Algorithm Analysis and Design

Pre-Requisites & Course Outcomes

PRE-REQUISITES:
1. Programming Skills
2. Discrete Structures
3. Data Structures

COURSE OUTCOMES (COs):
After completion of this course, the learners will be able to:-

CO #	Detailed Statement of the CO	BT Level	Mapping to PO #
CO1	Demonstrate P and NP complexity classes of the problem.	BTL2	PO1, PO2, PO3
CO2	Apply the concepts of asymptotic notations to analyze the complexities of various algorithms.	BTL4	PO1, PO2, PO3, PO4
CO3	Analyze and evaluate the searching, sorting and tree-based algorithms.	BTL5	PO1, PO2, PO3, PO4, PO5
CO4	Design efficient solutions using various algorithms for given problems.	BTL6	PO1, PO2, PO3, PO4, PO5, PO6, PO10
CO5	Develop innovative solutions for real-world problems using different paradigms.	BTL6	PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO9, PO10,

Syllabus (Unit-II)

- **Divide and Conquer Paradigm: Problem Solving, Comparative Analysis of different Sorting and Searching Techniques, Strassen's Matrix Multiplication Method.**
- **Sorting in linear time: Counting Sort, Bucket Sort and Radix Sort.**
- **String Matching Concept: Naive String-Matching Algorithm, String Matching with Finite Automata, Knuth Morris Pratt Algorithm, The Rabin-Karp Algorithm.**
- **Red Black Trees, Disjoint Set and their Implementation, Medians and Order Statistics.**
- **No. of Hours: 12**

Divide and Conquer Paradigm

Divide and Conquer Paradigm

- **Divide and Conquer** is **a recursive problem-solving approach which break a problem into smaller subproblems, recursively solve the subproblems**, and finally combines the solutions to the subproblems to solve the original problem.
- There are three parts of Divide and Conquer (DAQ) algorithms.
  - Divide the problem into a number of subproblems that are smaller instances of the same problem.
  - Conquer the subproblems by solving them recursively
  - Combine the solutions to the subproblems into the solution for the original problem.

Divide and Conquer Paradigm

- Advantages:
  - Solving difficult problems
  - Algorithm efficiency
  - Parallelism
  - Memory access
- Disadvantage:
  - Slow because of using recursion.

Divide and Conquer: Binary Search

- Example:
  - Binary search:

```
int binarySearch(int arr[], int l, int r, int x)
{
if (r >= l)
{
        int mid = l + (r - l)/2;

        if (arr[mid] == x)
        return mid;

        if (arr[mid] > x)
        return binarySearch(arr, l, mid-1, x);

        return binarySearch(arr, mid+1, r, x);
}

return -1;
}
```

Recurrence Relation

$T(n)=T(n/2) +1$

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

Divide and Conquer: Merge Sort

- The merge sort is sorting techniques which uses the merging technique of two arrays.
- The array is divided into equal half until single the single element and then it is combined with the merging technique.
- It uses divide and conquer paradigm
  - Divide: Divide the array into two equal subarray, each having half of the size of the initial array.
  - Conquer: Sort each of the two subarray until single element, i.e. size of the sub-array becomes 1.
  - Combine: Merge the two sorted subarray and combine into a single sorted list.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

Divide and Conquer: Merge Sort

- Algorithm:

```
MergeSort(A, lb, ub)
{
    if(lb<ub)
        Mid=(lb+ub)/2
        MergeSort(A,lb,Mid)
        MergeSort(A,Mid+1,ub)
        Merge(A, lb, Mid, ub)
}
```

Recurrence Relation:

$T(n)=2T(n/2)+\Theta(n)$

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

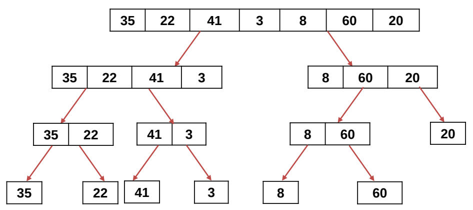
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Saumya Bansal

Divide and Conquer: Merge Sort

```
void merge(int arr[], int l, int m, int r)
{
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
  int L[n1], R[n2];

   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];

    i = 0; j = 0; k = l;
    while (i < n1 && j < n2)
  {
       if (L[i] <= R[j]) {
          arr[k] = L[i];
          i++;
       }
       else {
          arr[k] = R[j];
          j++;
       }
       k++;
    }

    while (i < n1)
    {
          arr[k] = L[i];
          i++;
          k++;
     }
     while (j < n2)
    {
          arr[k] = R[j];
          j++;
          k++;
     }
    }
```

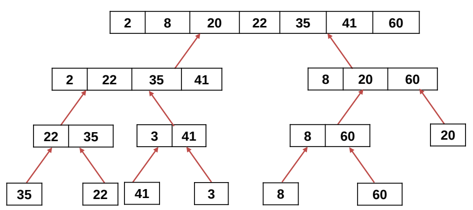
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

Divide and Conquer: Merge Sort



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

Divide and Conquer: Merge Sort



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Saumya Bansal

Divide and Conquer: Merge Sort

- Time Complexity:
  - Best Case: O(nlogn)
  - Average Case: O(nlogn)
  - Worst Case: O(nlogn)
- Space Complexity:
  - space=O(n)
    - ✓ Recursion stack: O(logn)
    - ✓ Merge: O(n)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal U1.13

Divide and Conquer: Quick Sort

- The quick sort algorithm divides the array into two subarray based on the pivot element.
- The elements of left subarray is less than of pivot element and the element of right subarray is greater than the pivot element.
- Quick sort is based on the Divide and Conquer Paradigm
  - Divide: The array is divided into two subarray
  - Conquer: Sort each of the subarray recursively
  - Combine: No combination stage. Once the conquer step done, the sorting is done

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal U1.14

Divide and Conquer: Quick Sort

9	7	10	5	16	6	15
Pivot	right					left

right<pivot; increment right pointer

9	7	10	5	16	6	15
Pivot		right				left

right>pivot; stop increment, start comparison with left

9	7	10	5	16	6	15
Pivot		right			left	

left>pivot; decrement left pointer

9	7	10	5	16	6	15
Pivot		right			left	

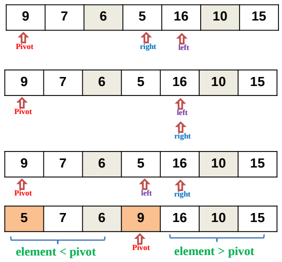
left<pivot; stop; now right<left; Swap and increment left and decrement right pointer after swapping

9	7	6	5	16	10	15
Pivot			right	left		

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal U1.15

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Saumya Bansal

Divide and Conquer: Quick Sort



right<pivot; increment right pointer

right>pivot; stop increment, start comparison with left

left>pivot; decrement left pointer

now left<right; Swap left element with pivot element

Divide and Conquer: Quick Sort

- **Algorithm:**

```
QuickSort(A, lb, ub)
{
    if(lb<ub)
      {
      pivot_index=partition(A,lb,ub);
      QuickSort(A,lb,pivot_index-1);
      QuickSort(A,pivot_index + 1, ub);
      }
}
```

Recurrence Relation:
Best Case: $T(n)=2T(n/2)+\Theta(n)$

Recurrence Relation:
Worst Case: $T(n)=T(n-1)+\Theta(n)$

Divide and Conquer: Quick Sort

- **Algorithm:**

```
int partition(A, lb, ub)
  {
    pivot = arr[lb];
    i = lb + 1
    j = ub - 1
    while(i<j)
    {
      while (arr[i] < pivot)
       {
         i++;
       }
```

```
while (arr[j] > pivot)
      {
        j--;
      }
        if(i!=j)
        swap arr[i++] and arr[j++]
   }
   swap arr[j] and arr[lb])
   return j
}
```


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Saumya Bansal

Divide and Conquer: Quick Sort

- Time Complexity:
  - Best Case: O(nlogn)
  - Average Case: O(nlogn)
  - Worst Case: $O(n^2)$ Why? (refer worst case recurrence relation)
- Space Complexity:
  - space=O(logn)
    - ✓ Recursion stack: O(logn)

Divide and Conquer: Strassen's Algorithm

- Basic Matrix Multiplication
  - Suppose we have two 2X2 matrices, A and B and C=A*B then
  - A=$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and B=$\begin{bmatrix} e & f \\ g & h \end{bmatrix}$ then C= A*B=$\begin{bmatrix} k & l \\ m & n \end{bmatrix}$

k= ae + bg
l= af + bh
m= ce + dg
n= cf + dh

Total 8 Multiplications and 4 additions

Divide and Conquer: Strassen's Algorithm

- Basic Matrix Multiplication

```
MatrixMultiplication (A, B, C) // A=R1XC1 B=R2XC2
for i = 1 to R1 do
        for j = 1 to C2 do
        C[i,j] = 0
                for k = 1 to C1 do
                C[i,j] = C[i,j] + A[i,k] × B[k,j]
```

Time Complexity=$O(n^3)$
//assume all integer additions and multiplications takes O(1)//

Divide and Conquer: Strassen's Algorithm

- Strassen showed that 2X2 matrix multiplication can be accomplished in 7 multiplication and 18 subtractions/additions
  - Suppose we have two 2X2 matrices, A and B and C=A*B then
  - A=$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and B=$\begin{bmatrix} e & f \\ g & h \end{bmatrix}$ then C= A*B=$\begin{bmatrix} k & l \\ m & n \end{bmatrix}$

p1=a(f-h)	p5=(a+d)(e+h)	k=p5+p4-p2+p6
p2=(a+b)h	p6=(b-d)(g+h)	l=p1+p2
p3=(c+d)e	p7=(a-c)(e+f)	m=p3+p4
p4=d(g-e)		n=p1+p5-p3-p7

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

Divide and Conquer: Strassen's Algorithm

- A=$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and B=$\begin{bmatrix} e & f \\ g & h \end{bmatrix}$ then C= A*B=$\begin{bmatrix} k & l \\ m & n \end{bmatrix}$

k=p5+p4-p2+p6

=(a+d)(e+h)+d(g-e)-(a+b)h+(b-d)(g+h)

=ae+$\cancel{ah}$+$\cancel{de}$+$\cancel{dh}$+$\cancel{dg}$-$\cancel{de}$-$\cancel{ah}$-$\cancel{bh}$+bg+$\cancel{bh}$-$\cancel{dg}$-$\cancel{dh}$

=ae+bg

In Normal multiplication:

k= ae + bg
l= af + bh
m= ce + dg
n= cf + dh

In similar manner, we can check the value of l, m and n

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

Divide and Conquer: Strassen's Algorithm

```
void matmul(int A[], int B[], int R[], int n)
{
if (n == 1) {
R +=A * B;
}
else
{
matmul(A, B, R, n/4);
matmul(A, B+(n/4), R+(n/4), n/4);
matmul(A+2*(n/4), B, R+2*(n/4), n/4);
matmul(A+2*(n/4), B+(n/4), R+3*(n/4), n/4);
matmul(A+(n/4), B+2*(n/4), R, n/4);
matmul(A+(n/4), B+3*(n/4), R+(n/4), n/4);
matmul(A+3*(n/4), B+2*(n/4), R+2*(n/4), n/4);
matmul(A+3*(n/4), B+3*(n/4), R+3*(n/4), n/4);
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Saumya Bansal

Divide and Conquer: Strassen's Algorithm

- Recurrence relation: $T(n)=7T(n/2)+O(n^2)$
- Time Complexity: $O(n^{2.81})$
- Generally, Strassen's Method is not preferred for practical applications for following reasons
  - The constants used in Strassen's method are high and for a typical application Naive method works better.
  - For Sparse matrices, there are better methods especially designed for them.
  - The submatrices in recursion take extra space.
  - Strassen's Matrix multiplication can be performed only on **square matrices** where **n** is a **power of 2**. Order of both of the matrices should be **n × n**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal U1.25

Linear Time Sorting Algorithms

- The minimum time sorting algorithm we have learnt so far is merge sort whose time complexity is O(nlogn)
- There are some algorithm that runs faster and takes linear time such as
  - Counting Sort,
  - Radix Sort, and
  - Bucket Sort.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal U1.26

Linear Time Sorting : Counting Sort

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63 by Dr. Saumya Bansal U1.27

Thank You

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Saumya Bansal