Actividad 3:

Problema: Selección óptima de proyectos Eres el gerente de una empresa que debe decidir en qué proyectos invertir. Cada proyecto tiene un costo asociado y un beneficio esperado. Tienes un presupuesto limitado y necesitas elegir qué combinación de proyectos maximiza el beneficio total sin exceder el presupuesto. Requerimientos: Te proporcionarán un arreglo de costos que representa el costo de cada proyecto. También te proporcionarán un arreglo de beneficios que indica el beneficio que se espera de cada proyecto. Implementa un algoritmo que determine qué proyectos deben seleccionarse para maximizar el beneficio total sin exceder el presupuesto. Calcular utilizando algoritmos de fuerza bruta, greedy y programación dinámica. Indicar complejidades.

Greedy:

- **Complejidad Temporal: O(nlogn)**

```
package clase6_Activ3;
import java.util.*;

public class GreedyOpmProyecto {
   static class Proyecto{
        int costo;
        int beneficio;

        Proyecto(int costo,int beneficio){
            this.costo = costo;
            this.beneficio = beneficio;
        }
        double getRelacion (){
            return (double) beneficio/costo;
        }
   }

   static int maxBene(int[] costos,int[]
beneficio,int presupuesto){
        List<Proyecto> proyectos = new ArrayList<>();
        //crear lista de proyecto
        for (int i = 0;i < costos.length;i++){
            proyectos.add(new
Proyecto(costos[i],beneficio[i]));
```

```
        }

proyectos.sort(Comparator.comparingDouble(Proyecto::g
etRelacion).reversed());

        int totalBenef = 0;
        int totalCosto = 0;

        for (Proyecto proyecto : proyectos){
            if (totalCosto + proyecto.costo <=
presupuesto){
                totalCosto += proyecto.costo;
                totalBenef += proyecto.beneficio;
            }
        }
        return totalBenef;
    }
}
```

Fuerza Bruta:

- **Complejidad** : O(2^n)

```
package clase6_Activ3;

public class FzBrutaOpmProyecto {
   static int maxBenef(int[] costos, int[] beneficio,int
presupuesto,int n){//n es numero total de proyecto
       return maxBenefRec(costos,beneficio,presupuesto,n,0);
   }
   static int maxBenefRec(int[] costos,int[] beneficio, int
presupuesto,int n, int index){
       if (index == n || presupuesto <= 0){
           return 0;
       }
       int exclude = maxBenefRec(costos, beneficio,
presupuesto, n, index+1);

       int include = 0;
       if (costos[index] <= presupuesto){
           include = beneficio[index] + maxBenefRec(costos,
beneficio, presupuesto-costos[index], n, index+1);
```

```
        }
        return Math.max(include,exclude);
    }



}
```

Programación Dinámica:

- **Complejidad Temporal: O(n×presupuesto)**

```
package clase6_Activ3;

public class DinamOpmProyecto {

    public static void optimizarPresupuesto(int[]
costos,int[]rendimientos,int presupuesto){
        int n = costos.length;
        int[][] dp = new int [n+1][presupuesto+1];

        for (int i = 1;i<=n;i++){
            for (int j = 0;j<=presupuesto;j++){
                if (costos[i-1] <= j){
                    dp[i][j] = Math.max(dp[i-1][j], dp[i-1][j -
costos[i-1]] + rendimientos[i-1]);
                }else {
                    dp[i][j] = dp[i-1][j];
                }
            }
        }
        System.out.println("Maxima beneficio: " +
dp[n][presupuesto]);
        imprimirProyecto(dp,n,presupuesto,costos);
    }

    private static void imprimirProyecto(int[][] dp, int n, int
w, int[] costos){
        System.out.println("Proyecto Elegido:");
        for (int i = n; i>0 && w > 0; i--){
            if (dp[i][w] != dp[i-1][w]){
                System.out.print((i-1) + " ");
                w -= costos[i-1];
            }
        }
```

```
        System.out.println();
    }
}
```