Técnico en Ciberseguridad

Facilitador: Kevin Feliz Henríquez

Nombre del Módulo: Estructuras de Datos y Algoritmos

Nombre Actividad: Actividad 2

Descripción: En esta actividad, abordaremos los algoritmos computacionales (, etc), y pondremos todo esos elementos en práctica con una actividad interactiva que servirá como instrumento de aprendizaje.

Actividad #2

Algoritmos

Un **algoritmo** es un conjunto de instrucciones bien definidas, ordenadas y finitas que resuelven un problema o logran una tarea específica.

Son la base de cualquier programa, sistema y servicio computacional que se usa día a día.

Actividad #2

Búsqueda Lineal (Linear Search)

La **búsqueda lineal** es uno de los algoritmos de búsqueda más simples.

Consiste en recorrer una lista o arreglo elemento por elemento, comparando cada uno con el valor que se está buscando.

Si el valor coincide, se devuelve la posición del elemento; si se llega al final de la lista sin encontrarlo, se reporta que no está presente.

Es útil para listas no ordenadas.

Actividad #2

Ejemplo (Busqueda Lineal):

```
function busquedaLineal(arr, objetivo) {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === objetivo) {
      return i; // Retorna el índice si lo encuentra
    }
  }
  return -1; // Retorna -1 si no lo encuentra
}

let numeros = [3, 8, 1, 6, 2, 7];
let indice = busquedaLineal(numeros, 6);
console.log(indice); // Salida: 3
```

Actividad #2

Búsqueda Binaria (Binary Search)

La **búsqueda binaria** es un algoritmo de búsqueda más eficiente, pero requiere que la lista o arreglo esté **ordenado**.

Funciona dividiendo repetidamente el arreglo a la mitad. Compara el valor del elemento central con el valor que se busca.

Si el valor del centro es el objetivo, la búsqueda termina.

Si no, decide si el objetivo está en la mitad izquierda o derecha del arreglo y repite el proceso en esa mitad.

Actividad #2

```
function busquedaBinaria(arr, objetivo) {
  let inicio = 0;
  let fin = arr.length - 1;

  while (inicio <= fin) {
    let medio = Math.floor((inicio + fin) / 2);
    if (arr[medio] === objetivo) {
      return medio;
    } else if (arr[medio] < objetivo) {
      inicio = medio + 1;
    } else {
      fin = medio - 1;
    }
  }
  return -1;
}
```

```
let numerosOrdenados = [1, 2, 3, 6, 7, 8];

let indiceBinario =
busquedaBinaria(numerosOrdenados, 6);

console.log(indiceBinario); // Salida: 3
```

Actividad #2

Algoritmos de Ordenamiento (Sorting Algorithms)

Los algoritmos de ordenamiento se utilizan para reorganizar elementos de una lista en un orden específico, como de menor a mayor.

Actividad #2

Ordenamiento por Burbuja (Bubble Sort)

El **ordenamiento por burbuja** es un algoritmo simple que funciona revisando la lista varias veces y comparando cada par de elementos adyacentes, intercambiándolos si están en el orden incorrecto.

Este proceso se repite hasta que la lista esté completamente ordenada. Es fácil de entender, pero muy ineficiente para listas grandes.

Actividad #2

```
function bubbleSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        // Intercambiar elementos
        let temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
  return arr;
}
```

```
let numerosDesordenados = [64, 34, 25, 12, 22, 11, 90];
let numerosOrdenados = bubbleSort(numerosDesordenados);
console.log(numerosOrdenados); // Salida: [11, 12, 22, 25, 34, 64, 90]
```

Actividad #2

Ordenamiento Rápido (Quicksort)

El **Quicksort** es un algoritmo de ordenamiento muy popular y eficiente que sigue la estrategia de "divide y vencerás".

El proceso funciona de la siguiente manera:

1. Se elige un elemento del arreglo, llamado **pivote**.
2. Se **particiona** el arreglo, reordenando sus elementos de forma que los valores menores que el pivote queden a su izquierda y los mayores a su derecha.
3. Se aplica este mismo proceso de forma **recursiva** a los sub-arreglos de la izquierda y la derecha hasta que el arreglo esté completamente ordenado.

Actividad #2

```
function quicksort(arr) {
  if (arr.length <= 1) {
    return arr;
  }

  const pivot = arr[0];
  const izquierda = [];
  const derecha = [];

  for (let i = 1; i < arr.length; i++) {
    if (arr[i] < pivot) {
      izquierda.push(arr[i]);
    } else {
      derecha.push(arr[i]);
    }
  }

  return [...quicksort(izquierda), pivot, ...quicksort(derecha)];
}

const numeros = [5, 3, 7, 6, 2, 8, 4];
const numerosOrdenados = quicksort(numeros);
console.log(numerosOrdenados); // Salida: [2, 3, 4, 5, 6, 7, 8]
```

Actividad #2

Ordenamiento por Mezcla (Merge Sort)

El **Merge Sort** también es un algoritmo de "divide y vencerás".

A diferencia de Quicksort, este no elige un pivote, sino que divide el arreglo en mitades repetidamente hasta que cada sub-arreglo contenga solo un elemento (y por lo tanto, esté ordenado).

Luego, fusiona estos sub-arreglos de forma ordenada hasta que se reconstruye el arreglo original.

Su principal ventaja es que es un algoritmo estable y su rendimiento no se degrada en el peor de los casos.

Actividad #2

Notas:

Tanto **Quicksort** como **Merge Sort** son significativamente más rápidos que **Bubble Sort** para arreglos grandes.

Aunque Quicksort es generalmente más rápido en la práctica, Merge Sort es preferido en situaciones donde se necesita un rendimiento consistente y garantizado.

Actividad #2

```
function mergeSort(arr) {
  if (arr.length <= 1) {
    return arr;
  }

  const medio = Math.floor(arr.length / 2);
  const izquierda = arr.slice(0, medio);
  const derecha = arr.slice(medio);

  return merge(mergeSort(izquierda), mergeSort(derecha));
}
```

```
function merge(izquierda, derecha) {
  let resultado = [];
  let i = 0;
  let j = 0;

  while (i < izquierda.length && j < derecha.length) {
    if (izquierda[i] < derecha[j]) {
      resultado.push(izquierda[i]);
      i++;
    } else {
      resultado.push(derecha[j]);
      j++;
    }
  }

  return [...resultado, ...izquierda.slice(i), ...derecha.slice(j)];
}
```

```
const numeros2 = [5, 3, 7, 6, 2, 8, 4];
const numerosOrdenados2 = mergeSort(numeros2);
console.log(numerosOrdenados2);
// Salida: [2, 3, 4, 5, 6, 7, 8]
```