ASSIGNMENT 1

Write a program to compute square of 20-digit large integer numbers using
divide and conquer
strategy.

```
import java.util.Scanner;
public class fine {
    public static String addition(String a, String b) {
        if (a.length() > b.length()) {
            String temp = a;
            a = b;
            b = temp;
        }
        String str = "";
        int n1 = a.length();
        int n2 = b.length();

        a = new StringBuilder(a).reverse().toString();
        b = new StringBuilder(b).reverse().toString();

        int carry = 0;
        for (int i = 0; i < n1; i++) {
            int sum = ((a.charAt(i) - '0') + (b.charAt(i) - '0') + carry);
            str += (char)(sum % 10 + '0');
            carry = sum / 10;
        }
        for (int i = n1; i < n2; i++) {
            int sum = ((b.charAt(i) - '0') + carry);
            str += (char)(sum % 10 + '0');
            carry = sum / 10;
        }
        if (carry != 0)
            str += (char)(carry + '0');

        str = new StringBuilder(str).reverse().toString();
        return str;
    }

    static String diff(String a, String b) {
        String str = "";
        int n1 = a.length(), n2 = b.length();

        a = new StringBuilder(a).reverse().toString();
        b = new StringBuilder(b).reverse().toString();

        int carry = 0;
        for (int i = 0; i < n2; i++) {
            int sub = ((a.charAt(i) - '0') - (b.charAt(i) - '0') - carry);
            if (sub < 0) {
                sub += 10;
                carry = 1;
            } else
                carry = 0;
            str += sub;
        }
        for (int i = n2; i < n1; i++) {
            int sub = ((a.charAt(i) - '0') - carry);
            if (sub < 0) {
                sub += 10;
                carry = 1;
            } else
                carry = 0;
            str += sub;
        }
        str = new StringBuilder(str).reverse().toString();
        return str;
    }

    public static String zeros(String str) {
        String pattern = "^0+(?!$)";
        str = str.replaceAll(pattern, "");
        return str;
    }

    public static String multiply(String A, String B) {
        if (A.length() > B.length()) {
            String temp = A;
            A = B;
            B = temp;
    }
    int n1 = A.length(), n2 = B.length();
    while (n2 > n1) {
        A = "0" + A;
        n1++;
    }
    if (n1 == 1) {
        int ans = Integer.parseInt(A) * Integer.parseInt(B);
        return Integer.toString(ans);
    }
    if (n1 % 2 == 1) {
        n1++;
        A = "0" + A;
        B = "0" + B;
    }

    String al = "", ar = "", bl = "", br = "";
    for (int i = 0; i < n1 / 2; ++i) {
        al += A.charAt(i);
        bl += B.charAt(i);
        ar += A.charAt(n1 / 2 + i);
        br += B.charAt(n1 / 2 + i);
    }
    String p = multiply(al, bl);
    String q = multiply(ar, br);
    String r = diff(multiply(addition(al, ar), addition(bl, br)), addition(p, q));

    for (int i = 0; i < n1; ++i)
        p = p + "0";
    for (int i = 0; i < n1 / 2; ++i)
        r = r + "0";
    String ans = addition(p, addition(q, r));
    ans = zeros(ans);
    return ans;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int choice;
```

```
do {
    System.out.println("Enter your choice:");
    System.out.println("1. Multiplication");
    System.out.println("2. Square");
    System.out.println("3. Addition");
    System.out.println("4. Subtraction");
    System.out.println("5. Exit");

    choice = sc.nextInt();
    sc.nextLine();

    switch (choice) {
        case 1:
            System.out.println("Enter numbers A and B:");
            String A = sc.nextLine();
            String B = sc.nextLine();
            System.out.println("Product: " + multiply(A, B));
            break;

        case 2:
            System.out.println("Enter number A:");
            A = sc.nextLine();
            System.out.println("Square: " + multiply(A, A));
            break;

        case 3:
            System.out.println("Enter numbers A and B:");
            A = sc.nextLine();
            B = sc.nextLine();
            System.out.println("Sum: " + addition(A, B));
            break;

        case 4:
            System.out.println("Enter numbers A and B:");
            A = sc.nextLine();
            B = sc.nextLine();
            System.out.println("Difference: " + diff(A, B));
            break;

        case 5:
                System.out.println("Exiting the program");
                break;

            default:
                System.out.println("Invalid choice.");
        }
    } while (choice != 5);
    sc.close();
  }
}
```

Output

```
PS D:\Java> javac fine.java
PS D:\Java> java fine.java
Enter your choice:
1. Multiplication
2. Square
3. Addition
4. Subtraction
5. Exit
1
Enter numbers A and B:
12345678
67894521
Product: 838203894230238
Enter your choice:
1. Multiplication
2. Square
3. Addition
4. Subtraction
5. Exit
2
Enter number A:
123456789
Square: 15241578750190521
Enter your choice:
1. Multiplication
2. Square
3. Addition
```

```
4. Subtraction
5. Exit
3
Enter numbers A and B:
567896
122334
Sum: 690230
Enter your choice:
1. Multiplication
2. Square
3. Addition
4. Subtraction
5. Exit
4
Enter numbers A and B:
788762544
656758
4. Subtraction
5. Exit
4
Enter numbers A and B:
788762544
656758
Difference: 788105786
Enter your choice:
1. Multiplication
2. Square
3. Addition
4. Subtraction
5. Exit
4. Subtraction
5. Exit
4
Enter numbers A and B:
788762544
656758
Difference: 788105786
Enter your choice:
1. Multiplication
2. Square
```

```
3. Addition
4. Subtraction
5. Exit
4
Enter numbers A and B:
788762544
656758
Difference: 788105786
5. Exit
4
Enter numbers A and B:
788762544
656758
4
Enter numbers A and B:
788762544
656758
788762544
656758
656758
Difference: 788105786
Enter your choice:
1. Multiplication
2. Square
3. Addition
4. Subtraction
5. Exit
5
Exiting the program
```

ASSIGNMENT 1   PYTHON

```
def algo(num1,num2):
    if num1 < 10 or num2 < 10:
        return num1*num2


    n=max(len(str(num1)),len(str(num2)))
    half=n//2
    a=num1//(10**half)
    b=num1%(10**half)
    c=num2//(10**half)
    d=num2%(10**half)

    ac=algo(a,c)
    bd=algo(b,d)
    ad_bc=algo(a+b,c+d)-ac-bd

    result = ac * (10 ** (2 * half)) + ad_bc * (10 ** half) + bd
    return result

while(1):
    c1=int(input("Enter Your Choice: (1:For Multipication:) (2:For Square:)"))
    if(c1==1):
        num1=int(input("Enter No.1: "))
        num2=int(input("Enter No.2: "))
        result=algo(num1,num2)
        print("Ans is: ",result)

    if(c1==2):
        num1=int(input("Enter No.: "))
        result=algo(num1,num1)
        print("Ans is: ",result)
```

Output:

```
 Output:
 PS D:\OneDrive\Desktop\python> python ass1.py
 Enter a number: 1234
 Square is 1522756
```

ASSIGNMENT 2

Problem Statement: Consider the scheduling problem. n tasks to be scheduled on single processor. Let d1, ...,dn be deadline and p1,….pn be the profit of each task to execute on single processor is known. The tasks can be executed in any order but one task at a time and each task take 1 unit of time

to execute. Design a greedy algorithm for this problem and find a schedule or sequence of jobs that gives maximum profit.
Program:
```
import java.util.*;
class Job {
    int jobID;
    int deadline;
    int profit;
    Job(int jobID, int deadline, int profit) {
        this.jobID = jobID;
        this.deadline = deadline;
        this.profit = profit;
    }
    int getprofit()
    {
        return profit;
    }
}
public class Ass_2{
   void final_ans(Job[] jobs, int n)
   {
       int maxDeadline = 0;
        for (int i = 0; i < n; i++) {
            if (jobs[i].deadline > maxDeadline) {
                maxDeadline = jobs[i].deadline;
            }
        }
        int m=maxDeadline;
        int[] slot=new int[m];
        Arrays.fill(slot, 0);
    Arrays.sort(jobs,Comparator.comparingInt(Job::getprofit).reversed());
    int cnt=0,total=0;
    for(int i=0;i<n && cnt<m;i++)
    {
        for(int j=jobs[i].deadline-1;j>=0;j--)
        {
            if(slot[j]==0)
            {
                slot[j]=jobs[i].jobID;
                cnt++;
                total+=jobs[i].profit;
                break;
            }
        }
 }
    System.out.println("Maximum profit is :"+total+" With following job slots");
    for(int i=0;i<m;i++)
    {
        System.out.println(slot[i]+" ");
    }
   }
public static void main(String[] args)
{
   Scanner sc=new Scanner(System.in);
   System.out.println("Enter the size of Array");
    int n = sc.nextInt();

    Job[] jobs = new Job[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter job ID, deadline, and profit for job " + (i + 1) +
":");
            int jobID = sc.nextInt();
            int deadline = sc.nextInt();
            int profit = sc.nextInt();
            jobs[i] = new Job(jobID, deadline, profit);
        }
    Ass_2 obj=new Ass_2();
    obj.final_ans(jobs,n);

}
}
```
Output:
```
Enter the size of Array
5
Enter job ID, deadline, and profit for job 1:
1
2
30
Enter job ID, deadline, and profit for job 2:
2
```

```
1
20
Enter job ID, deadline, and profit for job 3:
3
1
70
Enter job ID, deadline, and profit for job 4:
4
3
10
Enter job ID, deadline, and profit for job 5:
5
2
90
Maximum profit is :170 With following job slots
3
5
4

=== Code Execution Successful ===
```

ASSIGNMENT 3

Title : You have a business with several offices; you want to lease phone lines to connect them
up with each other; and the phone company charges different amounts of money to connect
different pairs of cities. You want a set of lines that connects all your offices with a minimum
total cost. Solve the problem by Floyd-Warshall algorithm.
Program :

```java
import java.util.Scanner;

public class Ass_3 {
    // Function to implement Floyd Warshall Algorithm
    public static void warshall(int mat[][]) {
        int n = mat.length;

        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    // Update the shortest distance between i and j through k
                    if (mat[i][k] != Integer.MAX_VALUE && mat[k][j] !=
Integer.MAX_VALUE) {
                            mat[i][j] = Math.min(mat[i][j], mat[i][k] + mat[k][j]);
                    }
                }
            }
        }
        // Print the resulting matrix
        printMatrix(mat);
    }

    // Function to print a matrix
    public static void printMatrix(int mat[][]) {
        int n = mat.length;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (mat[i][j] == Integer.MAX_VALUE) {
                    System.out.print("INF ");
                } else {
                    System.out.print(mat[i][j] + " ");
                }
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of vertices: ");
        int n = sc.nextInt();
        int mat[][] = new int[n][n];

        System.out.println("Enter -1 for infinite distances:");

        // Initialize the matrix
        for (int i = 0; i < n; i++) {
            System.out.print("Enter the distances from vertex " + (i + 1) + ": ");
            for (int j = 0; j < n; j++) {
                int input = sc.nextInt();
                if (input >= 0) {
                    mat[i][j] = input;
                } else {
                    mat[i][j] = Integer.MAX_VALUE; // Represent infinity
                }
            }
        }

        // Create an instance and apply the Floyd Warshall algorithm
        Ass_3 f = new Ass_3();

        System.out.println("The input matrix is:");
        printMatrix(mat);

        System.out.println("The matrix after applying Floyd Warshall Algorithm:");
        warshall(mat);

        sc.close();
    }
}
```

Output:

```
Enter the number of vertices: 4
Enter -1 for infinite distances:
Enter the distances from vertex 1: 0 3 -1 7
Enter the distances from vertex 2: 8 0 2 -1
Enter the distances from vertex 3: 5 -1 0 1
Enter the distances from vertex 4: 2 -1 -1 0
The input matrix is:
0 3 INF 7
8 0 2 INF
5 INF 0 1
2 INF INF 0
The matrix after applying Floyd Warshall Algorithm:
0 3 5 6
5 0 2 3
```

```
3 6 0 1
2 5 7 0
```

ASSIGNMENT 4

Problem Statement: You have been given a network of ‘N’ nodes from 1 to ‘N’ and ‘M’ edges. For each edge, you are given three values (ui, vi, wi) where “ui” and “vi” denote the nodes and “wi” denotes an integer value which represents the time taken by a signal to travel from “ui” to “vi”. Now, you are supposed to find the time which a signal takes to travel from a given node ‘K’ to all nodes. If it is impossible for all nodes to receive the signal then print -1.
Implement the given Network Delay Time using Dijkstra’s algorithm
Program:

```
import java.util.*;
public class assign4 {
    static int mincost(int cost[], boolean vis[])
    {
        int min=Integer.MAX_VALUE;
        int idx=-1;
        for(int i=0; i<cost.length;i++)
        {
            if(!vis[i]&&cost[i]<=min)
            {
                min=cost[i];
                idx=i;
            }
        }
        return idx;
    }
    static void shortestpath(int graph[][], int src, int n)
    {
        int cost[] =new int[n];
        boolean vis[]=new boolean[n];
        for(int i=0;i<n;i++)
        {
            cost[i]=Integer.MAX_VALUE;
            vis[i]=false;
        }
        cost[src]=0;
        for(int i=0;i<n;i++)
        {
            int x=mincost(cost,vis);
            vis[x]=true;
            for(int j=0;j<n;j++)
            {

if(!vis[j]&&graph[x][j]!=0&&cost[x]!=Integer.MAX_VALUE&&cost[x]+graph[x][j]<
cost[j])
                {
                    cost[j]=graph[x][j]+cost[x];
                }
            }
        }
        for(int i=0;i<cost.length;i++)
        {
            System.out.println(i+" "+cost[i]);
        }
    }
    public static void main(String args[])
    {
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter no of vertex");
        int v=sc.nextInt();
        int graph[][]=new int[v][v];

        for(int i=0;i<v;i++)
        {
            for(int j=0;j<v;j++)
            {
                System.out.println("Enter edge["+i+"]"+"["+j+"]");
                graph[i][j]=sc.nextInt();
            }
        }
        System.out.println("Shortest Path is ");
        shortestpath(graph,0,v);
    }
}
```

Output

```
PS D:\Java> javac assign4.java
PS D:\Java> java assign4.java
Enter no of vertex
6
Enter edge[0][0]
0
Enter edge[0][1]
2
Enter edge[0][2]
4
Enter edge[0][3]
0
Enter edge[0][4]
0
Enter edge[0][5]
0
Enter edge[1][0]
2
Enter edge[1][1]
0
Enter edge[1][2]
1
Enter edge[1][3]
7
Enter edge[1][4]
0
Enter edge[1][5]
0
Enter edge[2][0]
4
Enter edge[2][1]
1
Enter edge[2][2]
0
Enter edge[2][3]
0
Enter edge[2][4]
3
Enter edge[2][5]
0
```

```
Enter edge[3][0]
0
Enter edge[3][1]
7
Enter edge[3][2]
0
Enter edge[3][3]
0
Enter edge[3][4]
2
Enter edge[3][5]
1
Enter edge[4][0]
0
Enter edge[4][1]
0
Enter edge[4][2]
3
Enter edge[4][3]
2
Enter edge[4][4]
0
Enter edge[4][5]
5
Enter edge[5][0]
0
Enter edge[5][1]
0
Enter edge[5][2]
0
Enter edge[5][3]
1
Enter edge[5][4]
5
Enter edge[5][5]
0
Shortest Path is
0 0
1 2
2 3
```

3 8
4 6
5 9

ASSIGNMENT 5

Problem Statement : A classic problem that can be solved by backtracking is called the Knight's tour Problem. It is a problem in which we are provided with a NxN chessboard and a knight. For a person who is not familiar with chess, the knight moves two squares horizontally and one square vertically, or two squares vertically and one square horizontally. In this problem, there is an empty chess board, and a knight starting from any location in the board, our task is to check whether the knight can visit all of the squares in the board or not. When It can visit all of the squares, then place the number of jumps needed to reach that location from the starting point.

Code

```
import java.util.*;

class Knight_Tour {
        int N;
        Knight_Tour() {}
        Knight_Tour(int N) {
                this.N = N;
        }
        public void Solution_Knight_Tour(int x, int y) {
                int Chess[][] = new int[N][N];

                for(int i = 0; i < N; i++) {
                        for(int j = 0; j < N; j++) {
                                Chess[i][j] = -1;
                        }
                }

                int MovX[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
       int MovY[] = { 1, 2, 2, 1, -1, -2, -2, -1 };

       Chess[x][y] = 0;
```

```
        if (!Tour(x, y, 1, Chess, MovX, MovY)) {
            System.out.println("Solution does not exist for starting position.");
            return;
        }
        else {
          System.out.println("Solution does exist for starting position.");
          PrintChessBoard(Chess);
        }
          }
          public boolean Tour(int x, int y, int movei, int Chess[][], int MovX[], int
MovY[]) {
                int k, x_new, y_new;
                if (movei == N * N)
                        return true;

                for (k = 0; k < 8; k++) {
                        x_new = x + MovX[k];
                        y_new = y + MovY[k];
                        if (Safe(x_new, y_new, Chess)) {
                                Chess[x_new][y_new] = movei;
                                if (Tour(x_new, y_new, movei + 1, Chess, MovX,
MovY))
                                        return true;
                                else {
                                        Chess[x_new][y_new] = -1;
                                }
                        }
                }
                return false;
          }
          public boolean Safe(int x, int y, int Chess[][])
    {
        return (x >= 0 && x < N && y >= 0 && y < N && Chess[x][y] == -1);
    }
          public void PrintChessBoard(int Chess[][]) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                System.out.print(Chess[x][y] + " ");
            }
            System.out.println();
        }
          }
}
public class Knight_Tour_Problem {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Scanner sc = new Scanner(System.in);
                int n;
                System.out.print("Enter number of one dimension of chess: ");
                n = sc.nextInt();
                int x, y;
                System.out.print("Enter starting X position in Chess board: ");
                x = sc.nextInt();
                System.out.print("Enter starting Y position in Chess board: ");
                y = sc.nextInt();
                Knight_Tour K = new Knight_Tour(n);

                K.Solution_Knight_Tour(x, y);
        }

}
```

Output

```
Enter number of one dimension of chess: 8
Enter starting X position in Chess board: 0
Enter starting Y position in Chess board: 0
Solution does exist for starting position.
0 59 38 33 30 17 8 63
37 34 31 60 9 62 29 16
58 1 36 39 32 27 18 7
35 48 41 26 61 10 15 28
42 57 2 49 40 23 6 19
47 50 45 54 25 20 11 14
56 43 52 3 22 13 24 5
51 46 55 44 53 4 21 12

Enter number of one dimension of chess: 4
Enter starting X position in Chess board: 0
Enter starting Y position in Chess board: 0
```

Solution does not exist for starting position.

ASSIGNMENT 6

Problem Statement

Let there be N students and N clubs. Any student can be assigned to any club, incurring some cost that may vary depending on the student club assignment. It is required to allocate all clubs by assigning exactly one student to each club and exactly one club to each agent in such a way that the total cost of the assignment is minimized. Implement club assignment problem using Branch and bound.

Code:

```
import java.util.*;

class C implements Comparable<C> {
    int sl;
    int[] ac;
    int c;
    int lb;

    public C(int sl, int[] ac, int c, int lb) {
        this.sl = sl;
        this.ac = ac.clone();
        this.c = c;
        this.lb = lb;
    }

    @Override
    public int compareTo(C o) {
        return Integer.compare(this.lb, o.lb);
    }
}

public class A6 {
    public static int aClubs(int[][] cm, int[] res) {
        int n = cm.length;
        PriorityQueue<C> pq = new PriorityQueue<>();
        int[] ia = new int[n];
        Arrays.fill(ia, -1);
        C r = new C(0, ia, 0, lb(cm, ia, 0));
        pq.add(r);

        while (!pq.isEmpty()) {
            C cur = pq.poll();

            if (cur.sl == n) {
                System.arraycopy(cur.ac, 0, res, 0, n);
                return cur.c;
            }

            int st = cur.sl;
            for (int cl = 0; cl < n; cl++) {
                if (!clubAsg(cur.ac, cl)) {
                    int[] newAc = cur.ac.clone();
                    newAc[st] = cl;
                    int newC = cur.c + cm[st][cl];
                    int newLb = lb(cm, newAc, st + 1);

                    C newNode = new C(st + 1, newAc, newC, newLb);
                    pq.add(newNode);
                }
            }
        }

        return -1;
    }

    private static boolean clubAsg(int[] ac, int cl) {
        for (int a : ac) {
            if (a == cl) return true;
        }
        return false;
    }

    private static int lb(int[][] cm, int[] ac, int sl) {
        int n = cm.length;
        int l = 0;

        for (int i = 0; i < sl; i++) {
            l += cm[i][ac[i]];
        }

        for (int i = sl; i < n; i++) {
            int min = Integer.MAX_VALUE;
            for (int j = 0; j < n; j++) {
                if (!clubAsg(ac, j)) {
                    min = Math.min(min, cm[i][j]);
                }
            }
            l += min;
        }

        return l;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of students/clubs: ");
        int n = sc.nextInt();
        int[][] cm = new int[n][n];

        System.out.println("Enter the cost matrix: ");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cm[i][j] = sc.nextInt();
            }
        }

        int[] res = new int[n];
        int minC = aClubs(cm, res);

        System.out.println("The minimum cost of assigning clubs is: " + minC);
        System.out.println("Club assignments (student -> club): ");
        for (int i = 0; i < n; i++) {
            System.out.println("Student " + (i + 1) + " -> Club " + (res[i] + 1));
        }
    }
}
```
Output:

```
PS D:\Java> java assign6.java
Enter number of students/clubs: 4
Enter the cost matrix:
4
4
3
2
2
1
2
6
7
5
9
4
3
2
2
5
The minimum cost of assigning clubs is: 11
Club assignments (student -> club):
Student 1 -> Club 3
Student 2 -> Club 1
Student 3 -> Club 4
Student 4 -> Club 2
```