Algoritmos & Estructuras de Datos

Guía Práctica

Universidad de San Andrés

Magalí Marijuán I. Javier Mermet Matías Sandacz

2 de diciembre de 2022

Índice general

Parte I

Introducción

Capítulo 1

Lenguaje C

1.1 Ejercicios para entrar en calor

1. Escribir la función que dado n ∈ N devuelve si es primo. Recuerden que un número es primo si los únicos divisores que tiene son 1 y el mismo.

2. Escribir la función que dado n ∈ N devuelve la suma de todos los números impares menores que n.

1.2 Punteros y Arreglos

3. ¿Cuál es el valor de a y b luego de ejecutar el programa?

```
void myFunc(int* a, int b)
{
(*a)++;
b++;
}

void main()
{
    int a = 10;
    int b = 10;
    myFunc(&a, b);
}
```

4. ¿Que valor se imprime por consola luego de cada llamado a *printf*?

```
void main()
{
   int x = 10;
   int* px = &x;

   printf("%d \n", px);
   printf("%d \n", (*px));
```

```
    (*px)++;

    printf("%d \n", px);
    printf("%d \n", (*px));
}
```

5. Programar las siguientes funciones en C:

   (a) `void crearArreglo(int v)`

   Crea un arreglo estático de enteros de tamaño 8, inicializando todos sus elementos con v, y lo imprime en pantalla.

   (b) `int* crearArregloDin(int n, int v)`

   Dado un natural n, crea un arreglo de enteros de ese tamaño, inicializando todos sus elementos con v, y devuelve un puntero al mismo.

   (c) Invocar la siguiente función con cualquiera de los arreglos inicializados anteriormente y convencerse de que sus elementos están ubicados de manera **contigua** en memoria. *Recordar que cada elemento de tipo int ocupa 4 bytes.*

```
void mostrarMemoria(int* arr, int size)
{
    for(int i=0; i<size; i++)
    {
        printf("Elemento: %d, Direccion: %d\n", i, &arr[i]);
    }
}
```

6. Cuál es la diferencia entre **malloc** y **calloc**? Cuando utilizamos la función **free**?

7. Dado el siguiente struct que representa una Persona

```
typedef struct Persona{
    int edad;
    char* nombre;
} Persona;
```

```
Persona* inicializarPersonas(int n)
```

Completar la función inicializarPersonas, que dado un entero n, crea un arreglo de n personas, y devuelve un puntero al mismo.

   (a) Utilizando **malloc**.

   (b) Utilizando **calloc**.

1.2.1 Más arreglos!

8. Programar las siguientes funciones en C:

   (a) `int maximo(int* arr, int size)`

   Dado un arreglo de enteros *arr* de tamaño *size*, devuelve el máximo elemento.

(b) `void sumador(int* arr, int size, int c)`

Dado un arreglo de enteros *arr* de tamaño *size*, y un entero *c*, suma *c* a todos los elementos de *arr*.

(c) `char* copiar(char* arr)`

Crea una copia de *arr*, y devuelve un puntero a la copia.

(d) `int* reverso(int* arr, int size)`

Dado un arreglo de enteros *arr* de tamaño *size*, devuelve su reverso.
Ejemplo: Dado [1, -2, 85, 65] se debe devolver [65, 85, -2, 1].

i. Se puede modificar *arr*.

ii. Sin modificar *arr*.

(e) `bool estaOrdenado(int* arr, int size)`

Dado un arreglo *arr* de enteros de tamaño *size*, retorna true si es monótonamente creciente o monótonamente decreciente.

(f) `bool esPalindromo(char* s)`

Dado un string *s*, retorna true si es un palíndromo. *Recuerden que un palíndromo es una palabra que se lee igual en un sentido que en otro (por ejemplo; Ana, Anna, Otto).*

1.3 Búsqueda y Ordenamiento

9. `void ordenar(int* arr, int size)`

Dado un arreglo de enteros *arr* de tamaño *size*, escribir una función que lo ordene de menor a mayor y dar su complejidad temporal.

(a) Utilizando ***selection sort***.

(b) Utilizando ***insertion sort***.

Suponiendo que queremos obtener únicamente los $k < size$ menores elementos del arreglo, ¿cuál de los dos algoritmos utilizaría? ¿Por qué?

10. `int buscar(int* arr, int size, int target)`

Dado un arreglo de enteros *arr* de tamaño *size* y un entero *target*, escribir una función que busque al *target* en *arr*. Si existe, entonces devolver su índice. En caso contrario, devolver -1.
Por ejemplo, dado el arreglo [8,4,9,1] y el target 9, se debe devolver 2. En cambio, si el target es 10, se debe devolver -1.

11. Repetir el ejercicio anterior, pero ahora suponiendo que *arr* está ordenado. El algoritmo debe tener complejidad $O(logn)$.

12. `bool existe(int* arr, int size, int target)`

Dado un arreglo de enteros sin repetidos *arr* de tamaño *size*, se debe devolver true si existen dos elementos $a1$ y $a2$ tal que $a1 + a2 = target$. En caso contrario, devolver false.

(a) Dar una solución con complejidad temporal $O(n^2)$.

(b) Dar una solución con complejidad temporal $O(nlogn)$.
Sugerencia: Ordenar el arreglo.

13. `int* buscarRango(int* arr, int size, int target)`

Dado un arreglo de enteros *arr* de tamaño *size*, ordenado de menor a mayor, encontrar los índices de comienzo y de final de *target*. Devolverlos en un arreglo. Además, se puede suponer que *target* es un elemento de *arr*.
Por ejemplo, si arr = [5,7,7,8,8,8,10], target = 8, se debe devolver [3,5].

(a) El algoritmo dado debe tener complejidad $O(n)$.

(b) (Dificil) El algoritmo dado debe tener complejidad $O(logn)$.
Sugerencia: Utilizar búsqueda binaria

Capítulo 2

Recursividad

1. `int fibo(int n)`

   Los números de Fibonacci son una sucesión de números naturales definida por las ecuaciones:

   $$F_n = F_{n-1} + F_{n-2}$$

   con $F_1 = F_2 = 1$. Dar una función recursiva para computar el n-ésimo número de Fibonacci.

Capítulo 3

Lectura de Archivos

Capítulo 4

Magia con Punteros

1. ```
   typedef struct Persona {
       char* nombre;
       char* apellido;
       char* domicilio;
       int edad;
   } Persona;

   char* masGrande(Persona** personas, int n)
   {
       // COMPLETAR
   }
   ```

   Completar la función *masGrande*, que dado un arreglo de n personas, devuelve el nombre de la persona de mayor edad.

2. `void imprimirMatriz(int** matrix, int n, int m)`

   Dada una matriz de n filas y m columnas, imprimir todos sus elementos en pantalla. Se deben imprimir todos los elementos de la primer fila, luego todos los de la segunda fila, y así sucesivamente. Por ejemplo, dada la matriz de $2x3$ [[1,2,3], [4,5,6]], se debe imprimir 1, 2, 3, 4, 5, 6.

3. `void traspuesta(int** matrix, int n)`

   Dada una matriz cuadrada de nxn, modificarla para obtener su traspuesta. Por ejemplo, dada la matriz cuadrada [[1,2], [3,4]], se debe modificar para obtener [[1, 3], [2, 4]]. *Recuerden que la matriz traspuesta es aquella que surge como resultado de realizar un cambio de columnas por filas y filas por columnas en la matriz original.*
   ```

Parte II

Estructuras de Datos

Capítulo 5

Listas

Capítulo 6

Pilas

Capítulo 7

Colas

Capítulo 8

Heaps

Capítulo 9

Árboles

Capítulo 10

Grafos

Capítulo 11

Hashing