UNIVERSIDAD NACIONAL DE INGENIERÍA

Facultad de Ciencias
Escuela Profesional de Ciencia de la Computación

Curso: Fundamentos de Programación CC112 Semestre 2025-II

Laboratorio 3

Algoritmos de ordenamiento y búsqueda II

1. Escriba un programa que calcule en forma recursiva una matriz $A_{n\times n}$. Los elementos $a_{i,j}$ sobre la diagonal de A son nulos y los elementos restantes se forman de la suma $a_{i-1,j-1} + a_{i-1,j}$. Ejemplo de salida

```
1   0   0   0   0   0   0
1   1   0   0   0   0   0
1   2   1   0   0   0   0
1   3   3   1   0   0   0
1   4   6   4   1   0   0
1   5  10  10   5   1   0
1   6  15  20  15   6   1
```

2. Ingrese por teclado una lista de números diferentes (máxima cantidad de números 30) de tal forma que la lista tenga en los índices pares una secuencia ascendente y en los índices impares otra secuencia ordenada en forma descendente. Implemente el método de búsqueda binaria que permita encontrar el índice de un número buscado en dicha lista.

3. Dado dos arreglos arr1[] y arr2[] de números enteros ordenados en forma ascendente, cuyos tamaños son n1 y n2 respectivamente (tamaño máximo 20). Escribe una función llamada **mezcla** que fusione ambos arreglos en un solo arreglo ordenado. Ejemplo
   Entrada:
   arr1[] = $\{1, 4, 6, 9\}$
   arr2[] = $\{2, 3, 7, 8, 10\}$

   Salida:
   arr[] = $\{1, 2, 3, 4, 6, 7, 8, 9, 10\}$

4. Implementa el algoritmo de Merge Sort para ordenar un arreglo de números enteros en orden ascendente. El programa debe:

   - Leer `n` elementos desde la entrada estándar.
   - Imprimir el arreglo antes y después de ordenarlo.

- Utilizar una implementación recursiva y modularizada, separando claramente las funciones de división y fusión del arreglo.

Además, debe extender su funcionalidad para:

a) Mostrar el contenido del arreglo en cada llamada recursiva, antes y después de cada proceso de fusión.

b) Contabilizar el número total de comparaciones realizadas durante la ejecución del algoritmo.

c) Modificar el algoritmo para realizar la ordenación en orden descendente.

d) Adaptar Merge Sort para ordenar dos arreglos relacionados: uno con nombres y otro con sus respectivas notas, ordenando por el criterio de nota ascendente.

e) Comparar el tiempo de ejecución de Merge Sort con Bubble Sort utilizando arreglos grandes, analizando la eficiencia de ambos algoritmos.

5. Implementar una función llamada particion que tome un arreglo, un índice de inicio y un índice de fin, y reorganice el arreglo de tal manera que todos los elementos menores que el pivote se encuentren a su izquierda y todos los elementos mayores que el pivote se encuentren a su derecha.

   La función de partición tomará el primer elemento del arreglo como el pivote. Todos los elementos menores que el pivote deben ir a la izquierda del pivote. Todos los elementos mayores que el pivote deben ir a la derecha del pivote. El pivote debe estar en su posición final (donde todos los elementos menores están a su izquierda y todos los mayores están a su derecha). Ejemplo:
   Entrada:
   Arreglo: [12, 9, 3, 5, 2, 8, 7, 1]
   Índice de inicio: 0
   Índice de fin: 7

   Salida:
   Arreglo después de partición: [1, 9, 3, 5, 2, 8, 7, 12]
   Índice del pivote: 7

6. Implementa el algoritmo de quickSort para ordenar un arreglo de números enteros en orden ascendente.

   a) Imprime el contenido del arreglo en cada llamada recursiva a `quicksort()` (antes de dividir).

   b) Cuenta la cantidad total de comparaciones realizadas durante el proceso de ordenamiento.

   c) Modifica el algoritmo para ordenar en orden descendente.

   d) Adapta quickSort para ordenar dos arreglos paralelos (por ejemplo: notas[] y nombres[]) de acuerdo al criterio del arreglo de notas (orden ascendente).

   e) Compara el tiempo de ejecución de quickSort con bubbleSort y nergeSort, usando arreglos grandes generados aleatoriamente.

7. Implementar una función que reciba un arreglo de números (no necesariamente ordenado), y realice una búsqueda binaria adaptada para encontrar un número dado. Ejemplo:
   Entrada:
   Arreglo: [5, 10, 3, 8, 2, 6, 1, 7, 4, 9]
   Número a buscar: 6

   Salida:
   El número 6 se encuentra en el índice: 5

8. Pepe tiene n peras en una canasta. Cada día sigue las siguientes reglas:

   - Si el número actual de peras es divisible por 2, puede comerse la mitad.
   - Si el número actual de peras es divisible por 5, puede comerse cuatro quintas partes de ellas.
   - En caso contrario, puede comerse exactamente 2 peras.

   a) Plantee y justifique la relación de recurrencia para $f(n)$, el mínimo número de días para comerse todas las peras.

   b) Implemente un algoritmo recursivo que resuelva el problema.

   c) Calcule manualmente $f(10)$ mostrando el razonamiento paso a paso.