ESTRUCTURAS DE DATOS Y ALGORITMOS I **2020-2**
PRACTICA # 11. ESTRATEGIAS PARA LA CONSTRUCCIÓN DE ALGORITMOS

OBJETIVO: APLICAR LOS ALGORITMOS BÁSICOS PARA LA SOLUCIÓN DE PROBLEMAS

DESARROLLO DE LA PRÁCTICA:

Actividad 1.- Ejercicios de la guía

En cada una de las actividades propuestas en la práctica, vas a trabajar con alguna estrategia de desarrollo de algoritmos

i.- Describe detalladamente la ejecución de cada uno de los programas, así como el por qué pertenecen a la categoría donde se encuentran, para cada uno de los problemas, indica a grandes rasgos si pudiera pertenecer a cualquier otra categoría vista en clase y por qué.
Nota: la sección de Medición y gráficas de los tiempos de ejecución es opcional y la sección Modelo RAM no es necesario realizarla

Actividad 2:

El problema de calendarización de actividades es un problema de optimización relativo a la selección de actividades "no conflictivas" a realizar dentro de un marco de tiempo determinado. Dado un conjunto de actividades cada una marcada por un tiempo de inicio y un tiempo de finalización. El problema es seleccionar el número máximo de actividades que pueden ser realizadas por una persona, suponiendo que sólo puede trabajar en una sola actividad a la vez

Una implementación simple para resolver el problema en lenguaje C es la siguiente:

```
void activities(int s[], int f[], int n){
	int i, j;
	printf ("Selected Activities are:\n");
	i = 0;
	printf("A%d ", i+1);
	for (j = 1; j < n; j++)
	{
		if (s[j] >= f[i])
		{
			printf ("A%d ", j+1);
			i = j;
		}
	}
}
void main(){
	int s[] =  {1, 2, 3, 2, 4, 5, 6, 8, 7}; //horarios inicio de  actividades
	int f[] =  {4, 5, 6, 8, 6, 7, 7, 12, 9}; //horario fin de actividades
	int n = sizeof(s)/sizeof(s[0]);
	activities(s, f, n);
	getchar();
}
```

i.- Indica a cuál de las estrategias de construcción de algoritmos pertenece el problema y realiza la implementación en PYTHON.

CONTINÚA EN LA PARTE POSTERIOR…

3. CODIFICA y ejecuta el siguiente programa, indica la salida, lo que hace el programa y porqué pertenece a la categoría: Divide y vencerás

```
def MinMax(L):
  if len(L) == 1:
    return (L[0], L[0])
  elif len(L) == 2:
    if L[0] <= L[1]:
      return (L[0], L[1])
    else:
      return (L[1], L[0])
  else:
    mid = len(L) / 2
    (minL, maxL) = MinMax(L[:int(mid)])
    (minR, maxR) = MinMax(L[int(mid):])
    if minL <= minR:
      min = minL
    else:
      min = minR
    if maxL >= maxR:
      max = maxL
    else:
      max = maxR
    return (min, max)
def main():
    lista=[3, 10, 32, 100, 4, 76, 45, 32, 17, 12, 1];
    print("los valores son: ",MinMax(lista))
main()
```

4.- A continuación, se muestra una implementación del algoritmo de ordenamiento por "merge sort". Codifica y ejecuta el programa y describe por qué pertenece a la categoría divide y vencerás

```
def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0
    while left_idx < len(left) and right_idx < len(right):
        if left[left_idx] <= right[right_idx]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    if left:
        result.extend(left[left_idx:])
    if right:
        result.extend(right[right_idx:])
    return result
def merge_sort(m):
    if len(m) <= 1:
        return m
    middle = len(m) // 2
    left = m[:middle]
    right = m[middle:]
    left = merge_sort(left)
    right = merge_sort(right)
    return list(merge(left, right))
lista1=[4,12,87,1,32,54,36,78,90,7]
print(merge_sort(lista1))
```

5.- Elabora las conclusiones de tu práctica