Institute of Engineering & Management
Department of Computer Science & Engineering

CS591 Design & Analysis of Algorithm Lab

Assignment List

Week 1:

1. Find second largest and second smallest number simultaneously in an array using Divide & Conquer Principle.
2. Given a sorted array and a number X, search two elements of array such that their sum is X. Expected time complexity is O(n).
3. Apply Binary Search on 2D NxM array (A) having numbers stored in non-deceasing order under row-major scanning. `Hint: k-th element = A[k/M][k % M] for A[1…N][1…M]`
4. Given a sorted array and a number x, write a function that counts the occurrences of x in the array. Expected time complexity is O(logn).

```
Hint:
1) Use Binary search to get index of the first occurrence of x in arr[].
   Let the index of the first occurrence be i.
      if( ( mid == 0 || x > arr[mid-1]) && arr[mid] == x)
         return mid;
      else if(x > arr[mid])
         return first(arr, (mid + 1), high, x, n);
      else
         return first(arr, low, (mid -1), x, n);

2) Use Binary search to get index of the last occurrence of x in arr[].
   Let the index of the last occurrence be j.
   if( ( mid == n-1 || x < arr[mid+1]) && arr[mid] == x )
       return mid;
   else if(x < arr[mid])
     return last(arr, low, (mid -1), x, n);
   else
     return last(arr, (mid + 1), high, x, n);
3) Return (j – i + 1);
```

5. Median of two sorted arrays: There are 2 sorted arrays A and B; each of size n. Write an algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length 2n). The complexity should be O(log(n))

```
Hint:
1) Calculate the medians m1 and m2 of the input arrays ar1[]and ar2[]
respectively.
2) If m1 and m2 both are equal then we are done and return m1 (or m2)
3) If m1 is greater than m2, then median is present in one of the below
two subarrays.
    a)  From first element of ar1 to m1 (ar1[0...|_n/2_|])
    b)  From m2 to last element of ar2  (ar2[|_n/2_|...n-1])
```

4) If m2 is greater than m1, then median is present in one of the below two subarrays.
   a) From m1 to last element of ar1 (ar1[|_n/2_|...n-1])
   b) From first element of ar2 to m2 (ar2[0...|_n/2_|])
5) Repeat the above process until size of both the subarrays becomes 2.
6) If size of the two arrays is 2 then use below formula to get the median. Median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2

6. A sorted array is rotated clockwise arbitrarily. Find the minimum element in it.

   Hint:

   - The minimum element is the only element whose previous is greater than it. If there is no previous element, then there is no rotation (first element is minimum). We check this condition for middle element by comparing it with (mid-1)-th and (mid+1)-th elements.
   - If minimum element is not at middle (neither mid nor mid + 1), then minimum element lies in either left half or right half.
     1. If middle element is smaller than last element, then the minimum element lies in left half
     2. Else minimum element lies in right half.

7. You are given an array that represents elements of arithmetic progression in order. One element is missing in the progression. Find the missing number.

   Hint:
   We can solve this problem in O(Logn) time using Binary Search. The idea is to go to the middle element. Check if the difference between middle and next to middle is equal to diff or not, if not then the missing element lies between mid and mid+1. If the middle element is equal to n/2th term in Arithmetic Series (Let n be the number of elements in input array), then missing element lies in right half. Else element lies in left half.

8. A Bitonic Sequence is a sequence of numbers which is first strictly increasing then after a point strictly decreasing. A Bitonic Point is a point in bitonic sequence before which elements are strictly increasing and after which elements are strictly decreasing. Find bitonic point in a bitonic sequence.

   Hint:
   An efficient solution for this problem is to use modified binary search.

   If arr[mid-1] < arr[mid] > arr[mid+1] then we are done with bitonic point.
   If arr[mid] < arr[mid+1] then search in right sub-array, else search in left sub-array.

Week 2:

1. Given an array of digits, sort them with time complexity O(n).

2. Apply Merge Sort to count inversion pairs in an array. Two elements a[i] and a[j] form an inversion pair if a[i] > a[j] and i < j. Example: The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).
   Hint:

```
int inv_count = 0; i = left; /* i is index for left subarray*/

j = mid;  /* j is index for right subarray*/
k = left; /* k is index for resultant merged subarray*/
while ((i <= mid - 1) && (j <= right))
{
      if (arr[i] <= arr[j])
      {
            temp[k++] = arr[i++];
      }
      else
      {
            temp[k++] = arr[j++];

            inv_count = inv_count + (mid - i);
      }
}
```

3. Given two sorted arrays of size m and n respectively, find k'th element of the combined array.
   Hint: O(n) using Merging.
   Alternative solution with time complexity O(log n + log m)

```
    if (start1 == end1)
        return arr2[k];
    if (start2 == end2)
        return arr1[k];
    int mid1 = (end1 - start1) / 2;
    int mid2 = (end2 - start2) / 2;
    if (mid1 + mid2 < k)
    {
        if (arr1[mid1] > arr2[mid2])
            return kth(start1, start2 + mid2 + 1, end1, end2,
                k - mid2 - 1);
        else
            return kth(start1 + mid1 + 1, start2, end1, end2,
                k - mid1 - 1);
    }
    else
    {
        if (arr1[mid1] > arr2[mid2])
            return kth(start1, start2, start1 + mid1, end2, k);
        else
            return kth(start1, start2, end1, arr2 + mid2, k);
    }
```

4. Find neighbors of the median element in an array using partitioning strategy of Quick-Sorting method.

```
Hint:
General principle of finding k-th element

j = position of pivot element after partitioning
if (k < j) return doPartition( A[1...j-1])
if (k = j) return j
if (k > j) return doPartition( A[j+1...n])
```

5. Apply Strassen's Matrix Multiplication strategy for odd dimensional square matrix.

```
Hint: Do padding of zeros to make odd dimensional square matrix 2^kx2^k.
```

6. Apply min-heap building strategy to find k-th smallest element in an array.

```
Hint.
1) Build a Min Heap of the first k elements (arr[0] to arr[k-1]) of the
given array. O(k)

2) For each element, after the kth element (arr[k] to arr[n-1]), compare
it with root of min heap.
If the element is greater than the root then make it root and call heapify
for min heap.
else ignore it. The step 2 is O((n-k)*logk)
3) Finally, min heap has k largest elements and root of the min heap is
the kth largest element.

Time Complexity: O(k + (n-k)Logk) without sorted output. If sorted output
is needed then O(k + (n-k)Logk + kLogk)
```

7. Apply quick sort on a 2D NxM matrix of numbers so that the numbers are sorted in row-major fashion in the 2D matrix (without auxiliary array).
8. Arrange a list of words (each of equal length) using dictionary sorting strategy.

```
Hint: Radix Sort
```

Week 3:

1. Given a value ***V*** and infinite supply of coins of ***m***-denominations ***{C1=1*** < ***C2<C3<…..<Cm}***, we want to make change for Rs. ***V***. Apply DP strategy to find out minimum number of coins to make the change?

```
Hint:
Input: coins[] = {25, 10, 5}, V = 30
Output: Minimum 2 coins required
We can use one coin of 25 cents and one of 5 cents

Input: coins[] = {9, 6, 5, 1}, V = 11
Output: Minimum 2 coins required
We can use one coin of 6 cents and 1 coin of 5 cents


If V == 0, then 0 coins required.
If V > 0
   minCoin(coins[0..m-1], V) = min {1 + minCoins(V-coin[i])}
                               where i varies from 0 to m-1
                               and coin[i] <= V
```

2. Given a set of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

```
Hint:
Examples: set[] = {3, 34, 4, 12, 5, 2}, sum = 9
Output:  True  // There is a subset (4, 5) with sum 9.
isSubsetSum(set, n, sum) = isSubsetSum(set, n-1, sum) ||
                           isSubsetSum(set, n-1, sum-set[n-1])
Base Cases:
isSubsetSum(set, n, sum) = false, if sum > 0 and n == 0
isSubsetSum(set, n, sum) = true, if sum == 0
```

3. Given a cost 2D-matrix and a position (m, n), write a function that returns cost of minimum cost-path to reach (m, n) from (0, 0).

```
Hint:
Each cell of the matrix represents a cost to traverse through that cell.
Total cost of a path to reach (m, n) is sum of all the costs on that path
(including both source and destination). You can only traverse down, right
and diagonally lower cells from a given cell, i.e., from a given cell (i,
j), cells (i+1, j), (i, j+1) and (i+1, j+1) can be traversed. You may
assume that all costs are positive integers.

The path to reach (m, n) must be through one of the 3 cells: (m-1, n-1)
or (m-1, n) or (m, n-1). So minimum cost to reach (m, n) can be written
as “minimum of the 3 cells plus cost[m][n]”.

minCost(m, n) = min (minCost(m-1, n-1), minCost(m-1, n), minCost(m, n-
1)) + cost[m][n]
```

4. Given an array p[] which represents the chain of matrices such that the i-th matrix A_i is of dimension p[i-1] x p[i]. We need to write a function that should return the optimal parenthesizing expression resulting minimum multiplication cost to multiply the chain.

```
Hint:
  Input: p[] = {40, 20, 30, 10, 30}
  Output: 26000
  There are 4 matrices of dimensions 40x20, 20x30, 30x10 and 10x30.
  Let the input 4 matrices be A, B, C and D.  The minimum number of
  multiplications are obtained by putting parenthesis in following way
  (A(BC))D --> 20*30*10 + 40*20*10 + 40*10*30
```

5. There are N stations on route of a train. The train goes from station 0 to N-1. The cost of ticket for pair of stations (i, j) is given where j is greater than i. Find the minimum cost to reach the destination.

```
Hint:
Input:
cost[N][N] = { {0, 15, 80, 90},
               {INF, 0, 40, 50},
               {INF, INF, 0, 70},
               {INF, INF, INF, 0}
             };
There are 4 stations and cost[i][j] indicates cost to reach j
from i. The entries where j < i are meaningless.

Output:
The minimum cost is 65
The minimum cost can be obtained by first going to station 1
from 0. Then from station 1 to station 3.
minCost(0, N-1) = MIN { cost[0][n-1],
                  cost[0][1] + minCost(1, N-1),
                  minCost(0, 2) + minCost(2, N-1),
                  ........,
                  minCost(0, N-2) + cost[N-2][n-1] }
```

6. Implement all pairs of Shortest path algorithm for a graph using Floyd-Warshall's strategy.
7. Given an array of **n** numbers, design an algorithm for finding a contiguous subsequence **A[i], A[i+1],…,A[i+k]** having largest sum.
8. Given weights and values of ***n*** items, put these items in a knapsack of capacity ***W*** to get the maximum total value in the knapsack. You cannot break an item, either pick the item, or don't pick it.

```
Hint: To consider all subsets of items, there can be two cases for every
item: (1) the item is included in the optimal subset, (2) not included in
the optimal set. Therefore, the maximum value that can be obtained from
n items is max of following two values.
1) Maximum value obtained by n-1 items and W weight (excluding nth item).
2) Value of nth item plus maximum value obtained by n-1 items and W minus
weight of the nth item (including nth item). If weight of nth item is
greater than W, then the nth item cannot be included and case 1 is the
only possibility.
```

Week 4:

1. Implement BFS and DFS for a given undirected graph and starting vertex.
2. Detect whether an undirected graph contains cycle.
3. In an undirected graph, a cut vertex is a vertex whose removal from the graph results in two disconnected components. Write an algorithm to find set of cut vertices in an undirected graph.
4. Given a directed acyclic graph, write an algorithm for finding the depth.
5. Implement Dijkstra's algorithm for finding shortest path.
6. Implement Bellman-Ford's algorithm for finding shortest path.
7. Implement DP strategy to solve Travelling Salesman Problem(TSP).
8. Given a sorted dictionary of an alien language, find order of characters.
   Hint.

```
Input:  words[] = {"baa", "abcd", "abca", "cab", "cad"}
Output: Order of characters is 'b', 'd', 'a', 'c'
Note that words are sorted and in the given language "baa" comes before
"abcd", therefore 'b' is before 'a' in output. Similarly, we can find
other orders.
```

WEEK5

1. Find the smallest number with given digit sum **s** and number of digits **d**?

   ```
   Input  : s = 9, d = 2
   Output : 18
   There are many other possible numbers
   like 45, 54, 90, etc with sum of digits
   as 9 and number of digits as 2. The
   smallest of them is 18.
   ```
   There is a **Greedy approach** to solve the problem. The idea is to one by one fill all digits from rightmost to leftmost (or from least significant digit to most significant). We initially deduct 1 from sum s so that we have smallest digit at the end. After deducting 1, we apply greedy approach. We compare remaining sum with 9, if remaining sum is more than 9, we put 9 at the current position, else we put the remaining sum. Since we fill digits from right to left, we put the highest digits on the right side.

2. Every positive fraction can be represented as sum of unique unit fractions. A fraction is unit fraction if numerator is 1 and denominator is a positive integer (for example 1/3 is a unit fraction). Design and implement an algorithm to find all the unique unit fractions so that their sum equals a given positive fraction.

   ```
   2/3 is 1/2 + 1/6
   6/14 is 1/3 + 1/11 + 1/231
   ```
   For a given number of the form ‘nr/dr’ where dr > nr, first find the greatest possible unit fraction, then recur for the remaining part. For example, consider 6/14, we first find ceiling of 14/6, i.e., 3. So the first unit fraction becomes 1/3, then recur for (6/14 – 1/3) i.e., 4/42.

3. We need to connect **n** ropes of different lengths into one rope. The cost to connect two ropes is equal to sum of their lengths. Design and implement an algorithm to find the connection sequence resulting in minimum cost.

   Explanation: If we are given 4 ropes of lengths 4, 3, 2 and 6. We can connect the ropes in following ways.

   1) First connect ropes of lengths 2 and 3. Now we have three ropes of lengths 4, 6 and 5.
   2) Now connect ropes of lengths 4 and 5. Now we have two ropes of lengths 6 and 9.
   3) Finally connect the two ropes and all ropes have connected.

   Total cost for connecting all ropes is 5 + 9 + 15 = 29. This is the optimized cost for connecting ropes. Other ways of connecting ropes would always have same or more cost. For example, if we connect 4 and 6 first (we get three strings of 3, 2 and 10), then connect 10 and 3 (we get two strings of 13 and 2). Finally, we connect 13 and 2. Total cost in this way is 10 + 13 + 15 = 38.

```
Hint: Let there be n ropes of lengths stored in an array len[0..n-1]
1) Create a min heap and insert all lengths into the min heap.
2) Do following while number of elements in min heap is not one.
      2.1) Extract the minimum and second minimum from min heap
      2.2) Add the above two extracted values and insert the added
value to the min-heap.
3) Return the value of only left item in min heap.
```

4. Suppose you are given a set **S = {a1, a2, ..., an}** of tasks, where task $\mathbf{a_i}$ requires pi units of processing time to complete, once it has started. You have one computer on which to run these tasks, and the computer can run only one task at a time. Let ci be the completion time of task $\mathbf{a_i}$, that is, the time at which task $\mathbf{a_i}$ completes processing. Your goal is to design an algorithm to find a schedule of the tasks so that the average completion time becomes minimum.

   Explanation: Suppose there are two tasks, $a1$ and $a2$, with $p1 = 3$ and $p2 = 5$, and consider the schedule in which $a2$ runs first, followed by $a1$. Then $c2 = 5$, $c1 = 8$, and the average completion time is $(5 + 8)/2 = 6.5$.

5. Implement a greedy algorithm to solve fractional **knapsack** problem.
6. Implement **Kruskal's** and **Prim's** algorithm for finding MST.
7. Implement greedy algorithm to solve the problem of **Job Sequencing with deadlines**.
8. Implement greedy algorithm of **Huffman's** binary encoding of alphabets for a given list of words.

WEEK6

1. You are given a 3×3 board with 8 tiles (every tile has one number from 1 to 8 and one space is empty). The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

Initial Configuration

1	2	3
5	6	
7	8	4

Final Configuration

1	2	3
5	8	6
	7	4

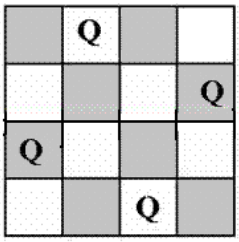
2. Job Assignment Problem: Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment. It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Worker A takes 8 units of time to finish job 4.

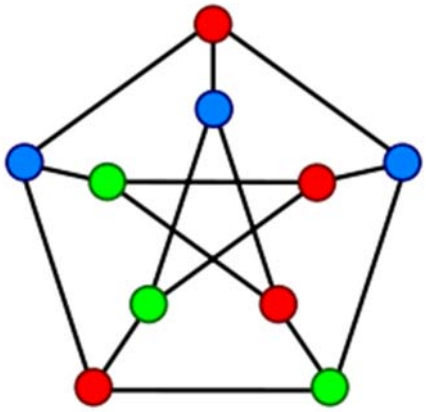
An example job assignment problem. Green values show optimal job assignment that is A-Job4, B-Job1 C-Job3 and D-Job4

3. Write a program to print all permutations of a given string (use Backtracking Principle).
4. The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen-problem.



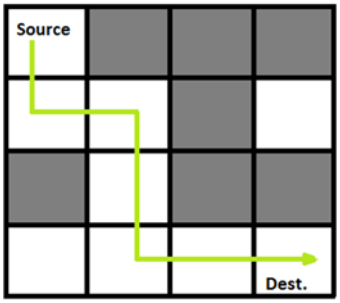
```
Hint:
1) Start in the leftmost column
2) If all queens are placed
    return true
3) Try all rows in the current column.  Do following for every tried
row.
    a) If the queen can be placed safely in this row then mark this [row,
column] as part of the solution and recursively check if placing queen
here leads to a solution.
    b) If placing queen in [row, column] leads to a solution then return
true.
    c) If placing queen doesn't lead to a solution then unmark this [row,
column] (Backtrack) and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked, return false to trigger
backtracking.
```

5. Given an undirected graph and a number **m**, determine if the graph can be colored with at most **m** colors such that no two adjacent vertices of the graph are colored with same color. Here coloring of a graph means assignment of colors to all vertices.



6. Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in graph) from the last vertex to the first vertex of the Hamiltonian Path. Determine whether a given graph contains Hamiltonian Cycle or not. If it contains, then print the path.

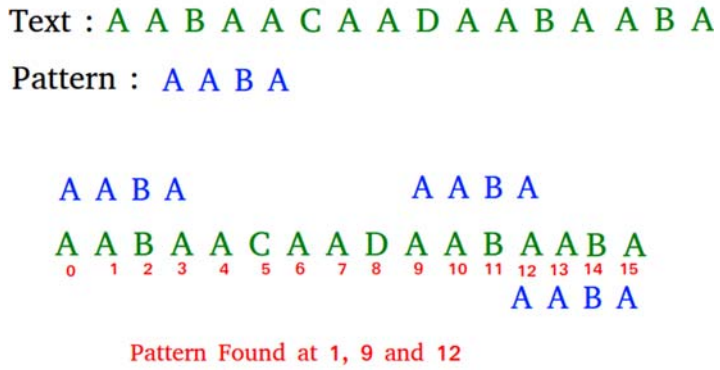
7. A Maze is given as **NxN** binary matrix of blocks where source block is the upper left most block i.e., **maze[0][0]** and destination block is lower rightmost block i.e., **maze[N-1][N-1]**. A rat starts from source and has to reach destination. The rat can move only in two directions: forward and down. Write an algorithm to find its path from source to destination.



Hint:

```
If destination is reached
    print the solution matrix
Else
   a) Mark current cell in solution matrix as 1.
   b) Move forward in horizontal direction and recursively check if this
move leads to a solution.
   c) If the move chosen in the above step doesn't lead to a solution
       then move down and check if this move leads to a solution.
   d) If none of the above solutions work then unmark this cell as 0
       (BACKTRACK) and return false.
```

8. **KMP String Matching:** Given a text **txt[0..n-1]** and a pattern **pat[0..m-1]**, write a function **search(char pat[], char txt[])** that prints all occurrences of **pat[]** in **txt[]**. You may assume that **n** > **m**.



Institute of Engineering & Management
Department of Computer Science & Engineering

CS591 Design & Analysis of Algorithms

Semester : 5 (3rd Year) Year : 2017

Project List

1.	**Text Analytics**	
	1.1.	Plagiarism Detection: Text Article
	1.2.	Opinion Mining/Sentiment Analysis: Movie Review
	1.3.	Spam Detection: e-mail
	1.4.	Recommendation System: Course Recommendation Engine
2.	**Constrained Planning**	
	2.1.	Class Routine Generation
	2.2.	Examination Coordination Event Planning
	2.3.	Trip/Route Planning
3.	**Vehicle-Traffic Flow Management**	
4.	**Sudoku Puzzle Solver**	