Northern University, Nowshera

Application of Information and Communication Technologies (AICT)

(Week 15) Lecture 29 & 30

Objectives: **Learning objectives of this lecture are**

- **Merge Sort**

Text Book & Resources: Introduction to Computers 6th International Edition, Peter, N. McGraw-Hill

Merge Sort

Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sub list consists of a single element and merging those sub lists in a manner that results into a sorted list.

Merge Sort follows the rule of Divide and Conquer to sort a given set of numbers/elements, recursively, hence consuming less time.

Divide and Conquer:
If we can break a single big problem into smaller sub-problems, solve the smaller sub-problems and combine their solutions to find the solution for the original big problem, it becomes easier to solve the whole problem.

Let's take an example, Divide and Rule.

When Britishers came to India, they saw a country with different religions living in harmony, hardworking but naive citizens, and unity in diversity, and found it difficult to establish their empire. So, they adopted the policy of Divide and Rule. Where the population of India was collectively a one big problem for them, they divided the problem into smaller problems, by instigating rivalries between local kings, making them stand against each other, and this worked very well for them.

Well that was history, and a socio-political policy (Divide and Rule), but the idea here is, if we can somehow divide a problem into smaller sub-problems, it becomes easier to eventually solve the whole problem.

Northern University, Nowshera

Application of Information and Communication Technologies (AICT)

Idea:

Divide the unsorted list into sub lists, each containing element.

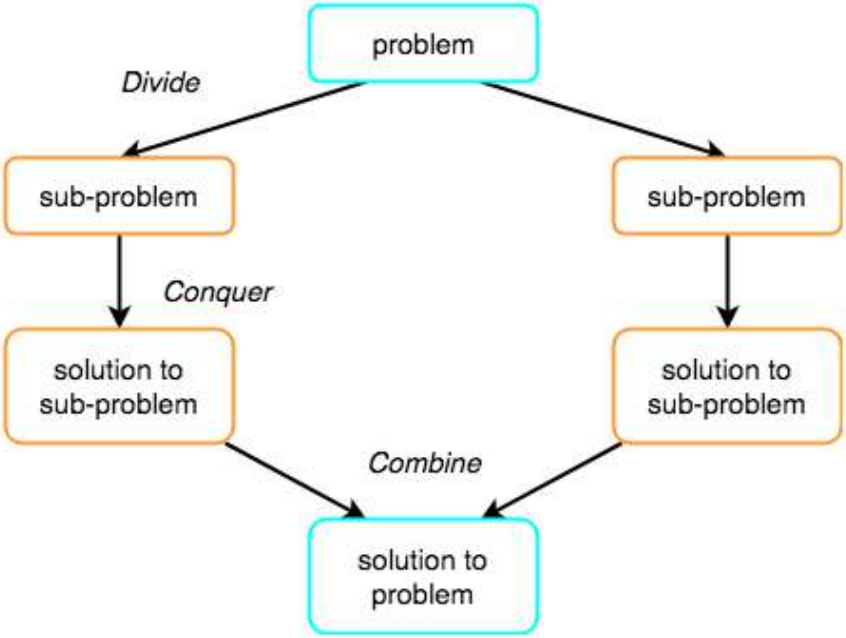
Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. Will now convert into lists of size 2.
Repeat the process till a single sorted list of obtained.

While comparing two sub lists for merging, the first element of both lists is taken into consideration. While sorting in ascending order, the element that is of a lesser value becomes a new element of the sorted list. This procedure is repeated until both the smaller sub lists are empty and the new combined sub list comprises all the elements of both the sub lists.

In Merge Sort, the given unsorted array with n elements, is divided into n sub arrays, each having one element, because a single element is always sorted in itself. Then, it repeatedly merges these sub arrays, to produce new sorted sub arrays, and in the end, one complete sorted array is produced.

The concept of Divide and Conquer involves three steps:

1. Divide the problem into multiple small problems.
2. Conquer the sub problems by solving them. The idea is to break down the problem into atomic sub problems, where they are actually solved.
3. Combine the solutions of the sub problems to find the solution of the actual problem.



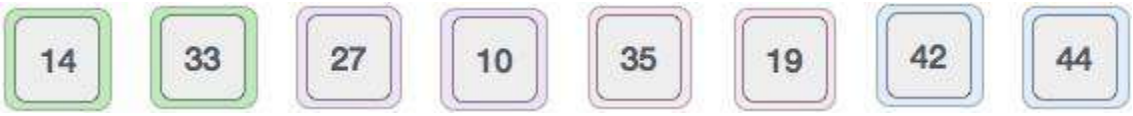
How Merge Sort Works?

To understand merge sort, we take an unsorted array as the following:

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

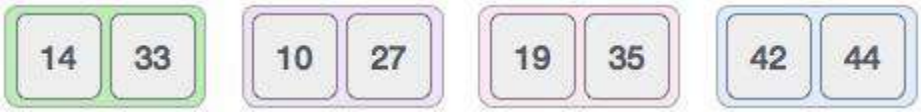
We further divide these arrays and we achieve atomic value which can no more be divided.



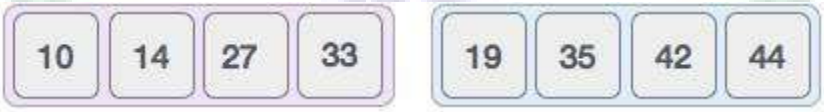
Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions.

We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.



In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

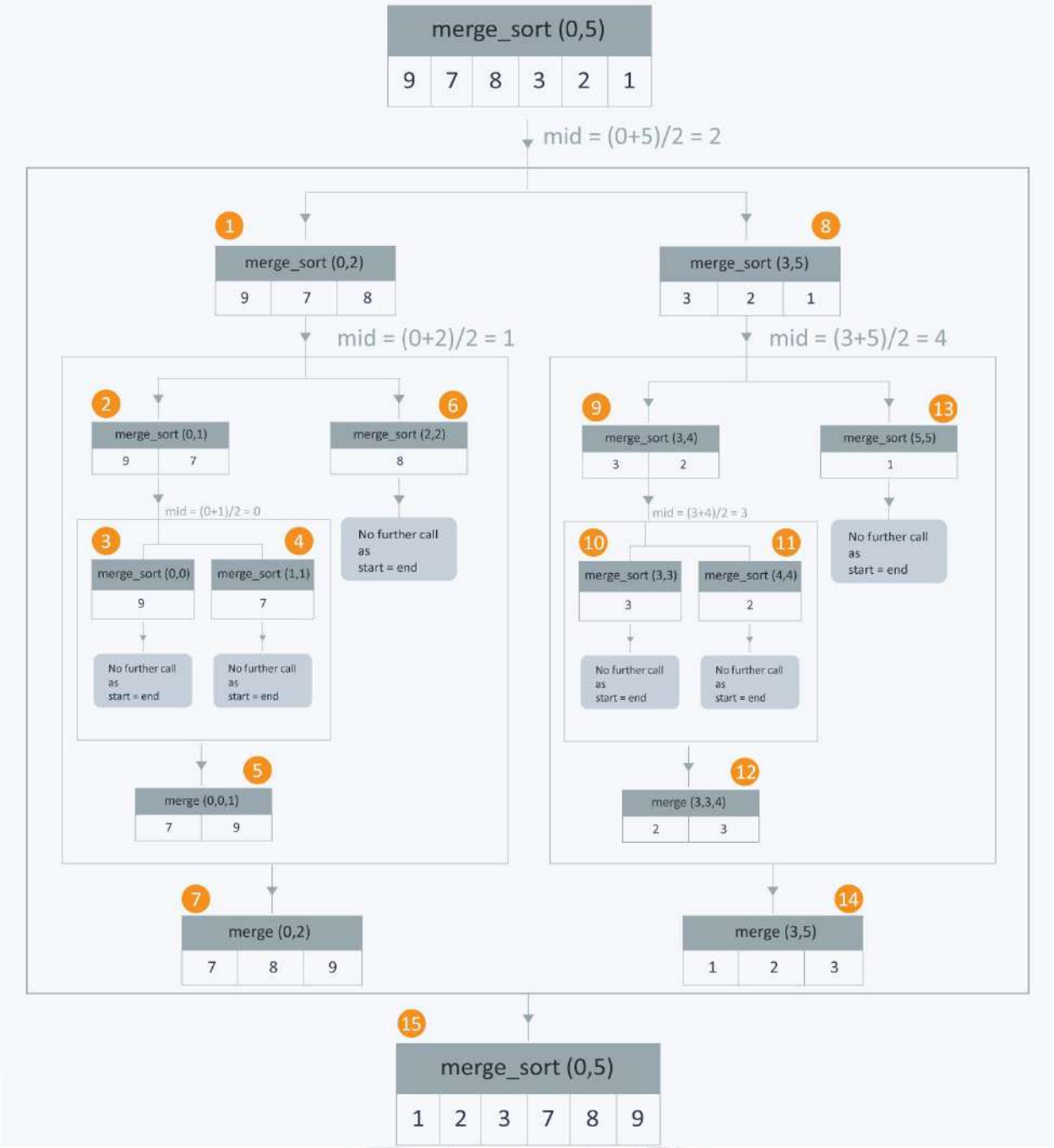


After the final merging, the list should look like this:

Northern University, Nowshera

Application of Information and Communication Technologies (AICT)

Example:



From the image above, at each step a list of size M is being divided into 2 sub lists of size M/2, until no further division can be done. To understand better, consider a smaller array A containing the elements (9, 7, 8).

At the first step this list of size 3 is divided into 2 sub lists the first consisting of elements (9, 7) and the second one being (8). Now, the first list consisting of elements (9, 7) is further divided into 2 sub lists consisting of elements (9) and (7) respectively.

As no further breakdown of this list can be done, as each sub list consists of a maximum of 1 element, we now start to merge these lists. The 2 sub-lists formed in the last step are then merged together in sorted order using the procedure mentioned above leading to a new list (7, 9). Backtracking further, we then need to merge the list consisting of element (8) too with this list, leading to the new sorted list (7, 8, 9).

Divide and Conquer Strategy:
Using the Divide and Conquer technique, we divide a problem into sub problems. When the solution to each sub problem is ready, we 'combine' the results from the sub problems to solve the main problem.

Suppose we had to sort an array A. A sub problem 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 sub array 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 sub arrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these sub arrays and try to sort them.

Combine:
When the conquer step reaches the base step and we get two sorted sub arrays 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 sub arrays A[p..q] and A[q+1, r].

Example:

As we have already discussed that merge sort utilizes divide-and-conquer rule to break the problem into sub-problems, the problem in this case being, sorting a given array.

In merge sort, we break the given array midway, for example if the original array had 6 elements, then merge sort will break it down into two sub arrays with 3 elements each.

But breaking the original array into 2 smaller sub arrays is not helping us in sorting the array.

So we will break these sub arrays into even smaller sub arrays, until we have multiple sub arrays with single element in them. Now, the idea here is that an array with a single element is already sorted, so once we break the original array into sub arrays which has only a single element, we have successfully broken down our problem into base problems.

And then we have to merge all these sorted sub arrays, step by step to form one single sorted array.

Let's consider an array with values {14, 7, 3, 12, 9, 11, 6, 12}

In merge sort we follow the following steps:

1. We take a variable p and store the starting index of our array in this. And we take another variable r and store the last index of array in it.

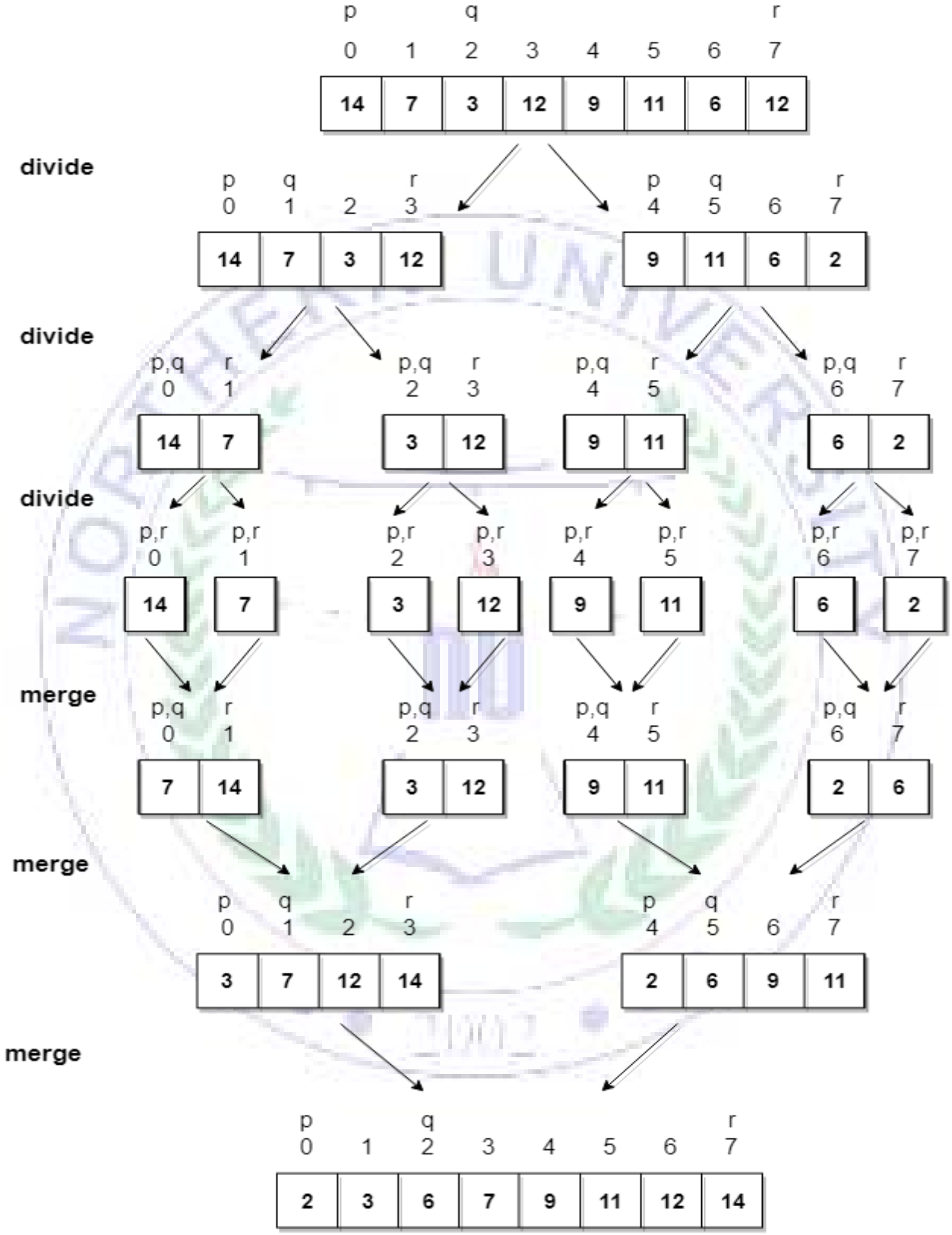
2. Then we find the middle of the array using the formula (p + r)/2 and mark the middle index as q, and break the array into two sub arrays, from p to q and from q + 1 to r index.

3. Then we divide these 2 sub arrays again, just like we divided our main array and this continues.

4. Once we have divided the main array into sub arrays with single elements, then we start merging the sub arrays.

Northern University, Nowshera

Application of Information and Communication Technologies (AICT)



Northern University, Nowshera

Application of Information and Communication Technologies (AICT)

Algorithm:

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.

Step 1 – if it is only one element in the list it is already sorted, return.

Step 2 – divide the list recursively into two halves until it can no more be divided.

Step 3 – merge the smaller lists into new list in sorted order.

Code:

```
void mergeSort(int a[], int p, int r)
{
   int q;
   if(p < r)
   {
      q = (p + r) / 2;
      mergeSort(a, p, q);
      mergeSort(a, q+1, r);
      merge(a, p, q, r);
   }
}
```

```
// function to merge the subarrays
    void merge(int a[], int p, int q, int r)
    {
       int b[5];   //same size of a[]
       int i, j, k;
       k = 0;
       i = p;
       j = q + 1;
       while(i <= q && j <= r)
       {
          if(a[i] < a[j])
          {
             b[k++] = a[i++];    // same as b[k]=a[i]; k++; i++;
          }
          else
          {
             b[k++] = a[j++];
          }
       }

       while(i <= q)
       {
          b[k++] = a[i++];
       }

       while(j <= r)
       {
          b[k++] = a[j++];
       }

       for(i=r; i >= p; i--)
       {
          a[i] = b[--k];  // copying back the sorted list to a[]
       }
    }
```

```
// function to print the array
        void printArray(int a[], int size)
        {
           int i;
           for (i=0; i < size; i++)
           {
              cout<<a[i];
           }
           cout<<"\n";
        }

        int main()
        {
           int arr[] = {32, 45, 67, 2, 7};
           int len = sizeof(arr)/sizeof(arr[0]);

           cout<<"Given array: \n";
           printArray(arr, len);

           // calling merge sort
           mergeSort(arr, 0, len - 1);

           cout<<"\nSorted array: \n";
           printArray(arr, len);
           return 0;
        }
```

Video Link:

https://www.youtube.com/playlist?list=PLeby327DX1ci-9YKyTbNqiCnwOf6km5o0