**ANALISIS EFISIENSI ALGORITMA SORTING**

**Rafly Athalla**

**Gunawan Nur Ahmad**

NIM : 1301194216

1301194292

e-mail: gunawanahmadef@gmail.com

raflyathalla16@gmail.com

### ABSTRAK

**Peran algoritma dalam perangkat lunak atau pemprograman begitu penting, sehingga perlu memahami konsep dasar algoritma. Begitu banyak logika pemprograman yang telah diciptakan, untuk kasus yang umum dan juga khusus. Seiring berkembangnya kemajuan di bidang informatika dan teknologi, tuntutan untuk menemukan metode pemecahan masalah secara lebih tepat, efektif dan kuat menjadi sebuah kebutuhan**. **penelitian ini bertujuan untuk mengetahui efisiensi lima algoritma sorting dengan berbagai macam kompleksitas dan basecase hingga worsecase. Lima algoritma tersebut adalah Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, dan Quick Sort. Pengurutan atau Sorting merupakan suatu proses mengatur susunan data-data menurut syarat tertentu. Meskipun pengurutan ini sepertinya hanya sebuah masalah klasik dalm keinformatikaan, namun perannya tidak dapat dipisahkan terutama dalam pengolahan data**..**Pertama-tama, makalah ini akan membahas definisi dari algoritma, sorting, dan istilah lain yang digunakan di dalam makalah. Setelah itu, baru menginjak pembahasan algoritma sorting. Hal-hal yang dibahas di antaranya adalah konsep algoritma, ide dasar, simulasi, dan kompleksitasnya. Kemudian digambarkan grafik yang menunjukkan seberapa besar peningkatan waktu yang dibutuhkan terhadap panjang tabel yang diurutkan.**

**Kata kunci:** sorting , efisiensi,algoritma , kompleksitas, pengurutan,

### 1. PENDAHULUAN

Dalam matematika dan komputasi, algorita merupakan kumpulan perintah untuk menyelesaikan suatu masalah. Perintah-perintah ini dapat diterjemahkan secara bertahap dari awal hingga akhr.Masalah tersebut dapat berupa apa saja, dengan catatan untuk setiap maslaah, ada kriteria kondisi awal yang harus dipenuhi sebelum menjalankan algoritma. Algoritma akan dapat selalu berakhir untuk semua kondisi awal yang memenuhi kriteria, dalam hal ini berbeda dengan heuristic. Algoritma sering mempunyai langkah pengulangan(iterasi) atau memerlukan keputusan (logika boolean dan perbandingan) sampai tugasnya selesei.

**1.2.Rumusan Masalah**

Dari latar belakang diatas Adapun permasalahan kami adalah sebagai berikut :

1.2.1. Apa pengertian algoritma sorting?

1.2.2. Apa saja bagian-bagian algoritma sorting?

1.2.3. Apa fungsi bagian-bagian algoritma sorting?

**1.3. Tujuan**

Dari rumusan masalah diatas, Adapun tujuan kami adalah sebagai berikut:

1.3.1. Untuk mengetahui pengertian algoritma sorting?

1.3.2. Untuk mengetahui bagian algoritma sorting?

1.3.3. Untuk mengetahui fungsi algoritma sorting?

### 2. METODE SORTING

**2.1. Bubble Sort**

**2.1.1. Konsep Bubble Sort**

Bubble Sort merupakan cara pengurutan yang

sederhana. Konsep dari ide dasarnya adalah seperti

“gelembung air” untuk elemen struktur data yang

semestinya berada pada posisi awal. Cara kerjanya

adalah dengan berulang-ulang melakukan traversal

(proses looping) terhadap elemen-elemen struktur data

yang belum diurutkan. Di dalam traversal tersebut,

nilai dari dua elemen struktur data dibandingkan. Jika

ternyata urutannya tidak sesuai dengan “pesanan”,

maka dilakukan pertukaran (swap). Algoritma sorting

ini disebut juga dengan comparison sort dikarenakan

hanya mengandalkan perbandingan nilai elemen untuk

mengoperasikan elemennya.

**2.1.2. Simulasi Bubble Sort**

Berikut ini adalah contoh simulasi algoritma sorting

terhadap larik (array) dengan panjang 5 elemen.

Misalnya untuk larik dengan elemen :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 3 | 4 | 1 | 6 |

akan diurutkan dengan algoritma bubble sort secara terurut menaik (ascending).

Proses pembandingan di dalam setiap looping digambarkan dengan warna kuning. Jika ternyata diperlukan pertukaran nilai, maka akan ditulis “tukar” di bawah elemen larik yang dibandingkan tersebut. Hasil penukaran digambarkan di tabel simulasi berikutnya. Untuk membedakan dengan elemen tabel yang lain, bagian tabel yang telah diurutkan digambarkan dengan warna biru muda.

Perhatikan bahwa nilai ‘1’ yang merupakan nilai terkecil di dalam larik seolah-olah mengapung mengikuti proses looping. Pengapungan ini terus berlangsung hingga menemukan elemen tabel yang nilainya lebih kecil lagi dari elemen tabel tersebut. Inilah yang dimaksud dengan “efek gelembung” di dalam bubble sort.

*Looping* Pertama :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 3 | 4 | 1 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 3 | 4 | 1 | 6 |

tukar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 3 | 1 | 4 | 6 |

tukar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 1 | 3 | 4 | 6 |

tukar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 7 | 3 | 4 | 6 |

*Looping* kedua :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 7 | 3 | 4 | 6 |

Tukar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 4 | 6 |

*Looping* ketiga :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 4 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 4 | 6 |

Tukar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 6 |

*Looping* keempat :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 6 |

Tukar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 6 | 7 |

**2.1.3. Algoritma Bubble Sort**

Algoritma bubble sort dapat diringkas sebagai

berikut, jika N adalah panjang elemen struktur data,

dengan elemen-elemennya adalah T1, T2, T3, …, TN-1,

TN, maka:

1. Lakukan traversal untuk membandingkan

dua elemen berdekatan. Traversal ini

dilakukan dari belakang.

1. Jika elemen pada TN-1 > TN , maka lakukan

pertukaran (swap). Jika tidak, lanjutkan ke

proses traversal berikutnya sampai bertemu

dengan bagian struktur data yang telah

diurutkan.

1. Ulangi langkah di atas untuk struktur data

yang tersisa.

Pseudocode dari algoritma ini untuk struktur data larik

(array) one-base (struktur data yang memiliki indeks

dasar satu) dapat ditulis sebagai berikut:

Procedure Bubble Sort

Const { PerformanceObserver,performance } = require(‘perf\_hooks’);

let arr = []

for (let i = 1; I <= 1000000; i++) {

let x = Math.round((Math.random()\*100) +1);

}

Console.log(‘done’)

function bubbleSort(inputArr) {

let n = inputArr.length;

for(let i = 0; I < n ; i++) {

for(let j = 0; j< n; J++) {

if(inputArr[j] > inputArr[j+1]) {

let t = inputArr[j];

inputArr[j] = inputArr[j+1];

inputArr[j+1] = t;

}

}

}

return inputArr;

}

let a = performance.now()

let sort = bubbleSort(arr)

let b = performance.now()

console.log(b-a)

**2.1.4. Kompleksitas Bubble Sort**

Algoritma di dalam *bubble sort* terdiri dari 2 kalang (*loop*) bertingkat. Kalang pertama berlangsung selama *N*-1 kali. Indeks untuk kalang pertama adalah *Pass*. Kemudian kalang tingkat kedua berlangsung dari *N* sampai dengan *Pass*+1.

Dengan demikian, proses *compare* yang terjadi sebanyak:

T(n) ini merupakan kompleksitas untuk kasus terbaik ataupun terburuk. Hanya saja pada kasus terbaik, yaitu pada kasus jika struktur data telah terurut sesuai perintah, proses Pass terjadi tanpa adanya assignment.

Hal ini jelas menunjukkan bahwa algoritma akan

berjalan lebih cepat. Hanya saja tidak terlalu signifikan.

Algoritma bubble sort, jika dikembangkan, kompleksitasnya mencapai Ω(n) .Hal ini hanya terjadi jika memanfaatkan tipe data boolean untuk menyatakan bahwa proses penukaran tak perlu dilakukan lagi. Dengan demikian, kompleksitas Ω(n) tidak berlaku bagi pseudocode di atas. Untuk itu perlu dilakukan perubahan. Perubahan yang terjadi pada

pseudocode-nya dapat diperhatikan sebagai berikut

Procedure Bubble Sort

Const { PerformanceObserver,performance } = require(‘perf\_hooks’);

let arr = []

for (let i = 1; I <= 1000000; i++) {

let x = Math.round((Math.random()\*100) +1);

}

Console.log(‘done’)

function bubbleSort(inputArr) {

let n = inputArr.length;

for(let i = 0; I < n ; i++) {

for(let j = 0; j< n; J++) {

if(inputArr[j] > inputArr[j+1]) {

let t = inputArr[j];

inputArr[j] = inputArr[j+1];

inputArr[j+1] = t;

}

}

}

return inputArr;

}

let a = performance.now()

let sort = bubbleSort(arr)

let b = performance.now()

console.log(b-a)

Dengan demikian ketika Pass terjadi sekali melalui semua bagian struktur data, kemudian ditemukan bahwa tidak diperlukan lagi proses penukaran elemen, maka proses pass akan berhenti. Bubble sort hanya melakukan N-1 perbandingan di dalam algoritmanya dan menetapkan bahwa struktur data telah terurut.

Dengan demikian,

Sementara nya tetep sama.

Bubble sort tergolong algoritma yang paling tidak efisien di antara algoritma sorting dengan kompleksitas

Berikut ini adalah data analisis empiris efisiensi Bubble Sort.

Berdasarkan tabel, tidak ada perbedaan performansi secara signifikan untuk pengurutan terhadap 100 item atau lebih. Namun, bubble sort tidak disarankan untuk pengurutan yang terus berulang, atau pengurutan yang menangani lebih dari 200 item.

**2.2. Selection Sort**

**2.2.1. Konsep Selection Sort**

Algoritma sorting sederhana yang lain adalah Selection Sort. Ide dasarnya adalah melakukan beberapa kali pass untuk melakukan penyeleksian elemen struktur data. Untuk sorting ascending (menaik), elemen yang paling kecil di antara elemenelemen yang belum urut, disimpan indeksnya, kemudian dilakukan pertukaran nilai elemen dengan indeks yang disimpan tersebut dengan elemen yang paling depan yang belum urut. Sebaliknya, untuk sorting descending (menurun), elemen yang paling 4 besar yang disimpan indeksnya kemudian ditukar.

**2.2.2. Simulasi Selection Sort**

Untuk lebih jelasnya, perhatikanlah simulasi Selection Sort Ascending berikut dengan menggunakan larik

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 33 | 41 | 4 | 16 | 21 |

Dalam satu kali pass, ditentukan elemen yang paling kecil di dalam bagian list yang belum urut. Elemen yang paling kecil ini, diwarnai merah muda. Untuk bagian larik yang telah diurutkan diberi warna biru muda.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 33 | 41 | 4 | 16 | 21 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 41 | 33 | 16 | 21 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 16 | 33 | 41 | 21 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 16 | 21 | 41 | 33 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 16 | 21 | 33 | 41 |

**2.2.3. Algoritma Selection Sort**

Algoritma selection sort dapat dirangkum sebagai berikut:

1. Temukan nilai yang paling minimum(atau sesuaikeinginan) di dalam struktur data.Jika ascending, maka yang harus ditemukan adalah

nilai yang paling minimum. Jika descending,

maka temukan nilai yang paling maksimum.

2.Tukar nilai tersebut dengan nilai pada posisi

pertama di bagian struktur data yang belum

diurutkan.

3.Ulangi langkah di atas untuk bagian struktur data

yang tersisa.

Pseudocode untuk algoritma ascending selectionsort

untuk struktur data larik basis satu dapat ditulis sebagai berikut

procedure selectionSort

Const { PerformanceObserver,performance } = require(‘perf\_hooks’);

let arr = []

for (let i = 1; I <= 1000000; i++) {

let x = Math.round((Math.random()\*100) +1);

}

Console.log(‘done’)

function selectionSort(inputArr) {

    let n = inputArr.length;

    for(let i = 0; i < n; i++) {

        let min = i;

        for(let j = i+1; j < n; j++){

            if(inputArr[j] < inputArr[min]) {

                min=j;

            }

         }

         if (min != i) {

             let tmp = inputArr[i];

             inputArr[i] = inputArr[min];

             inputArr[min] = tmp;

        }

    }

    return inputArr;

}

let a = performance.now()

let sort = selectionSort(arr)

let b = performance.now()

console.log(b-a)

**2.2.4. Kompleksitas Selection Sort**

Algoritma di dalam Selection Sort terdiri dari kalang

bersarang. Dimana kalang tingkat pertama (disebut

pass) berlangsung N-1 kali. Di dalam kalang kedua,

dicari elemen dengan nilai terkecil. Jika didapat,

indeks yang didapat ditimpakan ke variabel min. Lalu

dilakukan proses penukaran. Begitu seterusnya untuk

setiap Pass. Berdasarkan operasi perbandingan

elemennya:

Perhitungan ini mirip dengan perhitungan kompleksitas bubble sort sebelumnya. Namun, melalui uji empiris, didapatkan bahwa Selection Sort lebih efisien daripada Bubble Sort. Grafiknya dapat dilihat berikut ini:

Peningkatan performansi yang diberikan 60% lebih baik daripada bubble sort. Namun, sorting model ini tergolong buruk dan lebih baik dihindari penggunaannya, terutama untuk penanganan tabel dengan 1000 elemen. Karena masih ada algoritma lain yang implementasinya sama mudahnya, namun performansinya jauh lebih baik. Algoritma yang dimaksud adalah insertion sort yang akan dibahas di bawah ini

**2.3. Insertion Sort**

**2.3.1. Konsep Insertion Sort**

Cara kerja insertion sort sebagaimana namanya.

Pertama-tama, dilakukan iterasi, dimana di setiap

iterasi insertion sort memindahkan nilai elemen,

kemudian menyisipkannya berulang-ulang sampai ke

tempat yang tepat. Begitu seterusnya dilakukan. Dari proses iterasi, seperti biasa, terbentuklah bagian yang

telah di-sorting dan bagian yang belum.

**2.3.2. Algoritma Insertion Sort**

Algoritma Insertion Sort dapat dirangkum sebagai

berikut:

1. Simpan nilai Ti kedalam variabel sementara,

dengan i = 1.

1. Bandingkan nilainya dengan elemen

sebelumnya.

1. Jika elemen sebelumnya (Ti-1) lebih besar

nilainya daripada Ti, maka tindih nilai Ti

dengan nilai Ti-1 tersebut. Decrement i

(kurangi nilainya dengan 1).

4. Lakukan terus poin ke-tiga, sampai Ti-1 ≤ Ti.

5. Jika Ti-1 ≤ Ti terpenuhi, tindih nilai di Ti

dengan variabel sementara yang disimpan

sebelumnya.

6. Ulangi langkah dari poin 1 di atas dengan i

di-increment (ditambah satu).

Pseudocode untuk Insertion Sorting secara Ascending

dapat dituliskan sebagai berikut:

const { PerformanceObserver, performance } = require('perf\_hooks');

let arr = []

for(let i = 1; i<= 2000000; i++) {

    let x = Math.round((Math.random()\* 100) + 1);

    arr.push(x)

}

console.log('done')

function insertionSort(inputArr) {

    let n = inputArr.length;

        for (let i = 1; i < n; i++) {

            let current = inputArr[i];

            let j = i-1;

            while ((j > -1) && (current < inputArr[j])) {

                inputArr[j+1] = inputArr[j];

                j--;

            }

            inputArr[j+1] = current;

        }

    return inputArr;

}

let a = performance.now()

let sort = insertionSort(arr)

let b = performance.now()

console.log(b-a)

**2.3.3. Kompleksitas Insertion Sort**

Algoritma Insertion Sort juga terdiri dari 2 kalang bersarang. Dimana terjadi N-1 Pass (dengan N adalah banyak elemen struktur data), dengan masing-masing Pass terjadi i kali operasi perbandingan. i tersebut bernilai 1 untuk Pass pertama, bernilai 2 untuk Pass kedua, begitu seterusnya hingga Pass ke N-1.

Walaupun sama seperti dua algoritma sorting sebelumnya, insertion sort lebih efektif. Perhatikan gambar berikut:

Berdasarkan gambar, Insertion Sort lebih dari dua kali lebih efektif daripada Bubble Sort dan 40% lebih cepat daripada Selection Sort. Insertion Sort lebih baik tidak digunakan untuk menangani struktur data dengan lebih dari 200 elemen.

**2.4. Merge Sort**

**2.4.1. Sekilas Tentang Merge Sort**

Algoritma Merge Sort ditemukan oleh John von Neumann di tahun 1945. Merge Sort termasuk paradigma algoritma divide and conquer (kurang lebih berarti: bagi dan atasi). Hal ini dikarenakan algoritma ini melakukan pembagian struktur data sebelum kemudian dioperasi satu per satu. Intinya, algoritma ini menggunakan dua ide utama sebagai berikut,

1. Sebuah list yang kecil membutuhkan langkah yang lebih sedikit untuk pengurutan daripada sebuah list yang besar.

2. Untuk membentuk sebuah list terurut dari dua buah list terurut membutuhkan langkah yang 6 lebih sedikit daripada membentuk sebuah list terurut dari dua buah list tak terurut. Contoh: hanya diperlukan satu kali traversal untuk masing-masing list jika keduanya sudah terurut.

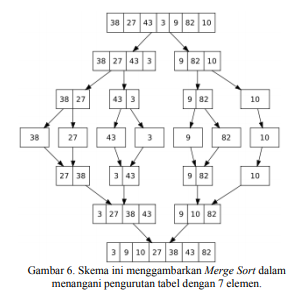
**2.4.2. Algoritma Merge Sort**

Algoritma Merge Sort sederhananya, dapat ditulis berikut:

1. Bagi list yang tak terurut menjadi dua sama panjang atau salah satunya lebih panjang satu elemen.

2. Bagi masing-masing dari 2 sub-list secara rekursif sampai didapatkan list dengan ukuran 1.

3. Gabung 2 sublist kembali menjadi satu list terurut. Untuk lebih jelasnya, perhatikan gambar berikut:



Pseudocode Algoritma Merge Sort dapat ditulis sebagai berikut (Tabel yang digunakan berbasis 0):

const { PerformanceObserver, performance } = require('perf\_hooks');

let arr = []

for(let i = 1; i<= 2000000; i++) {

    let x = Math.round((Math.random()\* 100) + 1);

    arr.push(x)

}

console.log('done')

function merge(left, right) {

    let arr = []

    while (left.length && right.length) {

        if (left[0] < right[0]) {

            arr.push(left.shift())

        } else {

            arr.push(right.shift())

        }

    }

    return [ ...arr, ...left, ...right ]

}

function mergeSort(array) {

    const half = array.length / 2

    if(array.length < 2){

      return array

    }

    const left = array.splice(0, half)

    return merge(mergeSort(left),mergeSort(array))

}

let a = performance.now()

let sort = mergeSort(arr)

let b = performance.now()

console.log(b-a)

**2.4.3. Kompleksitas Merge Sort**

Untuk pengurutan n elemen, merge sort memiliki kompleksitas Kompleksitas ini, juga berlaku untuk kasus terburuk.

Jika running time untuk merge sort terhadap struktur data dengan banyak elemen n adalah maka recurrence-nya berdasarkan definisi dari algoritma.

Algoritma ini jelas lebih efektif daripada algoritma algoritma sebelumnya.

Algoritma ini rekursif, sehingga algoritma ini menjadi pilihan buruk jika ingin dijalankan di mesin komputer dengan memori yang terbatas. 7 Perhatikan gambar! Bandingkan dengan algoritmaalgoritma dengan kompleksitas sebelumnya yang ketika diimplementasikan umumnya membutuhkan waktu di atas 100 detik ketika menangani tabel dengan ribuan elemen.

Perhatikan gambar di bawah ini:

**2.5. Quick Sort**

**2.5.1. Sekilas Tentang Quick Sort**

Quick Sort adalah algoritma sorting yang terkenal

yang dirancang oleh C.A.R. Hoare pada tahun 1960

ketika bekerja untuk perusahaan manufaktur komputer

saintifik kecil, Elliott Brothers. Algoritma ini rekursif,

dan termasuk paradigma algoritma divide and

conquer.

**2.5.2. Algoritma Quick Sort**

Algoritma ini terdiri dari 4 langkah utama:

1. Jika struktur data terdiri dari 1 atau 0 elemen

yang harus diurutkan, kembalikan struktur

data itu apa adanya.

2. Ambil sebuah elemen yang akan digunakan

sebagai pivot point (poin poros). (Biasanya

elemen yang paling kiri.)

3. Bagi struktur data menjadi dua bagian – satu

dengan elemen-elemen yang lebih besar

daripada pivot point, dan yang lainnya dengan

elemen-elemen yang lebih kecil daripada

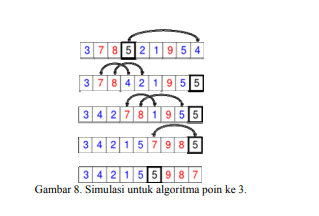
pivot point.

4. Ulangi algoritma secara rekursif terhadap

kedua paruh struktur data.

Perhatikan gambar simulasi berikut. Kotak yang

ditebalkan adalah pivot. Angka yang berwarna merah bernilai ≤ pivot, sementara yang berwarna biru sebaliknya. Simulasi ini menggambarkan proses penukaran yang terjadi supaya terpenuhi kondisi yang dimaksud algoritma no 3:



Versi sederhana pseudocode untuk algoritma ini dapat ditulis berikut:

const { PerformanceObserver, performance } = require('perf\_hooks');

let arr = []

for(let i = 1; i<= 2000000; i++) {

    let x = Math.round((Math.random()\* 100) + 1);

    arr.push(x)

}

console.log('done')

function partition(arr, start, end){

    const pivotValue = arr[end];

    let pivotIndex = start;

    for (let i = start; i < end; i++) {

        if (arr[i] < pivotValue) {

        [arr[i], arr[pivotIndex]] = [arr[pivotIndex], arr[i]];

        pivotIndex++;

        }

    }

    [arr[pivotIndex], arr[end]] = [arr[end], arr[pivotIndex]]

    return pivotIndex;

};

function quickSortIterative(arr) {

    stack = [];

    stack.push(0);

    stack.push(arr.length - 1);

    while(stack[stack.length - 1] >= 0){

        end = stack.pop();

        start = stack.pop();

        pivotIndex = partition(arr, start, end);

        if (pivotIndex - 1 > start){

            stack.push(start);

            stack.push(pivotIndex - 1);

        }

        if (pivotIndex + 1 < end){

            stack.push(pivotIndex + 1);

            stack.push(end);

        }

    }

}

let a = performance.now()

let sort = quickSortIterative(arr)

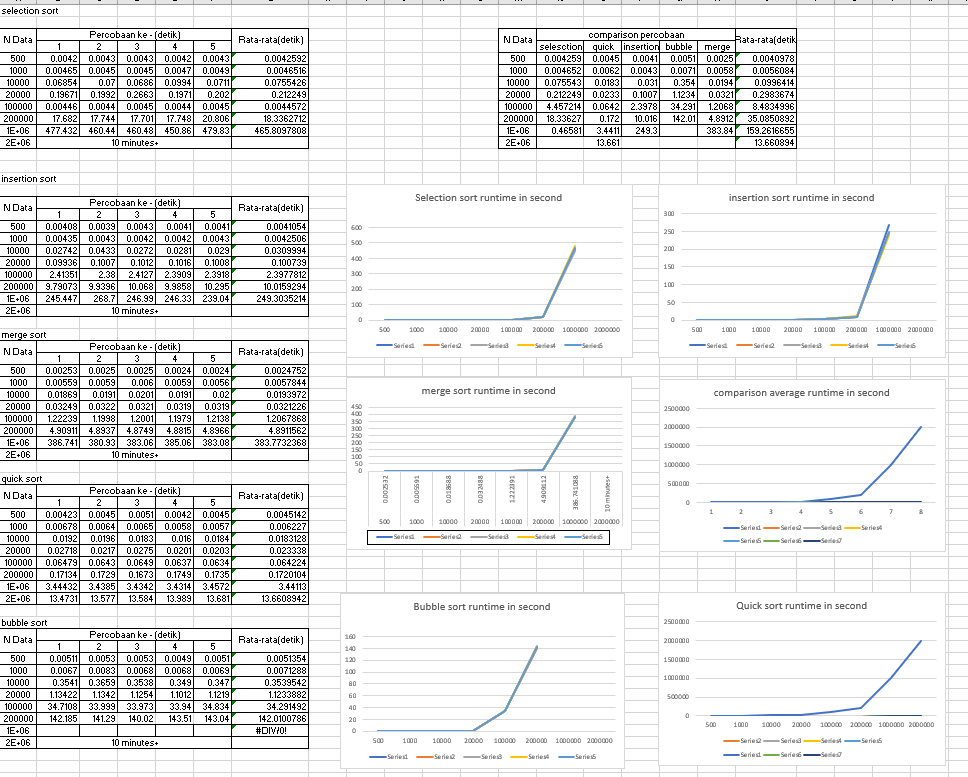
let b = performance.now()

console.log(b-a)

**2.5.3. Kompleksitas Quick Sort**

Efisiensi algoritma ini sangat dipengaruhi elemen mana yang dipilih sebagai pivot point. Kasus terburuk memiliki efisiensi terjadi jika struktur data telah terurut, sementara elemen yang paling kiri dipilih sebagai pivot point. Pemilihan pivot point secara random (acak) disarankan jika data yang akan diurutkan tidak random. Selama pivot point dipilih secara acak, Quick Sort memiliki kompleksitas algoritma Perhatikan gambar!

**3. Hasil eksperimen/pengujian**

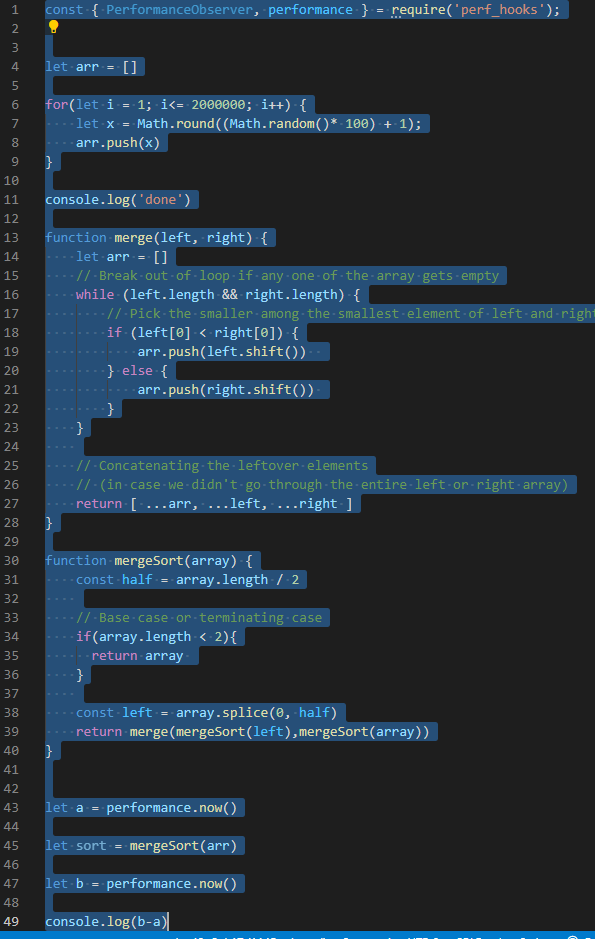
****

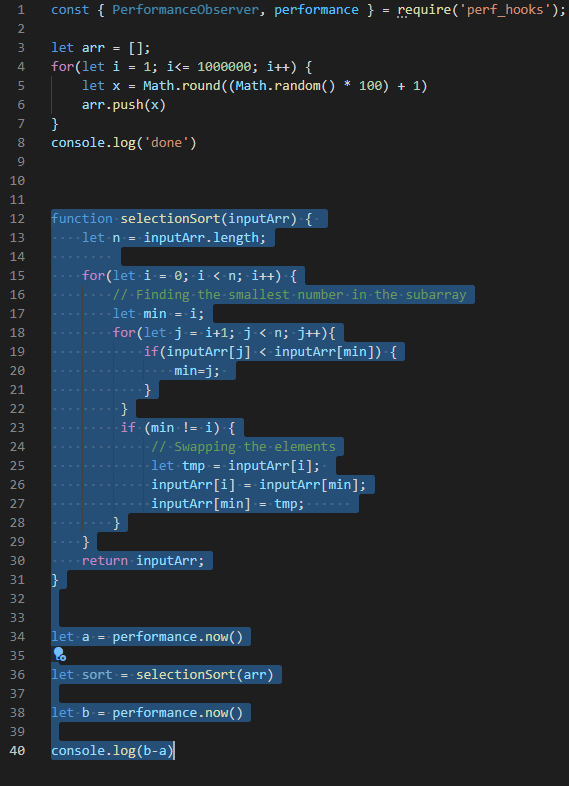
**4. KESIMPULAN**

Algoritma yang mudah dalam hal implementasi adalah Bubble Sort, Selection Sort, dan Insertion Sort. Ketiganya memiliki kompleksitas . Di antara algoritma ini, yang paling effisien adalah Insertion Sort. Algoritma yang lebih efektif adalah Quick Sort dengan kompleksitasnya adalah .

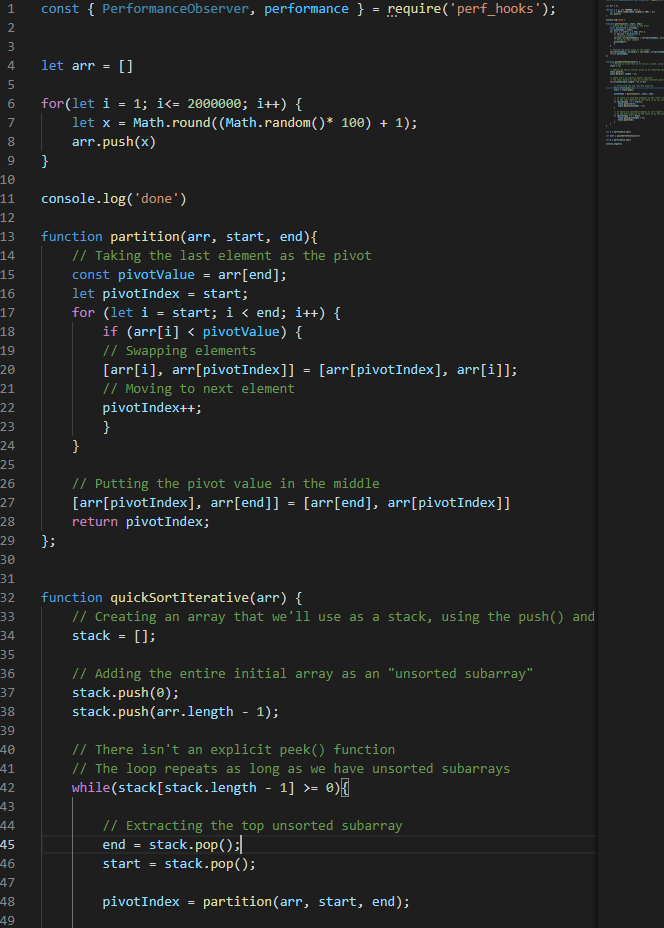
**LAMPIRAN**

****

****

****

****

****

**REFERENSI**

[1] Munir, Rinaldi. 2003. Diktat Kuliah IF2153 Matematika Diskrit. Program Studi Teknik Informatika, Institut Teknologi Bandung. hlm. V1 dan X-1 s.d. X-20

[2] Liem, Inggriani. 2007. Draft Diktat Kuliah Dasar Pemrograman (Bagian Pemrograman Prosedural). Program Studi Teknik Informatika, Institut Teknologi Bandung. hlm. 141-142

[3] GeekForGeeks.(25-07-2020)SortingAlgorithm, <https://www.geeksforgeeks.org/insertion-sort/>

[4] GeekForGeeks.(02-05-2020).SortingAlgorithm, <https://www.geeksforgeeks.org/selection-sort/>

[5] GeekForGeeks.(16-10-2020).SortingAlgorithm, <https://www.geeksforgeeks.org/iterative-merge-sort/>

[6] GeekForGeeks.(16-10-2020).SortingAlgorithm, <https://www.geeksforgeeks.org/iterative-quick-sort/>

[7] GeekForGeeks.(30-09-2020).SortingAlgorithm, https://www.geeksforgeeks.org/bubble-sort/