Solutions

1. Ways to decode a string

```
//Recursive Solution
    public static int helper(String s, int i) {
         if(i == s.length())
             return 1;
         if(i > s.length())
             return 0;

         int ans = 0;
         if(s.charAt(i) != '0') {
             ans += helper(s, i+1);

             if(i < s.length() - 1) {
                 int num = (s.charAt(i)-'0')*10 + s.charAt(i+1)-'0';
                 if(num <= 26)
                     ans += helper(s, i+2);
             }
         } else {
           ans = 0;
         }
         return ans;
     }

    public static int waysToDecode(String str) {
           return helper(str, 0);
    }
```

```
//Memoized Solution
	public static int helper(String str, int i, int[] dp) {

		if(i == str.length())
			return 1;
		if(i > str.length())
			return 0;

		int ans = 0;
		if(str.charAt(i) != '0') {
			if(dp[i+1] == -1) {
				dp[i+1] = helper(str, i+1, dp);
			}
			ans = dp[i+1];

			if(i < str.length() - 1) {
		   int num = (str.charAt(i)-'0')*10 + str.charAt(i+1)-'0';
		   if(num <= 26) {
			if(dp[i+2] == -1)
				dp[i+2] = helper(str, i+2, dp);
			ans += dp[i+2];
		   }
	       }
		} else {
			dp[i] = 0;
		}
		return ans;
	}

	public static int waysToDecode(String str) {
		int[] dp = new int[str.length() + 1];
		for(int i = 0; i < str.length()+1; ++i)
			dp[i] = -1;
		return helper(str, 0, dp);
	}
```

```
//iterative dp
	public static int waysToDecode(String str) {
		int dp[] = new int[str.length()+1];
		dp[str.length()] = 1;
		for(int i = str.length()-1; i >= 0; --i) {
			 if(str.charAt(i) != '0') {
			dp[i] = dp[i+1];

			if(i < str.length() - 1) {
			    int num = (str.charAt(i)-'0')*10 + str.charAt(i+1)-'0';
			    if(num <= 26)
				dp[i] += dp[i+2];
			}
		  } else
			 dp[i] = 0;
		}
		return dp[0];
	}
```

```
//fully optimized
	public static int waysToDecode(String str) {
		int prevAns = 0;
		int ans = 1;
		
		for(int i = str.length()-1; i>=0; --i) {
			int currAns = 0;
			if(str.charAt(i) != '0') {
				currAns = ans;
				if(i < str.length() - 1) {
					int num = (str.charAt(i) - '0')*10 + 
str.charAt(i+1)-'0';
					if(num <= 26)
						currAns += prevAns;
				}
			}
			prevAns = ans;
			ans = currAns;
		}
		return ans;
	}
```

2. Edit Distance

```
//Recursive Solution
   public static int helper(String str1, String str2, int i, int j) {
          if(i == str1.length())
                 return str2.length() - j;
          if(j == str2.length())
                 return str1.length() - i;

          if(str1.charAt(i) == str2.charAt(j))
                 return helper(str1, str2, i+1, j+1);

          int temp1 = 1 + helper(str1, str2, i+1, j);
          int temp2 = 1 + helper(str1, str2, i, j+1);
          int temp3 = 1 + helper(str1, str2, i+1, j+1);

          return Math.min(temp3, Math.min(temp1, temp2));
   }

   public static int editDistance(String str1, String str2) {
          return helper(str1, str2, 0, 0);
   }
```

```
//iterative dp
      public static int editDistance(String A, String B) {
        int lenA = A.length(), lenB = B.length();
        int[][] dp = new int[lenA+1][lenB+1];
        for(int i = 0; i < lenA; ++i)
            dp[i][lenB] = lenA - i;
        for(int j = 0; j < lenB; ++j)
            dp[lenA][j] = lenB - j;

        for(int i = lenA-1; i >= 0; --i) {
            for(int j = lenB-1; j >= 0; --j) {
                char ch1 = A.charAt(i);
                char ch2 = B.charAt(j);
                if(ch1 == ch2)
                    dp[i][j] = dp[i+1][j+1];
                else {
                    dp[i][j] = 1 + Math.min(dp[i+1][j], dp[i][j+1]);
                    dp[i][j] = Math.min(dp[i][j], 1 + dp[i+1][j+1]);
                }
            }
        }
        return dp[0][0];
       }
```

3. Min Sum Path

```
//iterative dp solution, Time & Space Complexity - O(mn)
   public static int minPathSum(int[][] A) {
        int m = A.length;
        int n = A[0].length;
        int[][] dp = new int[m+1][n+1];
        for(int i = 0; i < m-1; ++i)
            dp[i][n] = Integer.MAX_VALUE;
        for(int j = 0; j < n-1; ++j)
            dp[m][j] = Integer.MAX_VALUE;
        for(int i = m-1; i >= 0; --i ) {
            for(int j = n-1; j >= 0; --j) {
                dp[i][j] = A[i][j] + Math.min(dp[i+1][j], dp[i][j+1]);
            }
        }
        return dp[0][0];
    }
```

```
//optimized dp solution with Time Complexity - O(mn)
        //Space Complexity - O(n)
        public int minPathSum(int[][] A) {
         int m = A.length;
         int n = A[0].length;
         int[] dp1 = new int[n+1];
         for(int i = 0; i < n-1; ++i)
             dp1[i] = Integer.MAX_VALUE;
         for(int i = m-1; i >= 0; --i) {
             int[] currArr = new int[n+1];
             currArr[n] = Integer.MAX_VALUE;
             for(int j = n-1; j >= 0; --j) {
                 currArr[j] = A[i][j] + Math.min(dp1[j], currArr[j+1]);
             }
             dp1 = currArr;
         }
         return dp1[0];
        }
```