CM1602 : Data Structures and Algorithms for AI

6. Searching algorithms and Sorting algorithms

Lecture 6 - Part 2 | R. Sivaraman

MODULE CONTENT

Lecture	Topic
Lecture 01	Introduction to Fundamentals of Algorithms
Lecture 02	Analysis of Algorithms
Lecture 03	Array and Linked Lists
Lecture 04	Stack
Lecture 05	Queue
Lecture 06	Searching algorithms and Sorting algorithms
Lecture 07	Trees
Lecture 08	Maps, Sets, and Lists
Lecture 09	Graph algorithms

Learning Outcomes

- LO1 : Describe the fundamental concepts of algorithms and data structures.
- LO4 : Adapt and extend algorithms to real-world problems and address implementation requirements.
- On completion of this lecture, students are expected to be able to:
  - Describe and Implement and analyze Bubble Sort, Selection Sort, and Merge Sort
  - Analyze the performance of Sorting algorithms

Bubble Sort

Bubble Sort

- Complexity of Bubble sort is O(N^2)
- Move the largest value to the end using pair-wise comparisons and swapping
- Repeat the same process for all the elements

Bubble Sort

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
42	77	35	12	101	5

1	2	3	4	5	6
42	35	77	12	101	5

1	2	3	4	5	6
42	35	12	77	101	5

Bubble Sort

1	2	3	4	5	6
42	35	12	77	101	5

No need to swap

1	2	3	4	5	6
42	35	12	77	5	101

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

Bubble Sort

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
42	35	12	77	5	101

1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

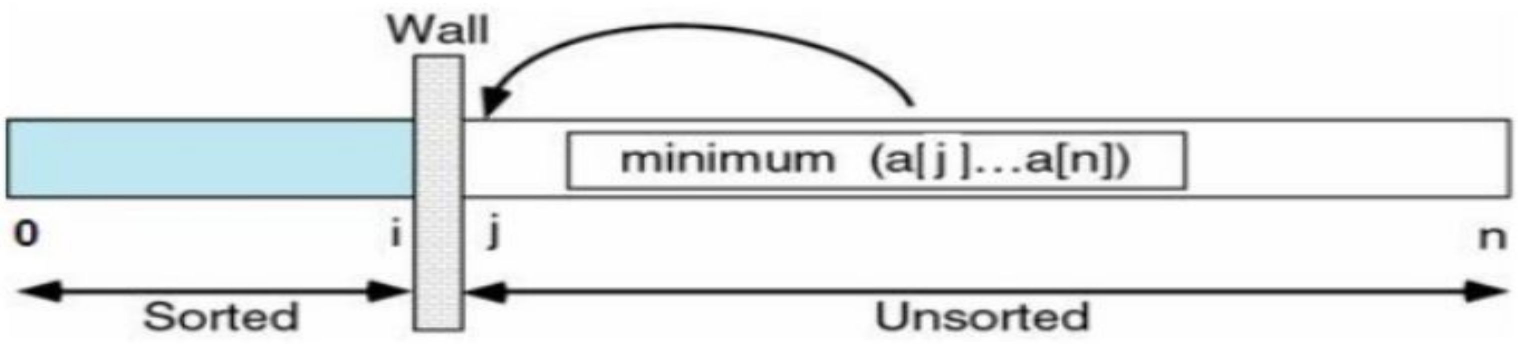
Bubble Sort

```
public void bubbleSort(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
            {
                // swap arr[j+1] and arr[j]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

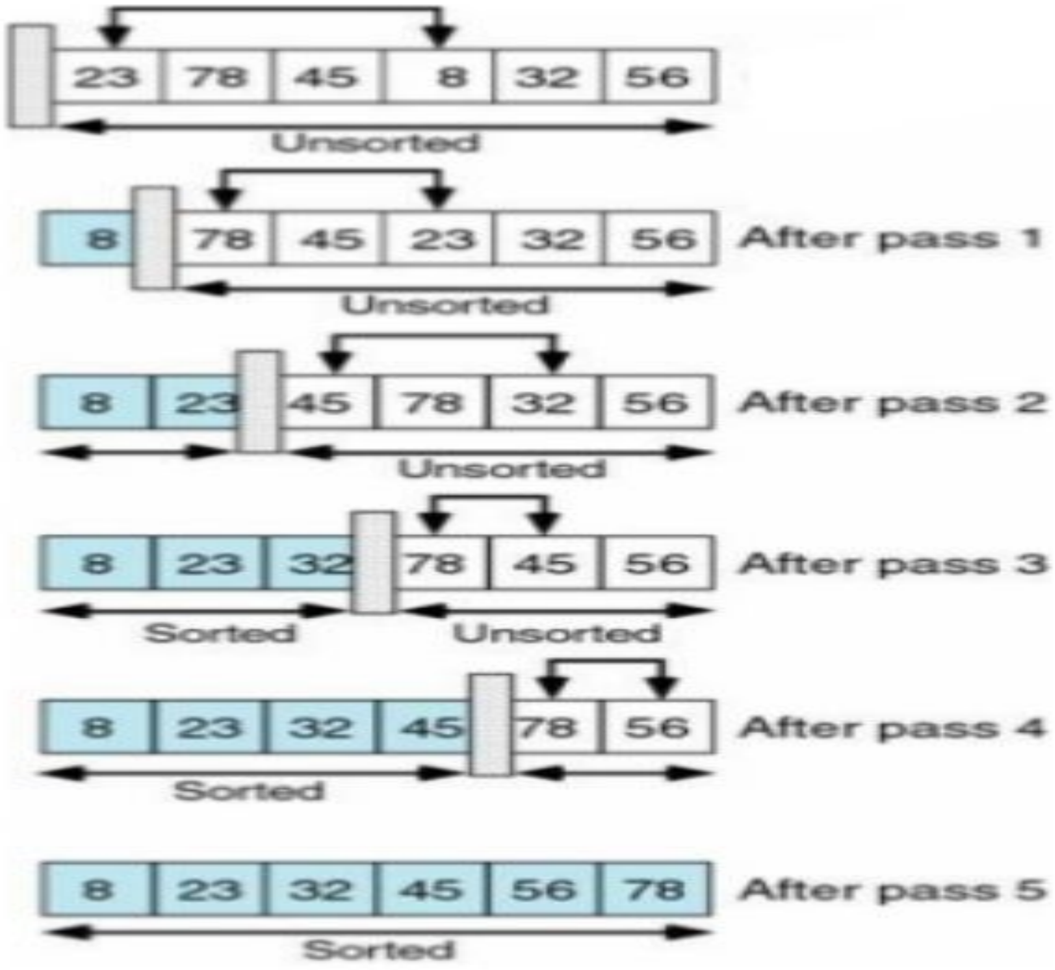
Selection Sort

Selection Sort

- Complexity of Selection sort is O(N^2)
- First find the smallest value
- Move the smallest value to the start



Selection Sort



Selection Sort

Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1 LOC=4	77	33	44	11	88	22	66	55
K=2 LOC=6	11	33	44	77	88	22	66	55
K=3 LOC=6	11	22	44	77	88	33	66	55
K=4 LOC=6	11	22	33	77	88	44	66	55
K=5 LOC=8	11	22	33	44	88	77	66	55
K=6 LOC=7	11	22	33	44	55	77	66	88
K=7 LOC=4	11	22	33	44	55	66	77	88

Sorted	11	22	33	44	55	66	77	88

Selection Sort

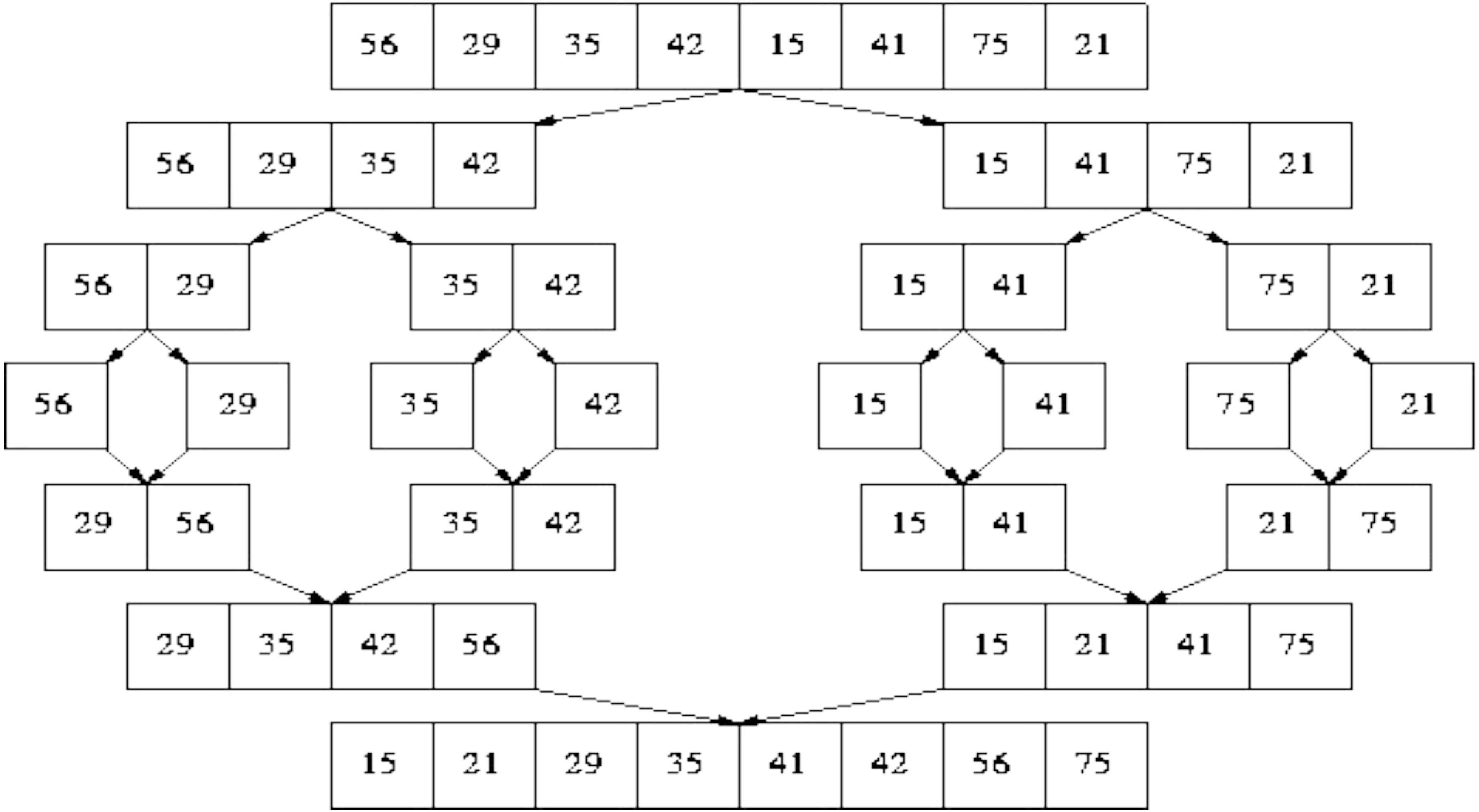
```
void selectionSort(int arr[])
{
    int n = arr.length;
    // One by one move boundary of unsorted subarray
    for (int i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        int min_idx = i;
        for (int j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        // Swap the found minimum element with the first
        // element
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}
```

Merge Sort

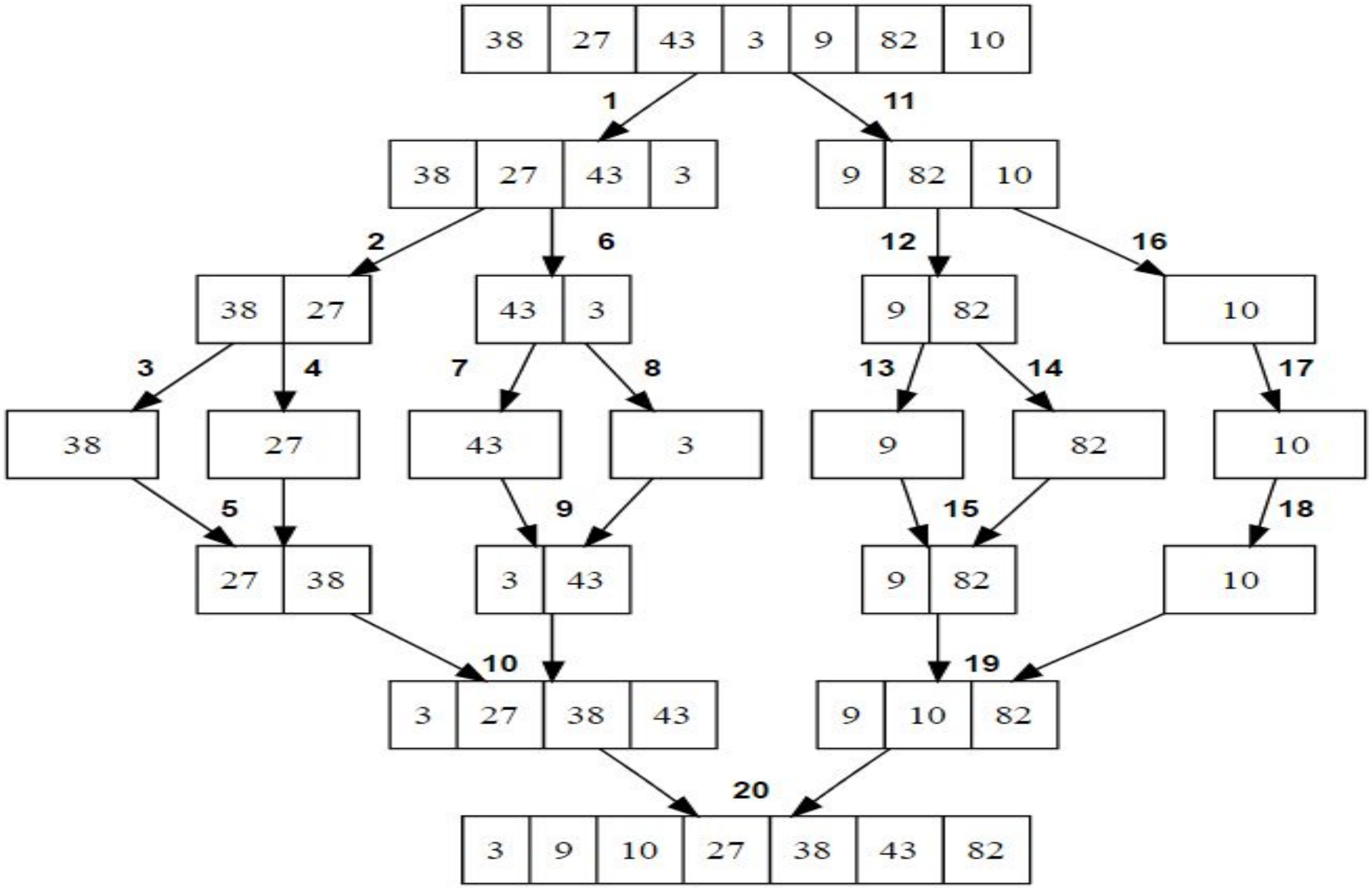
Merge Sort

- Complexity is O(NlogN)
- Merge sort uses Divide and Conquer strategy
- Merge method merges two sorted arrays and produce one sorted array
- Steps:
    - Divide the unsorted collection into two
    - Keep on dividing until the sub-arrays only contain one element
    - Then merge the sub-problem solutions together

Merge Sort


56 29 35 42 15 41 75 21
56 29 35 42
15 41 75 21
56 29
35 42
15 41
75 21
56
29
35
42
15
41
75
21
29 56
35 42
15 41
21 75
29 35 42 56
15 21 41 75
15 21 29 35 41 42 56 75

Merge Sort



38 27 43 3 9 82 10
1
11
38 27 43 3
9 82 10
2
6
12
16
38 27
43 3
9 82
10
3
4
7
8
13
14
17
38
27
43
3
9
82
10
5
9
15
18
27 38
3 43
9 82
10
10
19
3 27 38 43
9 10 82
20
3 9 10 27 38 43 82

Merge Sort – Merge Method

```
void merge(int arr[], int l, int m, int r)
{
    // Find sizes of two subarrays to be merged
    int n1 = m - l + 1;
    int n2 = r - m;

    /* Create temp arrays */
    int L[] = new int[n1];
    int R[] = new int[n2];

    /*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];

    /* Merge the temp arrays */

    // Initial indexes of first and second subarrays
    int i = 0, j = 0;
```

Merge Sort – Merge Method

```
        // Initial index of merged subarry array
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            }
            else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        /* 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++;
        }
    }
```

Merge Sort

```
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Find the middle point
        int m =l+ (r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}
```

For Additional Reading

- Quick Sort