DATA STRUCTURES

By

Dr. Yasser Abdelhamid

الجامعة المصرية
للتعلم الإلكتروني الأهلية

EELU

THE EGYPTIAN E-LEARNING UNIVERSITY

RESOURCES

- http://javatpoint.com/

OUTLINE

- Mergesort

MERGESORT

- **Mergesort algorithm follows divide and conquer approach.**
- **Mergesort algorithm works as follows:**
  - It divides the given list into two equal halves.
  - It recursively calls itself for the two halves.
  - Eventually, the sub-lists can not be furtherly divided, at this point we start combining pieces of the list into one sorted list.

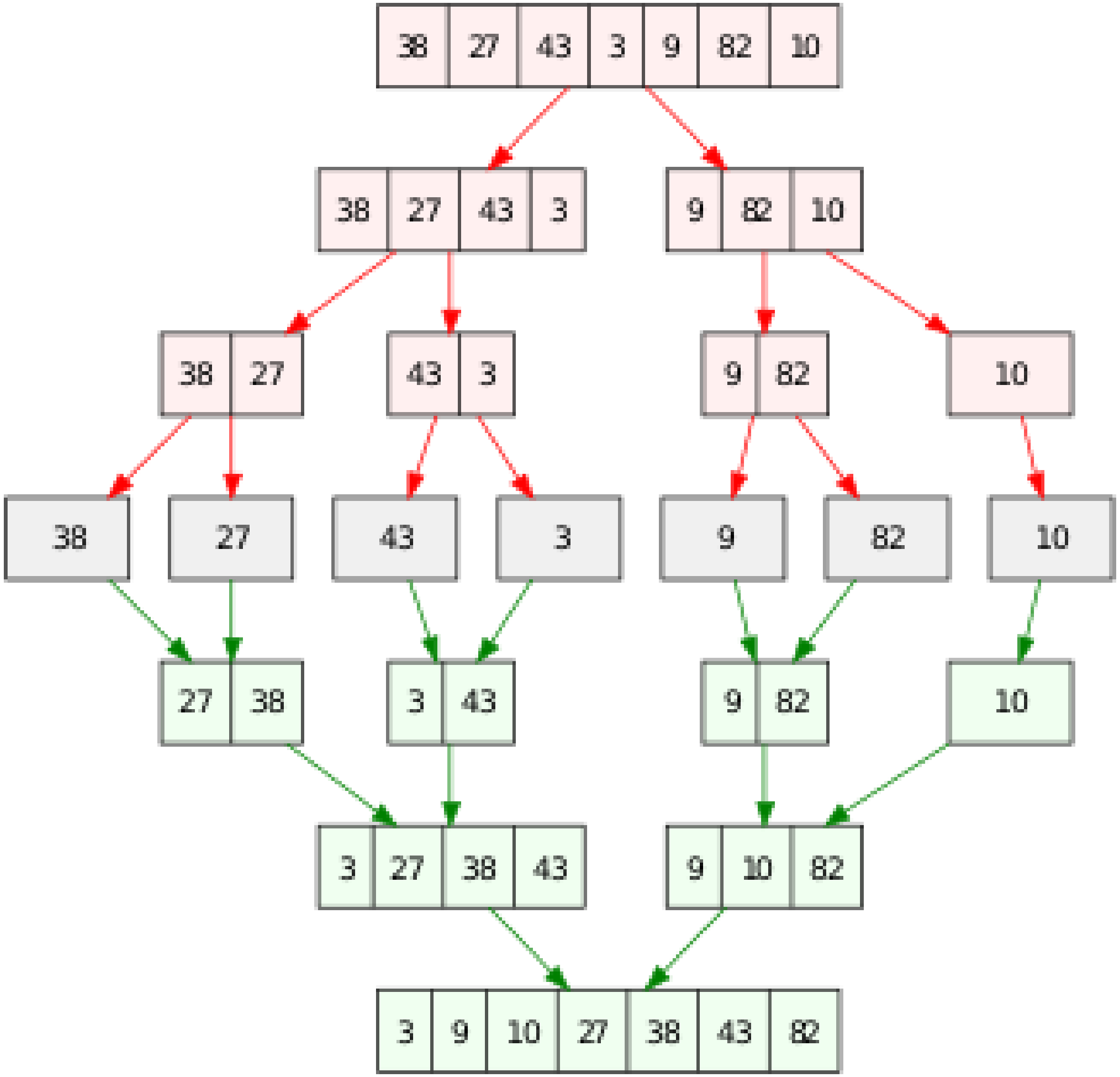
MERGE SORT ALGORITHM

```
void mergeSort(int a[], int start, int end)
{
    if (start < end)
    {
        int mid = (start + end) / 2;
        mergeSort(a, start, mid);
        mergeSort(a, mid + 1, end);
        merge(a, start, mid, end);
    }
}
```

MERGE FUNCTION

- This function merges two **sorted sub-arrays** that are arr[start…mid] and arr[mid+1…end], to build one sorted array arr[start…end].

EXAMPLE



38 27 43 3 9 82 10

38 27 43 3 | 9 82 10

38 27 | 43 3 | 9 82 | 10

38 | 27 | 43 | 3 | 9 | 82 | 10

27 38 | 3 43 | 9 82 | 10

3 27 38 43 | 9 10 82

3 9 10 27 38 43 82

MERGE FUNCTION

```
void merge(int a[], int start, int mid, int end)
{
   int i, j, k;
   int n1 = mid - start + 1;// size of first subarray
   int n2 = end - mid;   // size of second subarray

   int LeftArray[n1], RightArray[n2]; //temp arrays

   /* copy data to temp arrays */
   for (int i = 0; i < n1; i++)
          LeftArray[i] = a[start + i];
   for (int j = 0; j < n2; j++)
         RightArray[j] = a[mid + 1 + j];

   i = 0, /* initial index of first sub-array */
   j = 0; /* initial index of second sub-array */
   k = start;  /* initial index of merged sub-array */
```

```
 while (i < n1 && j < n2)
   {
      if(LeftArray[i] <= RightArray[j])
      {
         a[k] = LeftArray[i];
         i++;
      }
      else
      {
         a[k] = RightArray[j];
         j++;
      }
      k++;
   }
   while (i<n1)
   {
      a[k] = LeftArray[i];
      i++;
      k++;
   }

   while (j<n2)
   {
      a[k] = RightArray[j];
      j++;
      k++;
   }
}
```

MERGESORT ANALYSIS

- Let T(n) be the running time for an array of n elements
- Mergesort divides array in half and recursively calls itself on the two halves.
- After returning, it merges both halves using a temporary array.
- Each recursive call takes T(n/2) and merging takes O(n)

$$T(n) = 2\,T(n/2) + O(n)$$

Using recursion tree as in quicksort,

$$T(n) = O(n \log n)$$

MERGESORT ANALYSIS

- The time complexity of mergesort is dominated by the merging step.
- In the worst case, the merging step needs to compare each element in the two sub-lists being merged, resulting in a total of n comparisons for each level of recursion.
- Since the recursion depth is log n, the total number of comparisons required is O(n log n).

MERGESORT HAS A STABLE TIME COMPLEXITY

- **Mergesort has a stable time complexity**, meaning that its performance is consistent across different input data types and distributions.
- It is generally considered to be a reliable and efficient sorting algorithm, particularly for large lists or for situations where stability is important.

THANK YOU

الجامعة المصرية
للتعلم الإلكتروني الأهلية

EELU

THE EGYPTIAN E-LEARNING UNIVERSITY