

This class array is also declared one searching algorithm that is binary search, only the method that is that implements a binary search algorithm because it is the most proven and standard algorithm for searching like this also sorting, quick sort is implemented possibly it, sort this algorithm sort the elements into ascending order. Now, so, far sorting is concerned, it can sort any type of data.

Now, here you see the sorting algorithm that the different methods are there, is basically you can pass argument to the sort method here array of bytes. So, here it basically is sorting the binary numbers basically, sorting array of characters also you can use it and is a double, float, integer, long, sort. So, these are the all numeric types that you can sort using this method.

(Refer Slide Time: 11:15)

The slide is titled "Methods of class Arrays for sorting". It contains the following text: "The `sort()` method sorts an array so that it is arranged in ascending order. The `sort()` method has two versions. The first version, shown here, sorts the entire array:"

Method	Description
<code>static void sort(<T> array[])</code>	The methods sorts an array in ascending order.

```
static void sort(byte array[])
static void sort(char array[])
static void sort(double array[])
static void sort(float array[])
static void sort(int array[])
static void sort(long array[])
static void sort(short array[])
static void sort(Object array[])
static void sort(T array[], Comparator<? super T> c)
```

Note: Array of type boolean is not applicable to sort a method.

NPTEL Online Certification Course
IIT Kharagpur

And then there are some sorting method also, you note it here object, object mean any user defined type, you can implement. But whenever this object you want to use it for some own your type, then you have to override compared to, otherwise it will fail, it will not work for you. So, here the object is an argument type, this means that this method sort can be invoked for any type of data, any user defined data or data.

So, for your define type of data you have to over write the compareTo method that you should do and it also, so, the generically the, this sorting method is like this. So, it is defined then generally it is like this, anyway, so, these are the concept that you can think about it. So, how the sorting it works. Now, we can have certain demonstration about how we can call it.

(Refer Slide Time: 12:14)

Methods of class Arrays for sorting

JDK 8 adds several new methods to Arrays. Perhaps the most important is `parallelSort()` because it sorts, into ascending order, portions of an array in parallel and then merges the results.

Method	Description
<code>static void parallelSort(<T> array[])</code>	The methods sorts an array in ascending order.

```
static void parallelSort(byte array[])
static void parallelSort(char array[])
static void parallelSort(double array[])
static void parallelSort(float array[])
static void parallelSort(int array[])
static void parallelSort(long array[])
static void parallelSort(short array[])
static void parallelSort(T array[], Comparator<? super T> c)
```

NPTEL Online Certification Courses
IIT Kharagpur

I can, before going to discuss about demonstration in this class arrays, there is also one technique on method is defined, the method is called `parallelSort`. This is the most efficient sorting ways, but it basically it, it is possible in Java, this is because Java can follow multi threading, for example, in case of divide and conquer, you can do many operations in parallel, because here the recursion, two recursive routine can be executed in parallel, which again recurs up to recursion. So, 4 recursion can be endured in parallel. So, if you want to sort very large set, then definitely you can use the `parallelSort` method to sort it, for smaller set, you cannot find it, but for large set.

Now, to test very large sort, I can suggest you to create a very set of large numbers of random numbers into a file, maybe say 20,000 numbers, that are numbers you can generate within a certain range and then you can call this `parallelSort`. Now, like simple sort, this `parallelSort` has this byte array, you can character array, double, float, integer, long, short or any other type of user defined array also you can use it here. Because it is also support user defined type.

So, this `parallelSort` is basically equivalent to `(())(1:47)`, but only it is basically to do some parallel programming, using threading and it solve it and it is an efficient, usually this method will be called if you want to sort very large data, whenever you have to develop some application where you have to deal with very large set, then definitely you can call `parallelSort` or any collection.

So, if it is any collection, then you can convert this collection, copy this collection using only one statement that you can copy as list, into arrays and then you can get the element and then array can be there. So, then you can use it. Anyway, so, this basically, the idea about the two different methods, which are defined in the class arrays. Arrays is defined in Java collection framework, for which you have to import java dot util, if you, java dot util while you are writing program. Now let us have a demo so that you can understand about how we can call it.

(Refer Slide Time: 14:45)

```
/* The following method demonstrates the sorting using Arrays's sort method.
 */
import java.util.*;
class ArraysSortDemo {
    public static void main(String args[]) {
        // Allocate and initialize array.
        int array[] = new int[10];
        for(int i = 0; i < 10; i++)
            array[i] = -3 + i; // Alternatively, you can load random numbers

        System.out.print("Original contents: ");
        System.out.println(array);
        Arrays.sort(array);
        System.out.print("Sorted: ");
        System.out.println(array);
    }
}
```

So, here basically I am giving the demonstration about using the sort method, array sort. And we declare an array of integers here. And then in this array, I store some numbers of my own. So, this is basically my use numbers. So, it is basically negative numbers, all are negative numbers 0 to 27, minus 27 is a number. Alternatively, you can load random numbers also, you can just call a random math dot random method and then you can initialize this array that also you can do it.

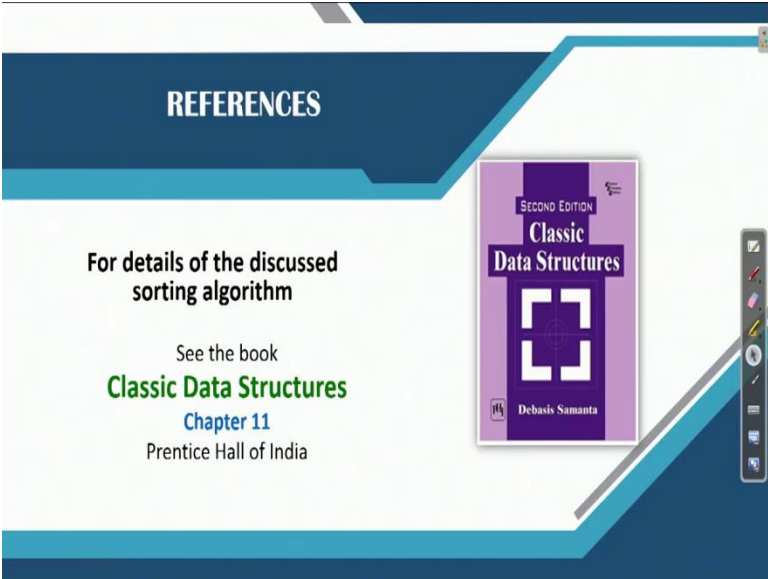
And one important point that you should note that in case of arrays, we do not have to store as a collection rather, we can just simply pass the primitive array that means integer array here, because the array sort method takes the argument as a primitive data type, not that integer object you have to pass it. So, this is a good point that you can note it and here is basically, you print the array before 30, then call the sort method on the array, it basically sort, then you can print after the sort. That is all.

So, this basically give a idea about how the short method which is defined in arrays class can be used in your program. And here again, thus, again, you can write, rewrite the program by writing parallel sort here, also you can run it, but that you can get the impact if you take very large elements stored in the array. So that, you can try of your own. So, these are the different methods, parallelSort and sort, only two methods are defined in Java collection framework to sort any elements.

And another question is that in different collection, if you see, they do not require any sorting explicitly defined, because there are some methods already store the collection, I mean, maintain the collection in a sorted fashion. Now, for example, array list also you can make into stored the element in a sorted fashion, or either linked list or even the sorted tree or any other has, sorted has map also, there are basically automatically store the element in the sorted.

So explicitly, they do not define any sorted sorting method for that, except this arrays's class, which basically includes the sort method defined in it. And, so this is now, so this is basically the idea about the sorting using JCF.

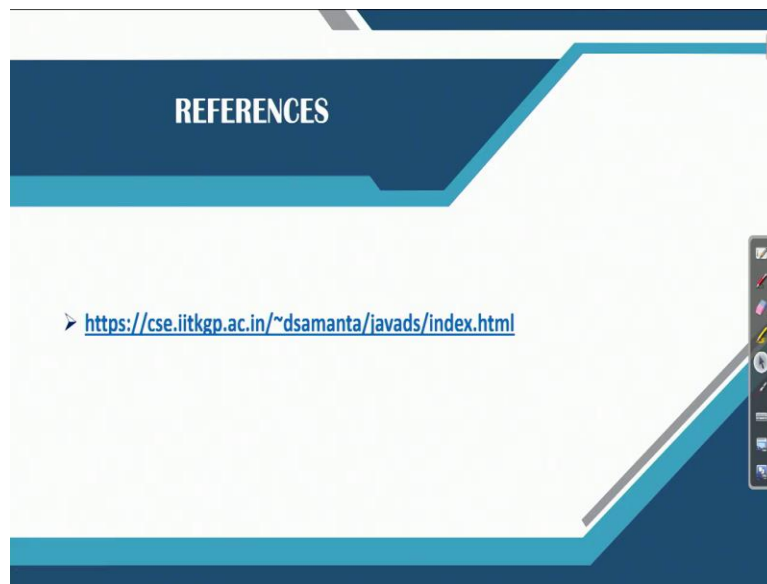
(Refer Slide Time: 17:53)



The slide features a dark blue header with the word "REFERENCES" in white. Below the header, the text reads: "For details of the discussed sorting algorithm", "See the book", "Classic Data Structures", "Chapter 11", and "Prentice Hall of India". To the right of the text is a book cover for "Classic Data Structures" by Debasis Samanta, Second Edition, published by Prentice Hall. The book cover is purple and white with a stylized geometric design. The slide is presented in a software interface with a toolbar on the right side.

There is no much information about sorting technique, there in Java collection framework.

(Refer Slide Time: 17:59)



And that is all I, for many other programs, which I have used in this discussion and the different sorting algorithms which I have covered in this course you can find from this link, this link is very useful for you. And, thank you. Thank you very much.