SESIÓN 22

Algoritmos / Búsqueda binaria

La búsqueda binaria es un método rápido y eficiente para encontrar un valor específico en una lista ordenada. Divide repetidamente la lista por la mitad y compara el valor buscado con el elemento central. Si coincide, se devuelve la posición. Si es menor, se busca en la mitad inferior; si es mayor, en la superior. Este proceso se repite hasta encontrar el valor o agotar la lista. Con una complejidad de tiempo O(log n), es ideal para grandes conjuntos de datos.

js

Copiar código

```
// Definición de la función de búsqueda binaria
function BusquedaBi(datos, valor){
  // Inicialización de los punteros izquierdo y derecho
  let izq = 0;
  let der = datos.length - 1;

  // Ciclo de búsqueda binaria mientras izquierda sea menor o igual a derecha
  while(izq <= der){
    // Cálculo de la mitad del arreglo y redondeo al entero más cercano
    let mitad = Math.floor((izq + der) / 2);
    // Obtención del valor en la posición de la mitad
    let dato = datos[mitad];

    // Comprobación si el valor en la mitad es igual al valor buscado
    if(dato == valor){
      // Se encontró el valor, se devuelve la posición de la mitad
      return mitad;
    }
    // Si el valor buscado es mayor al valor en la mitad
    else if(valor > dato){
      // Se ajusta el puntero izquierdo para buscar en la mitad derecha del arreglo
      izq = mitad + 1;
    }
    // Si el valor buscado es menor al valor en la mitad
    else{
      // Se ajusta el puntero derecho para buscar en la mitad izquierda del arreglo
      der = mitad - 1;
    }
  }

  // Si no se encontró el valor en el arreglo, se devuelve -1
  return -1;
}

const arreglo = [1,2,3,4,5,6,7,8,9,10,11,12,13,23,34,45,56];

let num = 11;
```

```
console.log(BusquedaBi(arreglo, num ));
```

js

Copiar código

```
// Ejemplo de búsqueda binaria 2
function BusquedaBinaria(arr, izq, der, n){
  if(izq > der) return -1;

  const mitad = Math.floor((izq + der) / 2);

  if(arr[mitad] == n) return mitad;

  if(arr[mitad] < n){
    return BusquedaBinaria(arr, mitad + 1, der, n)
  }else{
    return BusquedaBinaria(arr, izq, mitad -1, n);
  }
}

let arr = [1,2,3,4,5,6,7,8,9,10,11,12,13,23,34,45,56];
console.log(BusquedaBinaria(arr, 0, arr.length -1, 3));
```

Algoritmos / Quicksort

Quicksort es un algoritmo de ordenamiento eficiente y rápido que utiliza la estrategia de dividir y conquistar. Funciona seleccionando un elemento pivote de la lista y particionando los elementos restantes en dos subconjuntos: aquellos menores que el pivote y aquellos mayores que el pivote. Luego, se aplica recursivamente Quicksort a cada subconjunto. Este proceso continúa hasta que todos los elementos estén ordenados. Quicksort tiene un tiempo de ejecución promedio de O(n log n) y es ampliamente utilizado en la práctica debido a su eficiencia y simplicidad.

js

Copiar código

```
// Algoritmo quicksort
function quicksort(num){
```

```
  if(num.length <= 1){
    return num;
  }

  let izq = [];
  let der = [];
  let aux = [];
  let pivote = num.pop();
  let n = num.length;

  for(let i = 0; i < n; i++ ){
    if(num[i] <= pivote){
      izq.push(num[i]);
    }else{
      der.push(num[i]);
    }
  }
  return aux.concat(quicksort(izq), pivote, quicksort(der));
}

let arr = [14, 23, 3, 4, 54, 6, 7, 8, 9, 1, 11, 12, 13, 22, 34, 32, 56,65,16,78,87,98];
console.log(arr);

let res = quicksort(arr);
console.log(res);
```

js

Copiar código

```
// Ejemplo dos
const QSort = (nums) => {
  // caso base que parará la recursión ya que un arreglo que solo contiene
  // un elemento, ya está ordenado
  if (nums.length < 2) return nums;

  const pivot = nums[0];
  const pequeño = [];
  const grande = [];

    // ya que escogimos el primer elemento como pivote,
    // empezamos a recorrer el arreglo desde el segundo elemento
    for(let i = 1; i < nums.length; i++) {
    if(nums[i] < pivot) pequeño.push(nums[i]);
    else grande.push(nums[i]);
  }

  // puedes unir las sub-listas ordenadas usando spread de ES6 o .concat(
  return [...QSort(pequeño), pivot, ...QSort(grande)];
}

let arr = [14, 23, 3, 4, 54, 6, 7, 8, 9, 1, 11, 12, 13, 22, 34, 32, 56,65,16,78,87,98];
console.log(arr);

let res = QSort(arr);
console.log(res);
```

Aprende más sobre Quicksort

o mira un ejemplo de código Quicksort aquí

Algoritmos / MergeSort()

MergeSort es un algoritmo de ordenación eficiente implementado en JavaScript que divide una lista en subconjuntos más pequeños hasta que cada uno contiene un solo elemento. Luego, combina recursivamente estos subconjuntos en orden hasta que se obtiene una lista completamente ordenada. Utiliza la técnica de "divide y conquista", lo que significa que divide el problema en partes más pequeñas y resuelve cada parte por separado. Debido a su complejidad de tiempo O(n log n) en el peor caso, MergeSort es ideal para ordenar grandes conjuntos de datos de manera eficiente.

js

Copiar código

```js
// Implementación Merge()
function mergesort(izq, der){
  let i = 0;
  let d = 0;
  let res = [];

  while (i < izq.length || d < der.length){
    if(i === izq.length){
      res.push(der[d]);
      ++d;
    }else if(d === der.length || izq[i] <= der[d]){
      res.push(izq[i]);
      ++i;
    }else{
      res.push(der[d]);
      ++d;
    }
  }
  return res;
}

let num = [2,11,23,45,67];
let num2 = [3,12,24,54,87, 93];
```

```
let rest = mergesort(num, num2);
console.log(rest);
console.log(rest.length);
```

Aprende más sobre MergeSort()

o mira un ejemplo de código MergeSort() aquí