Design and Analysis of Algorithm (DAA)

Divide-and-Conquer

Dr. Dayal Kumar Behera

School of Computer Engineering
KIIT Deemed to be University, Bhubaneswar, India

Divide-and-Conquer

The most-well known algorithm design strategy:

Break the problem into several subproblems.

Solve the subproblem recursively.

Combine the solutions of sub-problems to create a solution to the original problem.

Steps

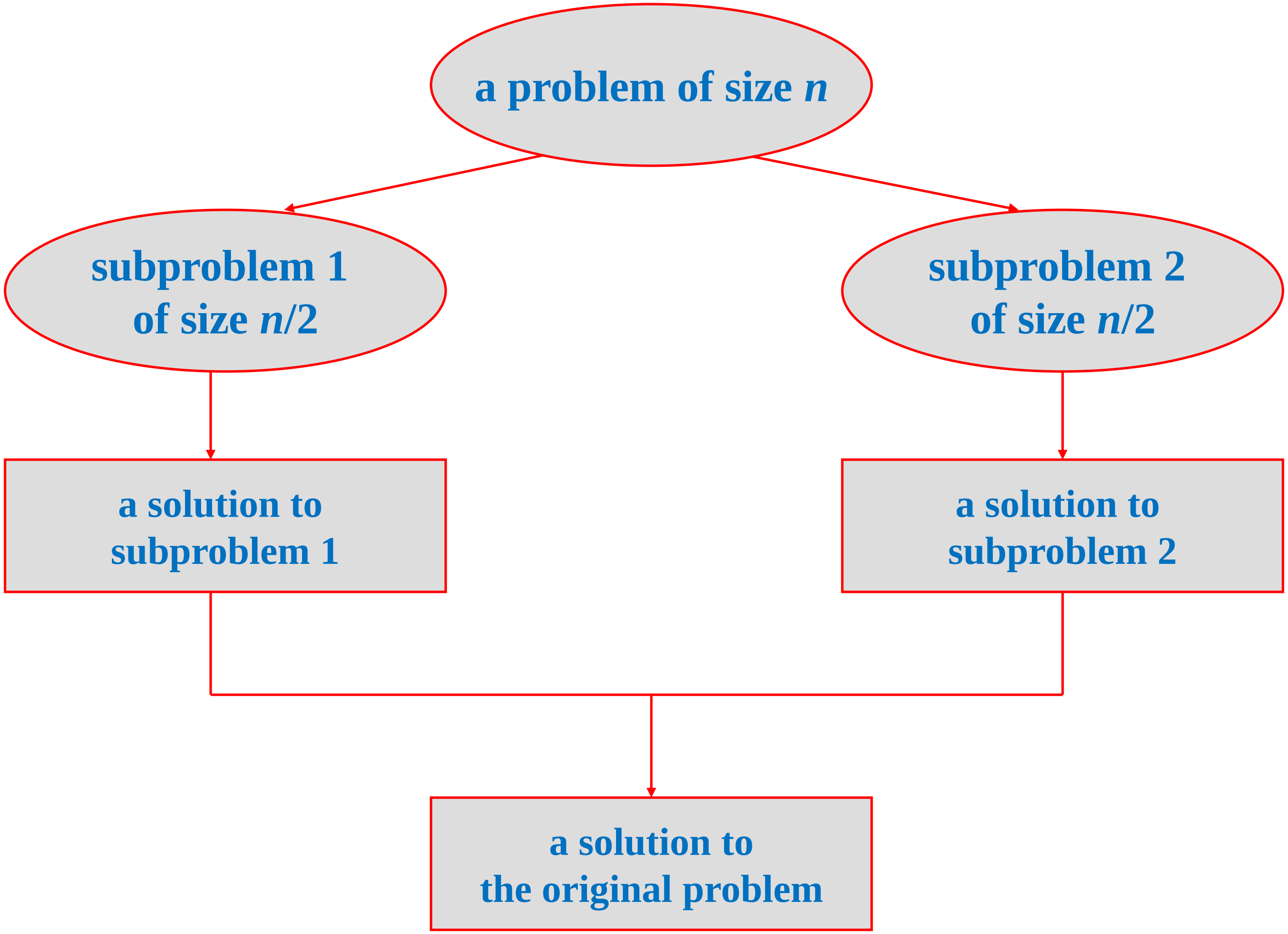
The divide and conquer paradigm involves three steps at each level of recursion

1. **Divide:** Divide the problem into two or more subproblems

2. **Conquer:** Solve subproblems recursively

3. **Combine:** Obtain solution to original (larger) problem by combining solutions of smaller subproblems.

Divide-and-Conquer Technique



Divide-and-Conquer Algorithms

Few examples of Divide-and-Conquer algorithms are

- Merge sort
- Quicksort
- Binary search
- Max-Min Problem

Merge Sort

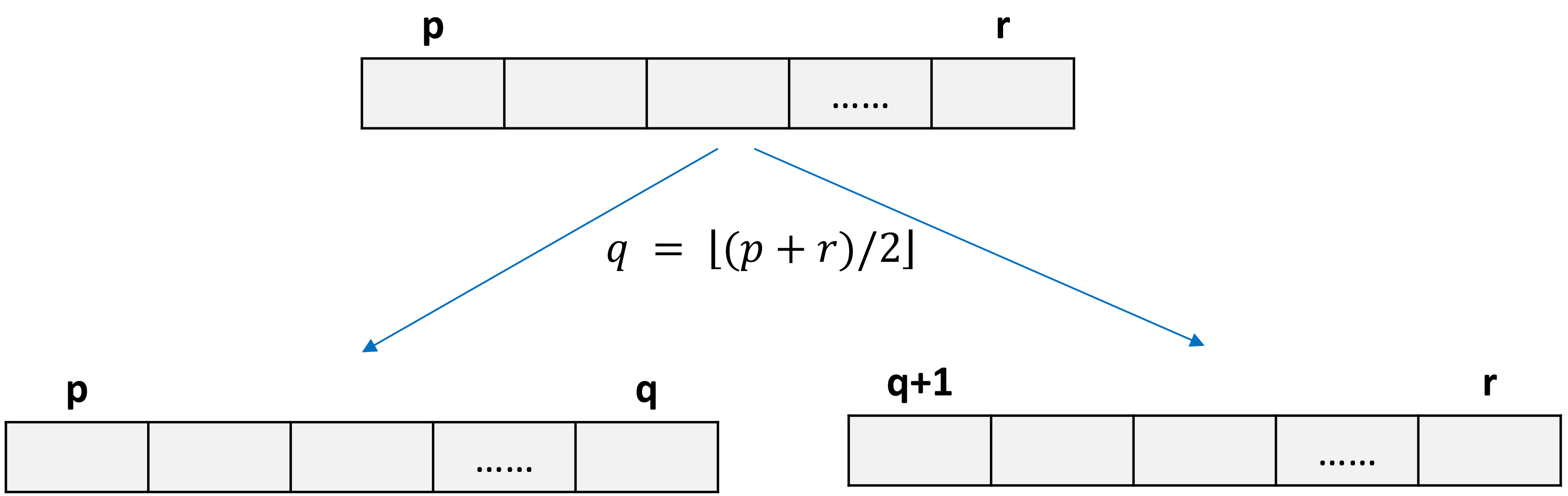
The ***merge sort*** algorithm closely follows the divide-and-conquer paradigm.

- **Divide:** Divide the n-element sequence to be sorted into two subsequences of n/2 elements each.
- **Conquer:** Sort the two subsequences recursively
- **Combine:** Merge the two sorted subsequences to produce the sorted answer.

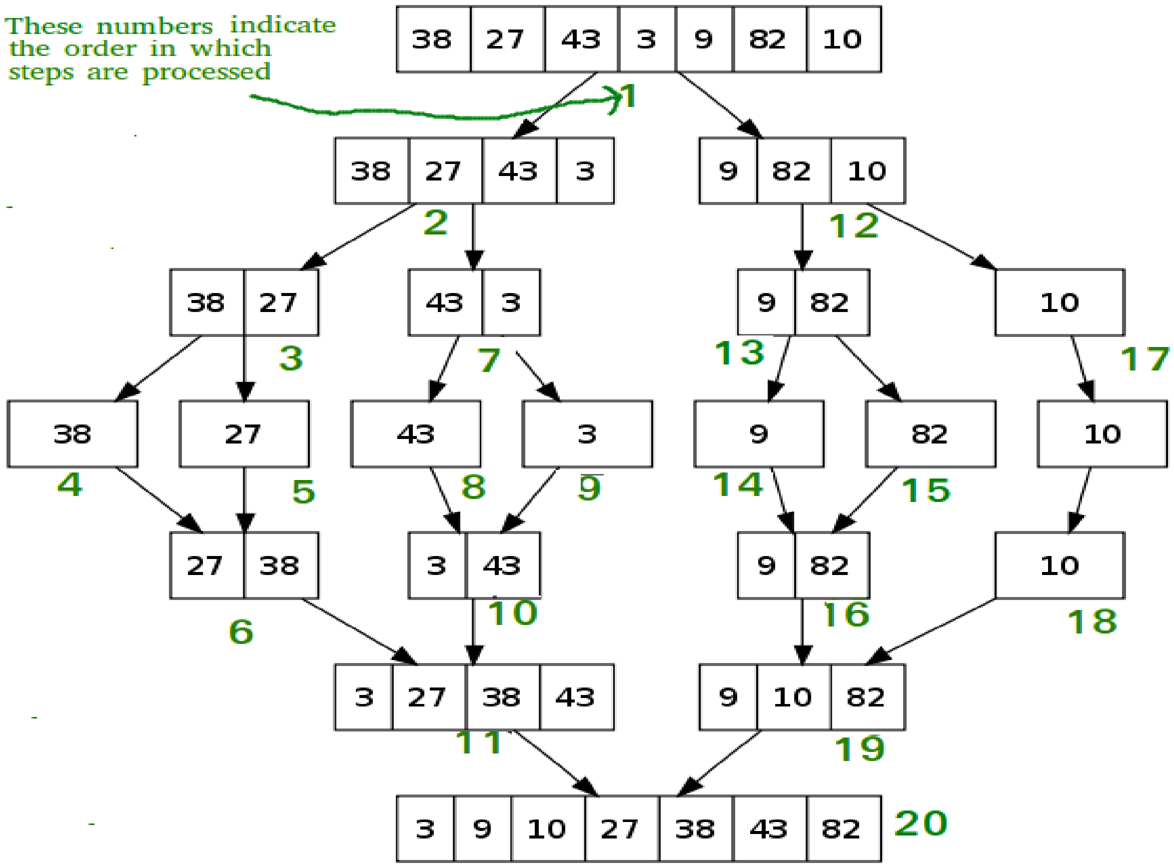
Merge Sort

- **Divide and Conquer Strategy :** Suppose we had to sort an array A. A subproblem 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 subarray 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 subarrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.
- **Combine:** When the conquer step reaches the base step and we get two sorted subarrays 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 subarrays A[p..q] and A[q+1, r].

Merge Sort Example



Merge Sort Example



Merge Sort Algorithm

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Illustration of Merge-Sort

	p			r
	1	**2**	**3**	**4**
A	50	20	40	70

Tracing Tree/Recursion Tree

The tracing tree or recursion tree of recursive function call in Merge Sort for the initial function call **MERGE-SORT(A, 1, 4)** for the example **<50, 20, 40, 70>** is as follows

	p			r
	1	**2**	**3**	**4**
A	50	20	40	70

Analyzing Divide-and-Conquer Algorithms

Generalized form of recurrence for running time of Divide-and-Conquer algorithm is as follows:

$$T(n) = \begin{cases} \theta\,(1) & if\ n \le c \\ \underbrace{a\,T\left(\frac{n}{b}\right)}_{\textbf{Conquer time}} + \underbrace{D(n)}_{\textbf{Divide time}} + \underbrace{C(n)}_{\textbf{Combine time}} & otherwise \end{cases}$$

a : no. of sub-problem

$\frac{n}{b}$: size of each sub-problem

Analysis of Merge Sort

- **Divide:** Computes the middle of the sub-array, takes constant time.
  Thus D(n) = θ (1)

- **Conquer:** recursively solve two sub-problems, each of size (n/2).
  Thus a T(n/2) = 2 T (n/2)

- **Combine:** Merge procedure on an “n” elements sub-array takes θ (n) time.
  Thus C(n) = θ (n)

MERGE-SORT(A, p, r)

if $p < r$
  $q = \lfloor (p + r)/2 \rfloor$ Divide
  MERGE-SORT(A, p, q) Conquer
  MERGE-SORT$(A, q + 1, r)$
  MERGE(A, p, q, r) Combine

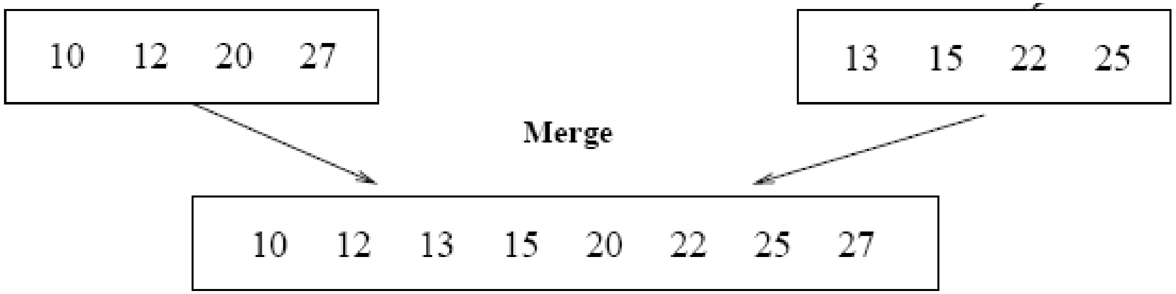
Total running time of merge sort:

$$T(n) = \begin{cases} \theta\,(1) & if\ n = 1 \\ 2\,T\left(\frac{n}{2}\right) + \theta(1) + \theta(n) & if\ n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & if\ n = 1 \\ 2\,T\left(\frac{n}{2}\right) + cn & if\ n > 1 \end{cases}$$

Combine Time/ Time Complexity of Merge

- Initialization (copying into temporary arrays):
  - $\Theta(n_1 + n_2) = \Theta(n)$
- Adding the elements to the final array:
  - n iterations, each taking constant time $\Rightarrow \Theta(n)$
- Total time for Merge:
  - $\Theta(n)$



Time Complexity of Merge Sort

- **Divide:**
  - compute q as the average of p and r: $D(n) = \Theta(1)$
- **Conquer:**
  - recursively solve 2 subproblems, each of size $n/2 \Rightarrow 2T\,(n/2)$
- **Combine:**
  - MERGE on an n-element subarray takes $\Theta(n)$ time $\Rightarrow C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Solve the Recurrence

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

Using Master's Theorem:

Case 2 of Master's Theorem: $T(n) = \Theta(n \lg n)$

Merge Sort - Discussion

- **Running time insensitive of the input:**
  - Merge sort pays no attention to the original order of the list
  - It keeps dividing the list into half until sub-lists of length 1, then start merging

- Therefore, **Average-Case Time Complexity** of Merge Sort is the same as its Worst-Case Time Complexity

- **Advantage:**
  - Guaranteed to run in $\Theta(n\log_2 n)$

- **Disadvantage:**
  - Requires extra space $\approx \Theta(n)$

Task

Check the algorithm given in the tutorial in the below link

https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/tutorial/

Play with the visualization with different problem size

https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/visualize/


Each of your
actions will
have an
impact on your
future.


Once you know
who is walking
with you on your path.
you will never
be afraid.

Thank you