~~~ QUIEN TIENE MIEDO A LA DERROTA, YA HA SIDO DERROTADO ~~~

APELLIDO, NOMBRE: ..... PADRON: .....

MAIL: ..... ENTREGO ..... HOJAS.

1.a	1.b	2	3.a	3.b	4	NOTA

Antes de empezar a resolver el examen lea las siguientes aclaraciones:

- Complete sus datos en esta hoja. Firme, numere e inicialice con nombre, apellido y padrón todas sus hojas.
- Léalo **todo** a conciencia, y haga preguntas sobre lo que no entiende en el espacio designado para ello.
- Recomendamos fuertemente realizar un análisis de **cada** ejercicio.
- Para aprobar es necesario tener bien, al menos, el 60% de todo el examen.
- Los ejercicios 1 y 2 no pueden estar mal.

EJERCICIOS

1. **Análisis de algoritmos**

   a. Sean:
   - $T1(n) = 2T(n/2) + O(1)$
   - $T2(n) = 2T(n/2) + O(n)$

   Determinar y demostrar cuál de los dos algoritmos tiene mejor tiempo de ejecución.

   b. Determinar el orden de crecimiento de la siguiente función:

```
void ejercicio_uno(int n){
     int i, j, k, contador=0;

     for( i=0; i<=n; i++)
          for( j=n ; j>=0; j--)
               for( k=1; k<n; k*=2)
                    contador++;

}
```

2. **Ordenamientos**

   a. Dado el siguiente vector:

6	3	9	13	88	33	56	2	4	21	57

   Ordenarlo ascendentemente mediante **quicksort** y **mergesort** mostrando todos los pasos intermedios.
~~~

3. **Aritmética de punteros**

Dado el siguiente algoritmo:

```
const size_t MAX_VECTOR = 5;

int main(){
	char*** frase = malloc(MAX_VECTOR*sizeof(char*));
	char letra = 'P';

	for (size_t i = 0; i < MAX_VECTOR; i++){
		frase[i] = malloc((i+1)*sizeof(char*));
		char* p_letra = malloc(sizeof(char));
		*p_letra = letra;
		for (size_t j = 0; j < i+1; j++){
			frase[i][j] = p_letra;
		}
		letra++;
	}

	*(frase[0][0]) = 'A';
	*(frase[1][0]) = *(frase[0][0]) + 10;

	printf("%c%c%c%c%c\n", frase[3][0][0], frase[4][0][0],
		frase[0][0][0], frase[2][0][0], frase[1][0][0]);

	for (size_t i = 0; i < MAX_VECTOR; i++){

		free(frase[i]);

	}
	free(frase);
	return 0;
}
```

a. Realice un diagrama del stack y del heap y como varían los datos y la memoria con la ejecución del programa. ¿Qué se imprime por pantalla?
b. ¿El programa pierde memoria? En caso afirmativo, cuanta y como lo arreglaría. En caso negativo, muestre en su diagrama donde fue liberada toda la memoria reservada.

4. **Recursividad**

Sea una *pila_t* un tda pila que almacena enteros y cuenta con las operaciones apilar, desapilar y vacía. Y sea **invertir_y_mostrar** un un procedimiento que imprime los elementos de la pila al revés. *Aclaración: El desapilar, además de sacarlo de la pila, devuelve el elemento.*

```
void invertir_y_mostrar(pila_t pila){
	pila_t pila_auxiliar;

	while (!vacia(pila))
		apilar(pila_auxiliar, desapilar(pila));

	while (!vacia(pila_auxiliar))
		printf("%i\n", desapilar(pila_auxiliar));

}
```

Se pide encontrar un algoritmo, que reciba los parámetros que crea convenientes y cumpla el mismo objetivo que **invertir_y_mostrar**, pero de manera recursiva.