PERBANDINGAN ALGORITMA SORTING DENGAN MENGGUNAKAN BAHASA PEMOGRAMAN JAVASCRIPT DALAM PENGGUNAAN WAKTU KOMPUTASI DAN PENGGUNAAN MEMORI

Surya Wijaya [1], Fauziah [2*], Trinugi Wira Harjanti [3]
Prodi S2 MTI, Fakultas Teknologi Komunikasi dan Informatika, Universitas Nasional [1, 2]
Prodi Informatika, Sekolah Tinggi Informatika, NIIT [3]
fauziah@civitas.unas.ac.id [2]

Submitted June 14, 2023; Revised March 27, 2024; Accepted March 28, 2024

Abstrak

Data yang berurut atau tersusun secara rapih sangat diperlukan karena data yang telah diurutkan memudahkan untuk dibaca, diperiksa, dan diperbaiki jika terdapat data yang salah. Dalam teknologi informasi ada banyak algoritma untuk pengurutan data antara lain *merge-sort, insertion-sort, heap-sort, quick-sort, selection-sort, shell-sort* dan sebagainya. Pada pengujian ini menggunakan tujuh algoritma pengurutan (*Algoritma Merge-Sort, Insertion-Sort, Bubble-Sort, Heap-Sort, Quick-Sort, Selection-Sort dan Shell-Sort*) dengan pengujian data acak dengan skenario pengujian 100, 150, 200 data dan menggunakan bahasa pemograman javascript. Hasilnya penelitian, bahwa *insertion sort* adalah algoritma paling efisien dalam waktu karena waktu rata-rata yang dibutuhkan 0,613 detik. *Quick sort* adalah algoritma paling efisien dalam memori karena memori rata-rata yang dibutuhkan 31072 Kb.
Kata Kunci : algoritma, sorting, waktu, memori.

Abstract

Data that is sequential or neatly arranged data is highly necessary because sorted data makes it easier to read, inspect, and correct any erroneous data. In information technology, there are many algorithms for data sorting, including merge sort, insertion sort, heap sort, quick sort, selection sort, shell sort, and so on. In this test, seven sorting algorithms (Merge Sort, Insertion Sort, Bubble Sort, Heap Sort, Quick Sort, Selection Sort, and Shell Sort) were used with random data testing scenarios of 100, 150, and 200 data points, using the JavaScript programming language. The research findings indicate that insertion sort is the most efficient algorithm in terms of time, with an average time of 0.613 seconds. Quick sort is the most efficient algorithm in terms of memory, with an average memory usage of 31072 Kb.
Keywords : *algorithm, sorting, timing, memory.*

1. PENDAHULUAN

Kemajuan Teknologi Informasi memiliki dampak bagi perubahan dalam kehidupan [1], seperti perubahan penyimpanan data, awalnya data dokumen disimpan dalam lemari arsip, saat ini data disimpan dalam komputer bahkan *cloud computing* [2] dalam bentuk data digital. Seiring waktu data yang disimpan dalam komputer makin banyak, data yang tidak tersusun secara rapih akan sulit untuk dicari dan sulit diperiksa jika terjadi kesalahan. Proses pengurutan data baik secara menaik (*ascending*) maupun menurun (*descending*) [3] dapat menggunakan algoritma pengurutan data antara lain algoritma *shell-sort*, *selection-sort*, *merge-sort*, *heap-sort*, *bubble-sort*, *insertion-sort*, *quick-sort*, dan sebagainya.

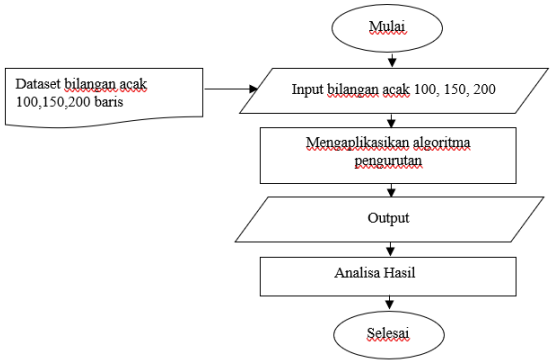
Banyak penelitian yang telah dilakukan untuk pengujian setiap algoritma yaitu Peneliti [4] mengoptimasi algoritma *shell sort* dalam mengurutkan data angka dan huruf. Peneliti [5] menganalisa kompleksitas ruang dan waktu terhadap laju algoritma *merge-sort, insertion-sort, heap-sort* dengan menggunakan bahasa pemograman java. Peneliti [6] menganalisa

perbandingan algoritma *quick-sort, shell-sort,* dan *bubble-sort* dalam mengurutkan deret angka acak menggunakan bahasa java. Peneliti [7] menganalisa algoritma *merge-sort, quick-sort,* dan *bubble-sort* dalam mengurutkan kombinasi huruf dan angka. Peneliti [8] menganalisa perbandingan algoritma *merge-sort* dan *quick-sort* dalam mengurutkan data terhadap waktu dan jumlah langkah. Peneliti [9] menganalisa algoritma *merge-sort* dan *insertion-sort* menggunakan Bahasa C++. Peneliti [10] mengetahui waktu komputasi algoritma *insertion-sort* dan *bubble-sort* menggunakan Bahasa pemograman .Net C# dan Golang. Peneliti [11] menganalisa perbandingan *shell-sort*, *bubble-sort*, dan *quick-sort* menggunakan Bahasa java. Peneliti [12] membandingkan algoritma *selection-sort* dan *insertion-sort* menggunakan pemograman java. Peneliti [13] membandingkan *bubble-sort*, *selection-sort*, dan *insertion-sort* Bahasa java menggunakan python. Peneliti [14] *selection-sort* pada Bahasa pemograman php. Peneliti [15] membandingkan *bubble-sort* dan *quick-sort* pada Bahasa pemograman *flutter*. Peneliti [16] menganalisa kompleksitas ruang dan waktu terhadap algoritma *sorting* menggunakan pyhton.

Pada penelitian saat ini dilakukan pengujian terhadap algoritma *merge-sort, heap-sort, quick-sort, shell-sort, bubble-sort, selection-sort,* dan *insertion-sort* menggunakan bahasa pemograman javascript. Setiap algoritma diukur berdasarkan waktu dan memori yang diperlukan dalam proses pengurutan data, kemudian dianalisa untuk mengetahui waktu dan memori yang lebih sedikit dalam proses komputasi algoritma tersebut.

2. METODE PENELITIAN

Metode atau tahapan pada penelitian ini dengan membuat dataset sebanyak 100, 150, 200 baris bilangan acak dengan memakai website calculatorsoup.com. Selanjutnya setiap algoritma pengurutan diaplikasikan kedalam program javascript terhadap dataset secara bertahap 100, 150, 200. Dari pengaplikasian program dihasilkan output berupa waktu dan memory dalam proses komputasi. Selanjutnya langkah terakhir hasil output dibuat kedalam tabel agar mudah untuk dianalisa. Gambar 1 merupakan gambaran dari tahapan yang digunakan dalam penelitian.



Gambar 1. Flowchart Tahapan Proses Algoritma Sorting

3. HASIL DAN PEMBAHASAN

Dalam pengujian ini menggunakan bahasa pemograman javascript compiler online yang digunakan https://www.jdoodle.com/execute-nodejs-online. Adapun spesifikasi perangkat laptop yang digunakan yaitu:

1. Windows 11 Pro 64Bit
2. Processor: AMD Ryzen 5 4500U 2.38 GHz,
3. Memory RAM 8 GB
4. SSD 256 GB

Algoritma yang digunakan pada pengujian ini adalah Algoritma *Merge-Sort, Bubble-Sort, Insertion-Sort, Selection-Sort, Shell-Sort, Heap-Sort,* dan *Quick-Sort*. Setiap Algoritma diuji dengan jumlah bilangan acak yang berbeda yaitu 100, 150, 200 baris bilangan kedalam bahasa pemograman javascript. Setiap Algoritma diukur

berdasarkan waktu dan memori saat komputasi. Dihasilkan tabel pengujian untuk dianalisa.

Dataset Bilangan Acak 100 Data

[91, 15, 22, 12, 94, 21, 38, 48, 56, 67, 54, 29, 88, 92, 59, 63, 98, 97, 9, 79, 11, 51, 40, 99, 85, 62, 10, 4, 30, 65, 25, 6, 53, 78, 26, 20, 47, 70, 90, 7, 77, 31, 84, 39, 23, 86, 64, 14, 28, 87, 93, 50, 1, 42, 100, 34, 74, 18, 27, 16, 46, 41, 71, 37, 60, 75, 19, 17, 96, 89, 8, 57, 58, 82, 66, 55, 44, 13, 83, 68, 36, 32, 69, 43, 5, 76, 2, 33, 45, 72, 49, 61, 24, 95, 35, 81, 3, 73, 52, 80]

Data diatas merupakan bilangan acak yang terdiri dari 100 data. Dataset tersebut diuji sebagai test ke-1 ke program javascript pada setiap algoritma.

Dataset Bilangan Acak 150 Data

[142, 84, 101, 1, 149, 57, 83, 12, 88, 16, 54, 78, 93, 130, 67, 109, 11, 117, 33, 41, 39, 103, 27, 85, 87, 7, 120, 5, 80, 131, 138, 113, 35, 135, 6, 66, 56, 148, 146, 98, 17, 69, 18, 92, 64, 74, 61, 140, 104, 50, 42, 91, 76, 89, 21, 79, 13, 111, 95, 37, 71, 119, 30, 141, 8, 24, 36, 46, 129, 4, 51, 133, 59, 126, 143, 75, 102, 68, 99, 63, 55, 62, 14, 15, 136, 115, 23, 22, 107, 45, 127, 110, 70, 97, 29, 32, 125, 81, 121, 19, 65, 96, 112, 86, 10, 26, 144, 100, 44, 58, 147, 43, 25, 94, 31, 52, 2, 106, 20, 137, 116, 38, 72, 124, 132, 145, 108, 9, 48, 90, 73, 150, 82, 114, 3, 105, 49, 60, 28, 139, 34, 123, 118, 53, 128, 134, 40, 77, 122, 47].

Data diatas merupakan bilangan acak yang terdiri dari 150 data. Dataset tersebut diuji sebagai test ke-2 ke program javascript pada setiap algoritma.

Dataset Bilangan Acak 200 Data

[95, 165, 70, 37, 192, 155, 117, 80, 121, 86, 102, 175, 187, 4, 194, 41, 83, 30, 25, 45, 29, 74, 193, 136, 64, 130, 24, 144, 114, 100, 34, 173, 107, 3, 134, 43, 33, 20, 181, 48, 159, 197, 32, 103, 1, 36, 31, 85, 79, 112, 26, 63, 168, 35, 101, 189, 139, 69, 78, 38, 178, 6, 124, 198, 153, 98, 131, 2, 39, 93, 132, 60, 75, 81, 150, 188, 128, 52, 12, 13, 176, 171, 164, 10, 82, 55, 118, 8, 104, 27, 77, 129, 67, 11, 84, 123, 66, 108, 161, 191, 180, 46, 170, 72, 154, 149, 199, 53, 151, 196, 51, 135, 115, 94, 119, 126, 200, 87, 22, 5, 49, 146, 157, 59, 17, 116, 50, 15, 113, 65, 18, 88, 71, 44, 89, 109, 110, 57, 14, 160, 148, 190, 92, 185, 127, 152, 138, 120, 7, 40, 162, 179, 147, 90, 125, 76, 183, 145, 156, 122, 195, 186, 21, 28, 167, 163, 174, 137, 16, 99, 106, 56, 19, 73, 42, 177, 143, 182, 105, 97, 9, 111, 169, 54, 184, 61, 172, 58, 96, 23, 133, 91, 166, 140, 158, 68, 47, 62, 142, 141]

Data diatas merupakan bilangan acak yang terdiri dari 200 data. Dataset tersebut diuji sebagai test ke-3 ke program javascript pada setiap algoritma.

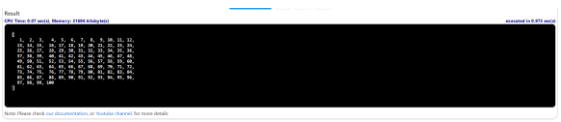
Kode Program dan Hasil Pengujian

Merge Sort

```
function mergeSort(arr,l,r){
    if(l>=r){
        return;
    }
    var m =l+ parseInt((r-l)/2);
    mergeSort(arr,l,m);
    mergeSort(arr,m+1,r);
    merge(arr,l,m,r);
}
```

Gambar 2. Kode Program Algoritma *Merge Sort*

Gambar 2 merupakan kode program algoritma *Merge Sort*. Dataset dimasukan kedalam program dan dijalankan sesuai skenario pengujian yaitu test ke-1 (100 data), test ke-2 (150 data), dan test ke-3 (200 data) dan dibuat tabel hasil pengujian *Merge Sort* (Tabel 1). Hasil dari kode program akan tercatat waktu dan memori yang diperlukan untuk komputasi seperti Gambar 3.



Gambar 3. Hasil Program Test-1 (100 data) Algoritma *Merge Sort*

Tabel 1. Hasil Pengujian *Merge Sort*

Test Ke	Jumlah Data	Waktu s	Memori kilobyte (s)
1	100	0,973	31696
2	150	0,579	31884
3	200	1,105	31212
Minimum		0,579	31212
Maksimum		1,105	31884
Rata-rata		0,886	31597,33

Dari tabel 1 dilihat berdasarkan waktu bahwa kasus terbaik (*best-case*) 0,579 detik, kasus terburuk (*worst-case*) 1,105 detik, kasus rerata (*average-case*) 0,886 detik. Berdasarkan memori bahwa kasus terbaik (*best-case*) 31212 Kb, kasus terburuk (*worst-case*) 31884 Kb, kasus rerata (*average-case*) 31597,33 Kb.

Bubble Sort

```
function bubbleSort(arr){
  var i, j;
  var len = arr.length;
  var isSwapped = false;
  for(i =0; i < len; i++){
    isSwapped = false;
    for(j = 0; j < len; j++){
        if(arr[j] > arr[j + 1]){
          var temp = arr[j]
          arr[j] = arr[j+1];
          arr[j+1] = temp;
          isSwapped = true;
        }
    }
    if(!isSwapped){
      break;
    }
  }
}

```

Gambar 4. Kode Program Algoritma *Bubble Sort*

Gambar 4 merupakan kode program algoritma *bubble sort*. Dataset dimasukan kedalam program dan dijalankan sesuai skenario pengujian yaitu test ke-1 (100 data), test ke-2 (150 data), dan test ke-3 (200 data) dan dibuat tabel hasil pengujian *Bubble Sort* (Tabel 2). Hasil dari kode program akan terlihat waktu dan memori yang diperlukan untuk komputasi seperti Gambar 5.

Gambar 5. Hasil Program Test-1 (100 data) Algoritma *Bubble Sort*

Tabel 2. Hasil Pengujian *Bubble Sort*

Test ke	Jumlah Data	Waktu s	Memori kilobyte (s)
1	100	0,728	31072
2	150	0,635	34152
3	200	0,620	34460
Minimum		0,62	31072
Maksimum		0,728	34460
Rata-rata		0,661	33228

Dari tabel 2 dilihat berdasarkan waktu bahwa kasus terbaik (*best-case*) 0,620 detik, kasus terburuk (*worst-case*) 0,728 detik, kasus rerata (*average-case*) 0,661 detik. Berdasarkan memori bahwa kasus terbaik (*best-case*) 31072 Kb, kasus terburuk (*worst-case*) 34460 Kb, kasus rerata (*average-case*) 33228 Kb.

Insertion Sort

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

```

Gambar 6. Kode Program Algoritma *Insertion Sort*

Gambar 6 merupakan kode program algoritma *insertion sort*. Dataset dimasukan kedalam program dan dijalankan sesuai skenario pengujian yaitu test ke-1 (100 data), test ke-2 (150 data), dan test ke-3 (200 data) dan dibuat tabel hasil pengujian *Insertion Sort* (Table 3). Hasil dari kode program akan tercatat waktu dan memori yang diperlukan untuk komputasi seperti Gambar 7.

Gambar 7. Hasil Program Test ke-1 (100 Data) Algoritma *Insertion Sort*

Tabel 3. Hasil Pengujian *Insertion Sort*

Test Ke	Jumlah Data	Waktu s	Memori kilobyte (s)
1	100	0,550	31204
2	150	0,600	30992
3	200	0,688	31196
Minimum		0,55	30992
Maksimum		0,688	31204
Rata-rata		0,613	31130,67

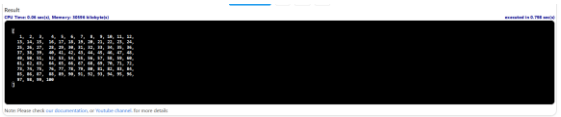
Dari tabel 3 dilihat berdasarkan waktu bahwa kasus terbaik (*best-case*) 0,550 detik, kasus terburuk (*worst-case*) 0,688 detik, kasus rerata (*average-case*) 0,613 detik. Berdasarkan memori bahwa kasus terbaik (*best-case*) 30992 Kb, kasus terburuk (*worst-case*) 31204 Kb, kasus rerata (*average-case*) 31130,67 Kb.

Selection Sort

```
function selectionSort(arr)
{
    let n=arr.length;
    var i, j, min_idx;
    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i + 1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;
         swap(arr,min_idx, i);
    }
}
```

Gambar 8. Kode Program Algoritma *Selection Sort*

Gambar 8 merupakan kode program algoritma *Selection Sort*. Dataset dimasukan kedalam program dan dijalankan sesuai skenario pengujian yaitu test ke-1 (100 data), test ke-2 (150 data), dan test ke-3 (200 data) dan dibuat tabel hasil pengujian *Selection Sort* (Tabel 4). Hasil dari kode program akan tercatat waktu dan memori yang diperlukan untuk komputasi seperti Gambar 9.



Gambar 9. Hasil Program Test ke-1 (100 Data) Algoritma *Selection Sort*

Tabel 4. Hasil Pengujian *Insertion Sort*

Test Ke	Jumlah Data	Waktu s	Memori kilobyte (s)
1	100	0,798	30596
2	150	0,895	31040
3	200	0,645	34356
Minimum		0,645	30596
Maksimum		0,895	34356
Rata-rata		0,779	31997,33

Dari tabel 4 dilihat berdasarkan waktu bahwa kasus terbaik (*best-case*) 0,645 detik, kasus terburuk (*worst-case*) 0,895 detik, kasus rerata (*average-case*) 0,779 detik. Berdasarkan memori bahwa kasus terbaik (*best-case*) 30596 Kb, kasus terburuk (*worst-case*) 34356 Kb, kasus rerata (*average-case*) 31997,33 Kb.

Shell Sort

```
function shellSort(arr)
{
    let n = arr.length;
        for (let gap = Math.floor(n/2); gap > 0; gap = Math.floor(gap/2))
        {
            for (let i = gap; i < n; i += 1)
            {
                let temp = arr[i]; let j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                    arr[j] = arr[j - gap];
                arr[j] = temp;
            }
        }
        return arr;
}
```

Gambar 10. Kode Program Algoritma *Shell Sort*

Gambar 10 merupakan kode program algoritma *Shell Sort*. Dataset dimasukan kedalam program dan dijalankan sesuai skenario pengujian yaitu test ke-1 (100 data), test ke-2 (150 data), dan test ke-3 (200 data) dan dibuat tabel hasil pengujian *Shell Sort* (Tabel 5). Hasil dari kode program akan tercatat waktu dan memori yang diperlukan untuk komputasi seperti Gambar 11.

Gambar 11. Hasil Program Test ke-1 (100 Data) Algoritma *Shell Sort*

Tabel 5. Hasil Pengujian *Shell Sort*

Test Ke	Jumlah Data	Waktu s	Memori kilobyte (s)
1	100	0,589	31148
2	150	0,692	31376
3	200	0,648	31320
	Minimum	0,589	31148
	Maksimum	0,692	31376
	Rata-rata	0,643	31281,33

Dari tabel 5 dilihat berdasarkan waktu bahwa kasus terbaik (*best-case*) 0,589 detik, kasus terburuk (*worst-case*) 0,692 detik, kasus rerata (*average-case*) 0,643 detik. Berdasarkan memori bahwa kasus terbaik (*best-case*) 31148 Kb, kasus terburuk (*worst-case*) 31376 Kb, kasus rerata (*average-case*) 31281,33 Kb.

Quick Sort

```
18  function quickSort(arr, low, high) {
19      if (low < high) {
20          let pi = partition(arr, low, high);
21          quickSort(arr, low, pi - 1);
22          quickSort(arr, pi + 1, high);
23      }
24  }
25
```

Gambar 12. Kode Program Algoritma *Quick Sort*

Gambar 12 merupakan kode program algoritma *Quick Sort*. Dataset dimasukan kedalam program dan dijalankan sesuai skenario pengujian yaitu test ke-1 (100 data), test ke-2 (150 data), dan test ke-3 (200 data) dan dibuat tabel hasil pengujian *Quick Sort* (Tabel 6). Hasil dari kode program akan tercatat waktu dan memori yang diperlukan untuk komputasi seperti Gambar 13.

Gambar 13. Hasil Program Test ke-1 (100 Data) Algoritma *Quick Sort*

Tabel 6. Hasil Pengujian *Shell Sort*

Test Ke	Jumlah Data	Waktu s	Memori kilobyte (s)
1	100	1,096	31160
2	150	0,900	31056
3	200	0,594	31000
Minimum		0,594	31000
Maksimum		1,096	31160
Rata-rata		0,863	31072

Dari tabel 6 dilihat berdasarkan waktu bahwa kasus terbaik (*best-case*) 0,594 detik, kasus terburuk (*worst-case*) 1,096 detik, kasus rerata (*average-case*) 0,863 detik. Berdasarkan memori bahwa kasus terbaik (*best-case*) 31000 Kb, kasus terburuk (*worst-case*) 31160 Kb, kasus rerata (*average-case*) 31072 Kb.

Heap Sort

```
1   function heapSort( arr)
2       {
3           var N = arr.length;
4          for (var i = Math.floor(N / 2) - 1; i >= 0; i--)
5              heapify(arr, N, i);
6          for (var i = N - 1; i > 0; i--) {
7          var temp = arr[0];
8              arr[0] = arr[i];
9              arr[i] = temp;
10             heapify(arr, i, 0);
11         }
12      }
```

Gambar 14. Kode Program Algoritma Heap *Sort*

Gambar 14 merupakan kode program algoritma *Heap Sort*. Dataset dimasukan kedalam program dan dijalankan sesuai skenario pengujian yaitu test ke-1 (100 data), test ke-2 (150 data), dan test ke-3 (200 data) dan dibuat tabel hasil pengujian *Heap Sort* (Tabel 7). Hasil dari kode program akan tercatat waktu dan memori yang diperlukan untuk komputasi seperti Gambar 15.

Gambar 15. Hasil Program Test ke-1 (100 data) Algoritma *Heap Sort*

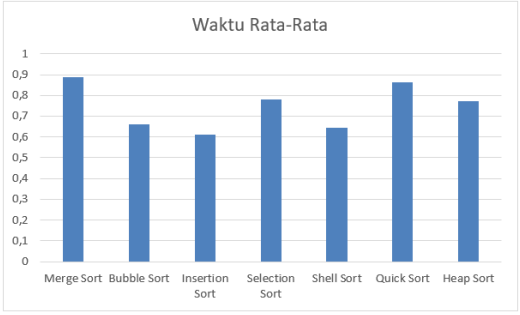
Tabel 6. Hasil Pengujian *Heap Sort*

Test Ke	Jumlah Data	Waktu s	Memori kilobyte (s)
1	100	0,770	31472
2	150	0,776	31304
3	200	0,776	31732
Minimum		0,77	31304
Maksimum		0,776	31732
Rata-rata		0,774	31502,67

Dari tabel 7 dilihat berdasarkan waktu bahwa kasus terbaik (*best-case*) 0,770 detik, kasus terburuk (*worst-case*) 0,776 detik, kasus rerata (*average-case*) 0,774 detik. Berdasarkan memori bahwa kasus terbaik (*best-case*) 31304 Kb, kasus terburuk (*worst-case*) 31732 Kb, kasus rerata (*average-case*) 31502,67 Kb.

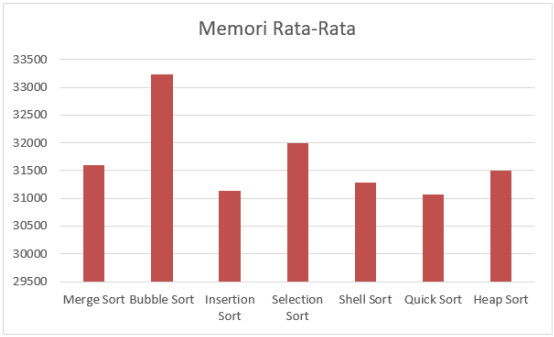
Algoritma	Waktu Rata-rata	Memori Rata-rata
Merge Sort	**0,886**	31597,33
Bubble Sort	0,661	**33228**
Insertion Sort	**0,613**	31130,67
Selection Sort	0,779	31997,33
Shell Sort	0,643	31281,33
Quick Sort	0,863	**31072**
Heap Sort	0,774	31502,67
Minimum	**0,613**	**31072**
Maksimum	**0,886**	**33228**

Dari tabel 8 dilihat berdasarkan waktu rerata paling sedikit adalah algoritma *Insertion Sort* yaitu 0,613 detik. Berdasarkan memori rerata paling sedikit adalah algoritma *Quick Sort* yaitu 31072 Kb.



Gambar 16. Waktu Rerata Setiap Algoritma

Gambar 16 merupakan grafik perbandingan hasil pengujian berdasarkan waktu rerata komputasi dari setiap algoritma. Dari gambar 16 dapat dilihat bahwa algoritma *insertion sort* yang paling efesien waktu.



Gambar 17. Memori Rerata Setiap Algoritma

Gambar 17 merupakan grafik perbandingan hasil pengujian berdasarkan memori rerata komputasi dari setiap algoritma. Dari gambar 17 dapat dilihat bahwa algoritma *quick sort* yang paling efesien memori.

4. SIMPULAN

Hasil pengujian yang telah dilakukan meggunakan tujuh algoritma *sorting* menggunakan Bahasa pemograman javascript menghasilkan waktu dan penggunaan memori. Rekomendasi dari penelitian ini menggunakan tujuh algoritma yaitu Algoritma *Merge-Sort*, *Bubble-Sort*, *Insertion-Sort*, *Selection-Sort*, *Shell-Sort*, *Heap-Sort*, dan *Quick-Sort* dengan menggunakan bilangan acak dari calculatorsoup.com (skenario pengujian 100, 150, 200 baris bilangan acak) menghasilkan penelitian, bahwa algoritma *insertion sort* membutuhkan waktu yang lebih efisien dengan waktu rata-rata 0,613 detik, *quick sort* membutuhkan memori yang lebih efisien dengan memori rata-rata yang 31072 Kb.

DAFTAR PUSTAKA

[1] D. Setiawan, "Dampak Perkembangan Teknologi Informasi dan Komunikasi Terhadap Budaya Impact of Information Technology Development and Communication on Culture," *Simbolika*, vol. 4, no. 1,

pp. 62–72, 2018.

[2] S. Kurniawan, W. Wiranata, N. Ma'muriyah, and V. Vannesse Ting, "Pemanfaatan Komputasi Awan (Cloud Computing) Pada Bidang Pendidikan," *Journal of Information System and Technology*, vol. 04, no. 02, pp. 403–405, 2023, [Online]. Available: https://doi.org/10.24123/saintek.v1i2.28

[3] E. Retnoningsih, "Algoritma Pengurutan Data (Sorting) Dengan Metode Insertion Sort dan Selection Sort," *Information Management for Educators and Professionals*, vol. 3, no. 1, 2018.

[4] H. S. Tambunan, Sumarno, I. Gunawan, and E. Irawan, "Optimasi Algoritma Shell Sort Dalam Pengurutan Data Huruf Dan Angka," *Junal Sistem Informasi Ilmu Komputer Prima*, vol. 2, no. 1, 2018.

[5] R. R Basir, "Analisis Kompleksitas Ruang Dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort Dan Merge Dengan Pemrograman Java," *STRING (Satuan Tulisan Riset dan Inovasi Teknologi*, Vol. 5 No. 2 2020.

[6] N. Sari, W. A. Gunawan, P. K. Sari, I. Zikri, and A. Syahputra, "Analisis Algoritma Bubble Sort Secara Ascending Dan Descending Serta Implementasinya Dengan Menggunakan Bahasa Pemrograman Java," *ADI Bisnis Digital Interdisiplin Jurnal*, vol. 3, no. 1, pp. 16–23, 2022, doi: 10.34306/abdi.v3i1.625.

[7] S. Anisya and N. Febrian, "Analisis Perbandingan Algoritma Bubble Sort, Merge Sort, Dan Quick Sort Dalam Proses Pengurutan Kombinasi Angka Dan Huruf," *Jurnal Pseucode*, vol. 2, no. 2, pp. 75–80, 2015.

[8] Y. Y. P. Rumapea, "Analisis Perbandingan Metode Algoritma Quick Sort Dan Merge Sort Dalam Pengurutan Data Terhadap Jumlah Langkah Dan Waktu," *Jurnal Methodika*, vol. 3, no. 2, 2017.

[9] R. W. Arifin and D. Setiyadi, "Algoritma Metode Pengurutan Bubble Sort dan Quick Dalam Bahasa Pemrograman C++," *Information System for Educators and Professionals*, vol. 4, no. 2, pp. 178–187, 2020.

[10] M. A. Jauhari, D. Hamidin, and M. Rahmatuloh, "Komparasi Stabilitas Eksekusi Kode Bahasa Pemrogrman .Net C# Versi 4.0.3019 Dengan Google Golang Versi 1.4.2 Menggunakan Algoritma Bubble Sort dan Insertion Sort," *Jurnal Teknik Informatika*, vol. 9, no. 1, 2017.

[11] M. Luthfi Zulfa, B. Nurina Sari, and U. Singaperbangsa Karawang Abstract, "Analisis Perbandingan Algoritma Bubble Sort, Shell Sort, dan Quick Sort dalam Mengurutkan Baris Angka Acak menggunakan Bahasa Java," *J. Ilm. Wahana Pendidik.*, vol. 8, no. 13, pp. 237–246, 2022, doi: 10.5281/zenodo.6962346.

[12] Endang Retnoningsih, "Algoritma Pengurutan Data (Sorting) Dengan Metode Insertion Sort dan Selection Sort," *Inf. Manag. Educ. Prof.*, vol. 3, no. 1, pp. 95–106, 2018.

[13] J. Iskandar, H. Suhendar, and B. D. Pamungkas, "Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python," *G-Tech J. Teknol. Terap.*, vol. 8, no. 1, pp. 104–113, 2023, doi: 10.33379/gtech.v8i1.3556.

[14] Y. A. Sandria, M. R. A. Nurhayoto, L. Ramadhani, R. S. Harefa, and A. Syahputra, "Penerapan Algoritma Selection Sort untuk Melakukan

Pengurutan Data dalam Bahasa Pemrograman PHP," *Hello World J. Ilmu Komput.*, vol. 1, no. 4, pp. 190–194, 2022, doi: 10.56211/helloworld.v1i4.187.

[15] D. R. Poetra, "Performa Algoritma Bubble Sort dan Quick Sort pada Framework Flutter dan Dart SDK(Studi Kasus Aplikasi E-Commerce)," *JATISI (Jurnal Tek. Inform. dan Sist. Informasi)*, vol. 9, no. 2, pp. 806–816, 2022, doi: 10.35957/jatisi.v9i2.1886.

[16] Y. Heryanto and T. Wira Harjanti, "Analisis Perbandingan Ruang dan Waktu pada Algoritma Sorting Menggunakan Bahasa Pemrograman Python," *J. Penerapan Sist. Inf. (Komputer Manajemen)*, vol. 4, no. 2, pp. 342–347, 2023.