Algoritmos de Ordenamiento

Para realizar la medición de tiempo de los algoritmos de ordenamiento, utilice el método performance.now() el cual es muy preciso.

Bubble Sort (341.5 milisegundos) [Codigo en Github]

```js
function generarNumerosAleatorios() {
  const numerosAleatorios = [];
  for (let i = 0; i < 10000; i++) {
    const numeroAleatorio = Math.floor(Math.random() * 10001);
    numerosAleatorios.push(numeroAleatorio);
  }
  return numerosAleatorios;
}

function bubbleSort(arr) {
  let n = arr.length;
  let swapped;
  do {
    swapped = false;
    for (let i = 1; i < n; i++) {
      if (arr[i - 1] > arr[i]) {
        let temp = arr[i - 1];
        arr[i - 1] = arr[i];
        arr[i] = temp;
        swapped = true;
      }
    }
    n--;
  } while (swapped);
  return arr;
}

function medirTiempoDeEjecucion(func, arr) {
  const t0 = performance.now();
  func(arr);
  const t1 = performance.now();
  return t1 - t0;
}

const arrayNumeros = generarNumerosAleatorios();
console.log("Antes de ordenar:", arrayNumeros);

const tiempoEjecucion = medirTiempoDeEjecucion(bubbleSort, arrayNumeros);
console.log(
  `Tiempo de ejecución del Bubble Sort: ${tiempoEjecucion} milisegundos`
);

const arrayOrdenado = bubbleSort(arrayNumeros);
console.log("Después de ordenar:", arrayOrdenado);
```

```
Antes de ordenar:                                                    Bubblesort.js:39
  (10000) [8585, 9598, 7109, 3482, 2652, 3802, 2907, 582, 6274, 9095, 2164, 6082, 6670, 3956, 3411, 2735, 5766, 8238, 6627, 5692, 1236, 7275, 3005, 6965,
  3423, 1237, 2330, 3062, 9237, 7554, 95, 689, 4248, 3870, 2567, 5460, 5238, 1747, 5001, 4112, 7741, 7431, 5776, 2148, 9466, 3067, 7264, 7343, 774, 4120,
  7230, 3305, 7773, 2807, 1943, 3937, 7102, 9628, 7603, 2899, 1831, 3358, 7412, 9294, 5996, 6618, 866, 8449, 6890, 1414, 8576, 1894, 3137, 997, 2166, 942
  0, 241, 5746, 2881, 7234, 4524, 5901, 607, 8513, 3816, 2605, 4975, 4856, 1167, 3831, 8835, 2342, 713, 2809, 9374, 3627, 8027, 7106, 4322, 3593, …]
Tiempo de ejecución del Bubble Sort: 354.30000001192093 milisegundos   Bubblesort.js:43
Después de ordenar:                                                  Bubblesort.js:49
  (10000) [0, 1, 3, 4, 4, 5, 5, 5, 6, 9, 11, 13, 14, 16, 17, 18, 19, 19, 19, 20, 22, 23, 23, 24, 25, 26, 27, 28, 29, 31, 31, 35, 35, 36, 39, 39, 39, 40,
  41, 41, 41, 42, 43, 43, 44, 47, 48, 49, 49, 50, 51, 51, 54, 55, 56, 58, 58, 59, 59, 59, 60, 61, 62, 66, 67, 67, 67, 68, 68, 68, 69, 69, 70, 70, 71, 71,
  72, 72, 73, 73, 73, 74, 75, 77, 77, 80, 80, 81, 81, 82, 82, 83, 84, 86, 88, 88, 88, 90, 90, 91, …]
```

Selection Sort (40.59) milisegundos [Codigo en Github]

```js
function generarNumerosAleatorios() {
  const numerosAleatorios = [];
  for (let i = 0; i < 10000; i++) {
    const numeroAleatorio = Math.floor(Math.random() * 10001);
    numerosAleatorios.push(numeroAleatorio);
  }
  return numerosAleatorios;
}

function selectionSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    let minIndex = i;
    for (let j = i + 1; j < n; j++) {
      if (arr[j] < arr[minIndex]) {
        minIndex = j;
      }
    }
    if (minIndex !== i) {
      let temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
    }
  }
  return arr;
}

function medirTiempoDeEjecucion(func, arr) {
  const t0 = performance.now();
  func(arr);
  const t1 = performance.now();
  return t1 - t0;
}

const arrayNumeros = generarNumerosAleatorios();
console.log("Antes de ordenar:", arrayNumeros);

const tiempoEjecucion = medirTiempoDeEjecucion(selectionSort, arrayNumeros);
console.log(
  `Tiempo de ejecución del Selection Sort: ${tiempoEjecucion} milisegundos`
);

const arrayOrdenado = selectionSort(arrayNumeros);
console.log("Después de ordenar:", arrayOrdenado);
```

```
Antes de ordenar:                                                    Selectionsort.js:38
  (10000) [2979, 577, 7067, 9191, 8625, 4374, 2991, 8251, 8654, 7544, 359, 8088, 4197, 1197, 6978, 996, 1159, 5448, 4190, 2048, 657, 783, 2773, 2422, 910
▶ 0, 5136, 7139, 2914, 9284, 4780, 5342, 9174, 7423, 2132, 7141, 2495, 2527, 6358, 4381, 5264, 724, 2643, 8526, 4103, 9725, 6808, 1092, 8427, 3797, 8858,
  1508, 7589, 2457, 1898, 8276, 5544, 5314, 5541, 1418, 6867, 3535, 8962, 2380, 8012, 4343, 5812, 3886, 2700, 4622, 4861, 8535, 6963, 1530, 6086, 4448, 8
  917, 5151, 1800, 739, 2278, 9919, 7017, 7109, 2060, 526, 8390, 8899, 1256, 3625, 9048, 3163, 3182, 3859, 5354, 2325, 4979, 739, 8639, 7394, 5485, …]

Tiempo de ejecución del Selection Sort: 40.599999994039536 milisegundos   Selectionsort.js:42

Después de ordenar:                                                  Selectionsort.js:48
  (10000) [0, 3, 4, 7, 7, 10, 11, 11, 11, 12, 13, 13, 14, 14, 14, 16, 16, 16, 17, 20, 21, 21, 21, 22, 24, 24, 27, 27, 30, 30, 31, 32, 33, 35, 36, 37, 37,
▶ 37, 37, 37, 37, 38, 39, 40, 40, 40, 40, 40, 41, 41, 41, 41, 41, 45, 45, 46, 46, 48, 51, 51, 51, 52, 53, 53, 53, 53, 56, 56, 57, 57, 57, 58, 59, 59, 60, 60, 61,
  63, 63, 63, 64, 65, 65, 65, 68, 72, 73, 74, 74, 75, 76, 76, 77, 78, 79, 80, 81, 81, 82, 83, 83, 83, …]
```

Insertion Sort: (20.69 milisegundos) [Codigo en Github]

```js
function generarNumerosAleatorios() {
  const numerosAleatorios = [];
  for (let i = 0; i < 10000; i++) {
    const numeroAleatorio = Math.floor(Math.random() * 10001);
    numerosAleatorios.push(numeroAleatorio);
  }
  return numerosAleatorios;
}

function insertionSort(arr) {
  let n = arr.length;
  for (let i = 1; i < n; i++) {
    let key = arr[i];
    let j = i - 1;
    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
  return arr;
}

function medirTiempoDeEjecucion(func, arr) {
  const t0 = performance.now();
  func(arr);
  const t1 = performance.now();
  return t1 - t0;
}

const arrayNumeros = generarNumerosAleatorios();
console.log("Antes de ordenar:", arrayNumeros);

const tiempoEjecucion = medirTiempoDeEjecucion(insertionSort, arrayNumeros);
console.log(
  `Tiempo de ejecución del Insertion Sort: ${tiempoEjecucion} milisegundos`
);

const arrayOrdenado = insertionSort(arrayNumeros);
console.log("Después de ordenar:", arrayOrdenado);
```

```
Antes de ordenar:                                                    Insertionsort.js:33
  (10000) [4486, 107, 1327, 9680, 6724, 2930, 4746, 5724, 9597, 9095, 7779, 9562, 2883, 3608, 22, 4442, 4785, 3795, 6192, 335, 1321, 2098, 7150, 2863, 33
▶ 9, 3275, 2277, 8087, 85, 4562, 6162, 3773, 6156, 7162, 1385, 56, 1862, 5035, 1194, 9088, 3751, 2103, 6895, 78, 1957, 5403, 1184, 9718, 774, 6659, 7352,
  7569, 1940, 1081, 4191, 6856, 8320, 1834, 2155, 8380, 339, 6657, 8823, 7293, 2934, 5296, 1401, 5510, 9784, 8468, 3052, 9235, 2175, 9189, 1303, 2772, 98
  14, 3544, 2471, 5145, 5467, 6647, 9552, 4336, 9934, 3667, 526, 1562, 4080, 6415, 1583, 6480, 3050, 3341, 8805, 4890, 8436, 6348, 6280, 6171, …]

Tiempo de ejecución del Insertion Sort: 20.69999998807907 milisegundos   Insertionsort.js:37

Después de ordenar:                                                  Insertionsort.js:43
  (10000) [0, 4, 4, 5, 6, 8, 8, 9, 9, 10, 10, 13, 14, 14, 15, 15, 17, 18, 19, 19, 22, 23, 24, 24, 28, 28, 30, 33, 34, 37, 39, 39, 39, 40, 42, 42, 43, 43,
▶ 44, 44, 45, 45, 46, 46, 47, 48, 49, 52, 55, 56, 57, 57, 60, 62, 63, 63, 64, 65, 65, 66, 69, 70, 71, 71, 72, 72, 73, 73, 73, 74, 75, 76, 76, 78, 78, 78,
  81, 81, 83, 85, 86, 86, 87, 89, 90, 93, 95, 96, 96, 96, 98, 99, 101, 102, 103, 105, 105, 105, 106, 106, …]
```

Merge Sort (9.69 milisegundos) [Codigo en Github]

```javascript
function generarNumerosAleatorios() {
  const numerosAleatorios = [];
  for (let i = 0; i < 10000; i++) {
    const numeroAleatorio = Math.floor(Math.random() * 10001);
    numerosAleatorios.push(numeroAleatorio);
  }
  return numerosAleatorios;
}

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

  const mid = Math.floor(arr.length / 2);
  const left = arr.slice(0, mid);
  const right = arr.slice(mid);

  return merge(mergeSort(left), mergeSort(right));
}

function merge(left, right) {
  let result = [];
  let leftIndex = 0;
  let rightIndex = 0;

  while (leftIndex < left.length && rightIndex < right.length) {
    if (left[leftIndex] < right[rightIndex]) {
      result.push(left[leftIndex]);
      leftIndex++;
    } else {
      result.push(right[rightIndex]);
      rightIndex++;
    }
  }

  return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
}

function medirTiempoDeEjecucion(func, arr) {
  const t0 = performance.now();
  const resultado = func(arr);
  const t1 = performance.now();
  return { tiempo: t1 - t0, resultado };
}

const arrayNumeros = generarNumerosAleatorios();
console.log("Antes de ordenar:", arrayNumeros);

const { tiempo: tiempoEjecucion, resultado: arrayOrdenado } =
  medirTiempoDeEjecucion(mergeSort, arrayNumeros);
console.log(
  `Tiempo de ejecución del Merge Sort: ${tiempoEjecucion} milisegundos`
);

console.log("Después de ordenar:", arrayOrdenado);
```

```
Antes de ordenar:                                                    Mergesort.js:49
  (10000) [5463, 1828, 5102, 8099, 520, 6485, 2963, 27, 4361, 9144, 3056, 8917, 5738, 9231, 2804, 1220, 2767, 5904, 6306, 9149, 923, 9060, 3367, 9701, 81
▶ 59, 1974, 8627, 1341, 6487, 5730, 1949, 7596, 1940, 5466, 5901, 4238, 5505, 2189, 1656, 2691, 391, 9661, 6347, 1999, 6641, 9862, 6119, 3564, 2078, 684
  3, 1205, 6072, 844, 2632, 2013, 7533, 8411, 9186, 7164, 3895, 1222, 5212, 9548, 6382, 7043, 9361, 6565, 153, 8008, 3158, 2103, 333, 735, 2011, 6930, 70
  82, 1466, 9110, 9314, 8708, 1041, 5328, 4250, 6287, 3793, 6659, 6051, 5442, 4682, 4385, 2207, 1451, 4587, 6751, 3205, 4225, 4259, 7683, 4997, 2712, …]

Tiempo de ejecución del Merge Sort: 9.699999988079071 milisegundos   Mergesort.js:54

Después de ordenar:                                                  Mergesort.js:60
  (10000) [1, 2, 3, 4, 7, 9, 9, 10, 10, 11, 16, 17, 17, 18, 19, 22, 23, 23, 23, 24, 25, 26, 27, 27, 28, 30, 30, 32, 33, 34, 37, 38, 39, 40, 42, 43, 43, 4
▶ 4, 44, 44, 46, 47, 50, 50, 50, 51, 51, 54, 55, 55, 56, 57, 60, 61, 63, 63, 63, 65, 67, 67, 69, 69, 70, 72, 72, 74, 75, 76, 77, 77, 77, 78, 81, 81, 82,
  83, 83, 83, 85, 85, 87, 88, 89, 89, 89, 90, 91, 91, 91, 92, 92, 93, 95, 95, 95, 95, 95, 96, 96, 96, …]
```

Quick Sort (8.29 milisegundos) [Codigo en Github]

```js
function generarNumerosAleatorios() {
  const numerosAleatorios = [];
  for (let i = 0; i < 10000; i++) {
    const numeroAleatorio = Math.floor(Math.random() * 10001);
    numerosAleatorios.push(numeroAleatorio);
  }
  return numerosAleatorios;
}

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

  const pivot = arr[Math.floor(arr.length / 2)];
  const left = [];
  const right = [];
  const equal = [];

  for (let num of arr) {
    if (num < pivot) {
      left.push(num);
    } else if (num > pivot) {
      right.push(num);
    } else {
      equal.push(num);
    }
  }

  return quickSort(left).concat(equal).concat(quickSort(right));
}

function medirTiempoDeEjecucion(func, arr) {
  const t0 = performance.now();
  const resultado = func(arr);
  const t1 = performance.now();
  return { tiempo: t1 - t0, resultado };
}

const arrayNumeros = generarNumerosAleatorios();
console.log("Antes de ordenar:", arrayNumeros);

const { tiempo: tiempoEjecucion, resultado: arrayOrdenado } =
  medirTiempoDeEjecucion(quickSort, arrayNumeros);
console.log(
  `Tiempo de ejecución del Quick Sort: ${tiempoEjecucion} milisegundos`
);

console.log("Después de ordenar:", arrayOrdenado);
```

```
Antes de ordenar:                                                    Quicksort.js:42
  (10000) [1629, 8395, 5730, 7494, 600, 2238, 1829, 9577, 8624, 6806, 5495, 1195, 8349, 5070, 6944, 8847, 4963, 6265, 5893, 9410, 8351, 2867, 8584, 5559,
▶ 503, 7481, 2004, 4712, 9036, 7842, 8879, 7053, 5217, 7571, 4011, 4784, 3021, 8905, 5350, 7831, 1328, 9184, 4317, 547, 7860, 3122, 16, 2055, 7061, 9893,
  6985, 9586, 2212, 6744, 8829, 7330, 9190, 4986, 2996, 3058, 2539, 9805, 911, 5049, 7147, 2603, 6731, 462, 9326, 9854, 7568, 6824, 681, 708, 3615, 6932,
  1687, 8624, 9852, 4941, 8656, 7839, 8510, 2042, 7185, 3065, 5221, 9121, 5014, 4382, 5924, 496, 957, 8882, 7078, 4813, 6683, 849, 8751, 6701, …]

Tiempo de ejecución del Quick Sort: 8.299999982118607 milisegundos   Quicksort.js:47

Después de ordenar:                                                  Quicksort.js:53
  (10000) [0, 1, 1, 2, 2, 4, 4, 5, 6, 7, 7, 9, 9, 10, 10, 10, 11, 12, 13, 14, 14, 16, 16, 16, 17, 20, 21, 23, 24, 28, 28, 29, 30, 32, 32, 34, 37, 38, 41,
▶ 42, 42, 43, 44, 44, 45, 45, 46, 48, 50, 50, 50, 53, 56, 57, 58, 59, 60, 61, 61, 62, 62, 62, 64, 68, 68, 73, 74, 74, 74, 75, 76, 76, 77, 77, 79, 79,
  80, 82, 86, 86, 87, 87, 88, 90, 90, 90, 94, 95, 96, 97, 97, 98, 98, 100, 101, 103, 105, 106, 108, …]
```

Conclusión: Quck Sort toma menos tiempo que cualquiera de los otros métodos de ordenamiento.

Bubble Sort (341.5 milisegundos)
Selection Sort (40.59) milisegundos
Insertion Sort: (20.69 milisegundos)
Merge Sort (9.69 milisegundos)
Quick Sort (8.29 milisegundos)