Analisis Perbandingan Kompleksitas Alternatif Algoritma *Outlier Detection* untuk Distribusi Non-Gauss

Cetta Reswara Parahita - 13521133[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521133@std.stei.itb.ac.id

***Abstract*—Outlier merupakan nilai yang sangat ekstrem perbedaannya, baik lebih tinggi atau lebih rendah, secara relatif terhadap data-data lain yang ada pada suatu grafik atau *dataset.* Perlu dibedakan antara *outlier* dengan nilai data yang baik. Banyak metode yang digunakan sebagai *outlier detection* antara lain adalah Peirce's *criterion*, Chauvenet's *criterion*, Grubbs's *test*, Z-*score*, dan MAD. Perlu dilakukan analisis perbandingan kompleksitas algoritma pada alternatif-alternatif algoritma yang digunakan untuk *outlier detection* sehingga dapat ditentukan metode yang paling cocok digunakan untuk *outlier detection.* Hasil perhitungan kompleksitas algoritma menunjukkan Pierce's *criterion,* Grubb's *test*, dan MAD memiliki kompleksitas $O(N^2)$ sedangkan Chauvenet's *criterion* dan Z-*score* memiliki kompleksitas O(N). Hasil ini menunjukkan Chauvenet's *criterion* dan Z-*score* cocok menjadi pilihan algoritma *outlier detection* apabila diperlukan algoritma yang memiliki kecepatan kerja yang relatif cepat dengan masukan data yang banyak.**

Keywords*—Algoritma, komplekstias, non-gauss, *outlier detection

I. PENDAHULUAN

Penggunaan data dalam jumlah besar dewasa ini menjadi suatu kebutuhan dasar, utamanya dalam sebuah rangkaian penelitian maupun penemuan metode atau algoritma baru yang mendukung suatu inovasi. Analisis *dataset* banyak mendatangkan manfaat dan hasil olahannya juga dapat berguna di berbagai sektor.

Dalam pengolahan data, perlu dibedakan antara sebuah data yang baik (*good values*) dan data yang kurang sesuai dengan keumuman *dataset* (*bad values*). Data yang kurang sesuai dengan keumuman sebuah *dataset* biasa disebut juga sebagai *outlier*.

Outlier merupakan nilai yang sangat ekstrem perbedaannya, baik lebih tinggi atau lebih rendah, secara relatif terhadap data-data lain yang ada pada suatu grafik atau *dataset. Outlier* dianggap sebagai sebuah nilai yang buruk (*bad values*) dari *dataset* asli sehingga perlu dibedakan antara sebuah nilai yang buruk (*bad values*) dari nilai-nilai yang baik (*good values*).

Metode yang paling umum digunakan untuk menentukan *outlier* dari sebuah data adalah distribusi normal, namun metode ini menggunakan asumsi yang sifatnya terlalu kuat. Asumsi yang terlalu kuat ini menyebabkan timbulnya banyak galat atau kondisi kesalahan perhitungan di mana beberapa data yang seharusnya tidak termasuk sebagai *bad values* diakui sebagai *bad values*. Akibatnya, pada penelitian yang lebih lanjut diperlukan algoritma yang lebih teliti dalam deteksi *outliers*. Perlu dibuat algoritma-algoritma lain yang dapat menentukan *outlier* tanpa melakukan *normality assumption*.

Beberapa algoritma lain yang saat ini telah dibuat dan digunakan sebagai *outlier detection* antara lain adalah Peirce's *criterion*, Chauvenet's *criterion*, Grubbs's *test*, Z-*score*, dan MAD. Karena banyaknya algoritma yang dapat dipilih ini, pengguna algoritma harus mengetahui secara lebih spesifik algoritma mana yang sesuai dengan keperluan pengolahan data mereka.

Salah satu indikator yang dapat dijadikan tolak ukur pemilihan algoritma ini adalah perbandingan kompleksitas algoritmanya. Kompleksitas algoritma akan menunjukkan efisiensi algoritma dalam dimensi waktu dan penggunaan ruang. Dengan mengetahuinya akan memberikan kemudahan pengguna algoritma dalam memilih algoritma yang sesuai dengan spesifikasi yang diharapkan.

Oleh karena itu, perlu dilakukan analisis perbandingan kompleksitas algoritma pada alternatif-alternatif algoritma yang digunakan untuk *outlier detection* sehingga dapat ditentukan metode yang paling cocok digunakan untuk *outlier detection.*

II. TEORI DASAR

A. Kompleksitas Algoritma

Algoritma adalah prosedur yang berisi tahapan-tahapan untuk menemukan solusi dari suatu permasalahan dalam waktu yang terbatas. Sebuah algoritma mentransformasikan sebuah masukan menjadi luaran menggunakan sebuah fungsi yang berdependensi pada obyek masukan dalam waktu eksekusi tertentu sesuai dengan jalannya fungsi tersebut. Hal ini dapat dimaknai bahwa sebuah masukan yang ukurannya lebih besar memerlukan waktu eksekusi yang lebih lama dalam menjalankan fungsi tersebut.

Secara umum, dalam menyelesaikan sebuah masalah pasti dapat menghasilkan lebih dari satu algortima. Oleh karena itu, perbandingan tingkat keoptimalan sebuah algoritma sangat diperlukan untuk mengetahui algoritma mana yang lebih efisien

dalam menyelesaikan masalah. Metode perbandingan keoptimalan sebuah algoritma dapat ditinjau berdasarkan dua metode yakni kompleksitas waktu dan kompleksitas ruang.

1) *Kompleksitas Waktu*

Kompleksitas waktu dari sebuah algoritma merupakan hasil perhitungan kuantitatif waktu yang diperlukan sebuah algoritma untuk menjalankan fungsi berdasarkan panjang masukan yang dianggap beragam. Untuk menghitungnya, dibuat asumsi sebuah konstanta waktu c sebagai waktu eksekusi sebuah operasi kemudian dilakukan perhitungan total operasi pada sebuah masukan dengan panjang N. Contoh dari perhitungan kompleksitas waktu adalah sebagai berikut:

```
int a[n];
for(int i = 0;i < n;i++)
  cin >> a[i]


for(int i = 0;i < n;i++)
  for(int j = 0;j < n;j++)
    if(i!=j && a[i]+a[j] == z)
       return true

return false
```

Fig. 1. *Pseudo-code* algoritma pencarian pasangan X dan Y pada fungsi pair(X,Y) yang jumlahnya sama dengan Z (*Sumber: geeksforgeeks.org*)

Pada gambar di atas, terdapat pseudo-code fungsi pair(X,Y) yang akan melakukan pencarian pasangan nilai X dan Y elemen array A yang jumlahnya Z dan mengembalikan nilai kebenaran apakah pasangan ditemukan atau tidak. Diasumsikan bahwa sistem operasi komputer memerlukan waktu konstan c untuk mengeksekusi tiap operasi. Waktu operasi ini bergantung terhadap penemuan pasangan X dan Y yang jumlahnya Z di array A sehingga nilai eksak total waktu eksekusi tidak bisa dipastikan. Namun, dapat dipastikan skenario terburuk dari keberjalanan algoritma ini adalah ketika tidak ditemukan pasangan X dan Y pada array A yang jumlahnya Z karena sistem harus menghitung setiap kemungkinan pasangan untuk mengembalikan nilai kesalahan (false).

Pada kasus terburuk (worst-case scenario) ini, dapat dihitung total waktu eksekusi sebagai berikut:

- Waktu eksekusi operasi masukan: **N*c**
- Waktu eksekusi loop for bagian luar (parameter i): **N kali waktu eksekusi badan fungsinya**
- Waktu eksekusi loop for bagian dalam (parameter j): **N kali waktu eksekusi badan fungsinya**
- Waktu eksekusi kondisional if: **c**
- Waktu eksekusi operasi luaran: **c**

Didapatkan total waktu eksekusi algorimat tersebut adalah N*c + N*N*c + c. Perhatikan bahwa untuk sebuah kontanta c, apabila variabel N nilainya diubah menjadi sangat-sangat besar, maka eksistensinya tidak signifikan untuk dihitung dibandingkan dengan kenaikan N yang fluktual. Begitu pula variabel N perkembangan nilainya akan sangat kurang signifikan terhadap perkembangan N*N. Oleh sebab itu, nilai operasi yang lebih kecil dapat diabaikan dan didapatkan kompleksitas waktunya N^2 .

Untuk menotasikan kompleksitas waktu dari sebuah algoritma ini digunakan sebuah notasi yang dinamaka big-O notation yang menjelaskan mengenai hubungan orde pertumbuhan dengan kompleksitas waktu eksekusi program. Contoh penggunaannya pada hasil kompleksitas algoritma di atas adalah $O(N^2)$. Orde pertumbuhan adalah teori mengenai bagaimana waktu eksekusi sebuah program dipengaruhi oleh panjang masukannya. Beberapa kompleksitas waktu yang secara umum diketahui dan diterima dalam competitive programming adalah sebagai berikut.

TABLE I. LIST KOMPLEKSITAS WAKTU UMUM

General Time Complexities List		
Input Length	***Worst Accepted Time Complexity***	***Usually Type of Solutions***
10-12	O(N!)	Rekursif, backtracking
15-18	$O(2^N * N)$	Rekursif, backtracking, dan *bit manipulation*
18-22	$O(2^N * N)$	Rekursif, backtracking, dan *bit manipulation*
30-40	$O(2^{N/2} * N)$	Meet in the middle, Divide and conquer
100	$O(N^4)$	Dynamic programming, constructive
400	$O(N^3)$	Dynamic programming, constructive
2K	$O(N^2 * \log N)$	Dynamic programming, binary search, sorting, divide and conquer
10K	$O(N^2)$	Dynamic programming, graph, trees, contructive
1M	O(N * log N)	Sorting, binary search, divide and conquer
100M	O(N), O(log N), O(1)	Constructive, mathematical, greedy algorithms

(*Sumber: geeksforgeeks.org*)

2) *Kompleksitas Ruang*

Kompleksitas ruang dari sebuah algoritma merupakan hasil perhitungan kuantitatif mengenai jumlah penyimpanan yang digunakan oleh suatu algoritma saat dijalankan sebagai fungsi dari masukan dengan panjang tertentu. Contoh dari perhitungan kompleksitas ruang adalah sebagai berikut:

```
int freq[n];
int a[n];

for(int i = 0; i<n; i++)
{
   cin>>a[i];
   freq[a[i]]++;
}
```

Fig. 2. *Pseudo-code* algoritma perhitungan frekurensi elemen array A (*Sumber: geeksforgeeks.org*)

Gambar diatas merukan pseudo-code fungsi perhitungan frekuensi elemen dari array a yang kemudian frekuensi dari i disimpan pada array freq sebagai jumlah pada elemen ke-i. Pada kasus ini, digunakan sebuah kontanta c sebagai asumsi ruang yang digunakan oleh rata-rata tipe

data. Kemudian dapat dihitung total ruang yang digunakan adalah sebagai berikut:

- Ruang yang digunakan pada array *freq*: **N*c**
- Ruang yang digunakan pada array *n*: **N*c**
- Ruang yang digunakan variabel *i:* **c**

Didapatkan total ruang yang digunakan adalah N*c + N*c + c Dengan cara dan asumsi yang sama dengan perhitungan kompleksitas waktu, didapatkan kompleksitas ruang yang digunakan oleh fungsi tersebut adalah O(N).

Dari kedua metode perbandingan keoptimalan algoritma ini, metode yang digunakan sebagai tolak ukur utama kompleksitas algoritma adalah metode kompleksitas waktu. Goldreich (2008) menjelaskan bahwa pertimbangan kompleksitas waktu eksekusi selain dilihat berdasarkan *worst-case scenario* seperti pada contoh di subbab kompleksitas waktu, juga perlu dihitung berdasarkan *best-case scenario* (kasus terbaik) dan *average-case scenario* (kasus rerata). Pada kerja sebuah algoritma, semakin sedikitnya waktu yang digunakan, maka kerja dianggap semakin baik.

Analisis menggunakan *best-case scenario* berguna untuk memperbaiki akurasi analisis *worst-case scenario* sekaligus sebagai *benchmark* bagi algoritma-algortima lain meskipun pada kenyataan komputasi *real-time*, kasus ini jarang sekali digunakan. Analisis *average-case scenario* dan *worst-case scenario* menjadi fokus utama dalam perhitungan kompleksitas algoritma. Namun, keduanya memiliki sejumlah kemiripan dalam ketidakpastian nilai dan penentuan titiknya. Pada *average-case scenario,* perhatian utama terjadi pada sifat-sifat tipe data yang berbeda sehingga sukar dikarakterisasi secara matematis sehingga kebanyakan analisis kasus rerata ini dilakukan hanya pada algoritma yang beroperasi pada teks string karakter. Sedangkan pada *worst-case scenario*, penentuan kondisi terburuk ini sebenarnya sangat sukar ditemukan, bahkan tidak mungkin secara eksak ditentukan. Sebagai alternatif, digunakan *scenario* yang dipandang dekat atau bentuk *underestimate* dari kasus terjelek yang sebenarnya dapat terjadi.

Selanjutnya, analisis dari kasus-kasus ini dapat dijelaskan menggunakan *big-o notation. Big-o notation* memiliki definisi sebagai berikut:

Jika *n* adalah ukuran masukan dan f(n), g(n) adalah fungsi positif dari *n*, maka f(n) memiliki kompleksitas waktu O(g(n)) jika dan hanya jika terdapat konstanta positif *c* dan bilangan bulat positif n_0 sedemikian rupa sehingga $f(n) \leq c\ g(n)$ untuk semua $n \geq n_0$

Deskripsi hasil analisis kompleksitas algoritma menggunakan *big-o notation* kemudian dapat diperluas dengan komponen-komponen yang dimiliki sebagai berikut:

1) Kompleksitas Presisi

Kompleksitas presisi yang disimbolkan dengan T(n) a adalah perhitungan kompleksitas sebuah algoritma yang menghitung secara eksak dan menampilkan hasil perhitungan kompleksitas waktunya dengan spesifik. Contohnya, pada perhitungan kompleksitas waktu yang sebelumnya dilakukan terhadap algoritma pair(X,Y) nilai T(n) = N*c + N*N*c + c.

2) Kompleksitas Batas Atas Asimptotik

Kompleksitas batas atas asimptotik atau biasa disebut sebagai *big-o notation* disimbolkan dengan O(n). Kompleksitas ini menghitung penggunaan waktu paling besar dari sebuah algoritma atau biasa disebut sebagai *worst-case performance.* Secara formal, kompleksitas batas atas asimptotik didefinisikan sebagai:

T(n) adalah O(f(n)), yang artinya T(n) berorde paling besar f(n) apabila terdapat terdapat konstanta positif *c* dan bilangan bulat positif n_0 sedemikian rupa sehingga $T(n) \leq c\ f(n)$ untuk semua $n \geq n_0$

Selain itu, teradapat beberapa teorema yang dapat digunakan dalam menentukan nilai T(n) dari sebuah algoritma.

Teorema 1:
Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ adalah polinom dengan derajat kurang dari sama dengan m ($d(T(n)) \leq m$), maka $T(n) = O(n^m)$

Teorema 2:
Misalkan $T_1(n) = O(f(n))$ $T_2(n) = O(g(n))$, maka
1. $T_1(n) + T_2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$
2. $T_1(n)\ T_2(n) = O(f(n))\ O(g(n)) = O(f(n)\ g(n))$
3. $O(c(f(n)) = O(f(n))$
4. $f(n) = O(f(n))$

Pada *big-o notation*, terdapat urutan spektrum kompleksitas waktu algoritma yang menandakan seberapa buruk atau bagusnya suatu kompleksitas dibandingkan kompleksitas lainnya. Urutan tersebut adalah sebagai berikut, dengan catatan semakin ke kanan atau semakin besar nilainya maka algoritma tersebut kualitasnya semakin buruk.

$$1 < \log n < n < n \log n < n^2 < n^3 < \ldots < 2^n < n!$$

3) Kompleksitas Batas Bawah Asimptotik

Kompleksitas batas bawah asimptotik atau biasa disebut sebagai *big-omega notation* disimbolkan dengan Ω(n) adalah waktu paling kecil yang diperlukan atau kemungkinan paling efisien dalam menjalankan sebuah algoritma (*best-case performance*). Secara formal, Ω(n) didefinisikan sebagai berikut:

T(n) adalah Ω(g(n)), yang artinya T(n) berorde paling kecil g(n) apabila terdapat terdapat konstanta positif *c* dan bilangan bulat positif n_0 sedemikian rupa sehingga $T(n) \geq c\ f(n)$ untuk semua $n \geq n_0$

4) Kompleksitas Paling Terikat

Kompleksitas paling terikat diketahui juga sebagi *big-tetha notation* disimbolkan sebagai Θ(n). Kompleksitas ini

menghitung yang terbaik dari semua kemungkinan kerja terburuk dari yang sebuah algoritma dapat jalankan. Secara formal, Θ(n) dapat didefinisikan sebagai berikut:

T(n) adalah Θ(h(n)), yang artinya T(n) berorde sama dengan h(n) apabila T(n) = O(h(n)) dan T(n) = Ω(h(n)).

B. *Algoritma* Outlier Detection *untuk Distribusi Non-Gauss*

Outlier adalah nilai atau hasil observasi yang sangat berbeda dari data-data atau titik-titik lain pada sampel dari sebuah populasi. Banyak hal yang dapat menyebabkan *outlier* dapat muncul dalam pengumpulan suatu data, antara lain seperti kesalahan manusia/ *human error* (*wrong data entry*), kesalahan pengukuran/ *measurement error* (*system/ tools error*), kesalahan pemilihan sampel/ *sampling error* (*creating samples from heterogenous sources*), kesalahan manipulasi data/ *data manipulation error* (*faulty data preprocessing error*), dan lain sebagainya.

Deteksi *outlier* adalah step paling mendasar dalam pengolahan sebuah data dan menjadi bagian paling penting dalam proses *data mining*. Deteksi ini berfungsi untuk mengetahui anomali-anomali dalam sebuah observasi atau sampel yang tidak sesuai dengan tipikal distribusi dari sebuah dataset. Tujuan dari deteksi *outlier* ini adalah untuk mengetahui parameter-parameter yang dipengaruhi oleh *outlier tools* dari parameter yang jumlahnya sangat banyak.

Secara spesifik, deteksi *outlier* dilakukan berdasarkan jenis *dataset* yang dimiliki. Untuk *dataset* dengan tipe distirbusi gaussian atau normal, penentuan *outlier* dapat dengan sederhana dilakukan menggunakan metode asumsi normalitas atau *normality assumption.* Sedangkan, terdapat metode tertentu untuk mengetahui nilai-nilai *outlier* dari data dengan distribusi non-gaussian atau non-gauss.

Distribusi non-gaussian, non-gauss, atau non-normal secara sederhana menjelaskan mengenai data yang tidak dapat dianggap atau direpresentasikan sebagai distribusi normal yang berbentuk seperti *bell-curve*. Distribusi non-gauss sendiri memiliki berbagai bentuk yang kemudian dapat diklasifikasikan lagi menjadi *beta distribution, exponential distribution, gamma distribution, inverse gamma distribution, log nomal distribution, logistic distribution, maxwel-boltzman distribution, poisson distribution, skewed distribution, symmetric distribution, uniform distribution, unimodal distribution,* dan *weibull distribution.*

Banyak tipe-tipe *dataset* yang memang secara natural memenuhi distribusi non-gauss, namun beberapa kasus yang terjadi juga memungkinkan *dataset* yang harusnya terdistibusi secara *gauss* mengalami penyimpangan akibat adanya *outlier*, terdapat kombinasi distribusi pada *dataset* yang harus diolah kembali, tidak cukupnya data yang dimiliki, dan pembentukan grafik yang tidak sesuai.

Untuk data yang tergolong dalam *dataset* berdistribusi non-gauss, terdapat banyak metode atau kriteria untuk menentukan *outlier*. Pada makalah ini, perbandingan kompleksitas algoritma akan dilakukan pada lima (5) metode yang paling umum digunakan untuk *outlier detection* pada data terdistribusi non-gaussian. Metode-metode tersebut adalah sebagai berikut:

1) Pierce's Criterion

Metode Pierce merupakan turunan *outlier detection* dari distribusi gaussian. Metode ini dipertimbangkan untuk dibandingkan dengan metode-metode lain karena dapat menghilangkan lebih dari 1 outlier dalam sekali keberjalanan algoritma. Langkah-langkah deteksi menggunakan metode Pierce adalah sebagai berikut:

1. Hitung nilai rerata μ dan standard deviasi σ dari suatu *dataset* uji
2. Tentukan nilai R dengan asumsikan kasus sementara hanya memiliki satu *outlier*. Nilai R dapat diperoleh dari *Pierce's criterion table* (Fig.3) atau dengan rumus

$$R[m] = p_1[m] * \log(n) + p_2[m]$$

dengan m adalah asumsi jumlah outlier ($m \leq 9$), nilai p_1 diperoleh dari array *[0.4094 0.4393 0.4565 0.4680 0.477 0.4842 0.4905 0.4973 0.5046],* dan nilai p_2 diperoleh dari array *[0.991 0.6069 0.3725 0.2036 0.0701 -0.0401 -0.1358 -0.2242 -0.3079]*

	Number Of Suspected Outliers			
n	1	2	3	4
3	1.196			
4	1.383	1.078		
5	1.509	1.200		
6	1.610	1.299	1.099	
7	1.693	1.382	1.187	1.022
8	1.763	1.453	1.261	1.109
9	1.824	1.515	1.324	1.178
10	1.878	1.570	1.380	1.237
11	1.925	1.619	1.430	1.289
12	1.969	1.663	1.475	1.336
13	2.007	1.704	1.516	1.379
14	2.043	1.741	1.554	1.417
15	2.076	1.775	1.589	1.453
20	2.209	1.914	1.732	1.599
25	2.307	2.019	1.840	1.709
50	2.592	2.326	2.158	2.035

Fig. 3. *Pierce's criterion table of R value* (*Sumber: Maas, Y.R. 2019*)

3. Untuk setiap titik eksentrik x^*, hitunglah nilai $|x^* - \mu|$
4. Buang nilai praasumsi x^* apabila $\sigma R < |x^* - \mu|$
5. Apabila hanya terdapat satu titik yang dihilangkan dari *dataset*, kemudian asumsikan kasus saat ini berlaku untuk dua *outliers* dengan rerata dan standard deviasi yang sudah didapatkan, selanjutnya lanjutkan ke langkah kedelapan
6. Apabila terdapat lebih dari satu titik yang dihilangkan dari *dataset*, maka asumsikan kasus maksimal selanjutnya dari dataset dengan rerata dan standard deviasi yang sudah didapatkan. Kemudian lanjutkan ke langkah kedelapan
7. Ulangi langkah kedua s.d. kelima hingga tidak ada *outlier* yang ditemukan
8. Tentukan nilai rerata dan standard deviasi terbaru dari *dataset* yang telah tereduksi

2) Chauvenet's Criterion

Metode Chauvenet adalah metode yang mengambil asumsi terhadap normalias. Metode ini kerap digunakan pada berbagai intitusi pendidikan dan laboratorium sebagai *outlier detection algorithm.* Meskipun metode ini cukup akurat untuk menentukan kebenaran deduksi ke-*outlier*-an sebuah nilai, namun metode ini dapat menghilangkan lebih dari 1 outlier dalam sekali keberjalanan algoritma. untuk membuang sebuah nilai data. Metode ini juga tidak membedakan antara asumsi jumlah *outlier* pada *dataset.* Secara lebih terinci, langkah-langkah deteksi menggunakan metode Chauvenet adalah sebagai berikut:

1. Hitung nilai rerata μ dan standard deviasi σ dari suatu *dataset* uji
2. Buang nilai eksentrik x^* apabila $erfc\left(\frac{|x^* - \mu|}{\sigma}\right) < \frac{1}{2n}$
3. Ulangi langkah pertama dan kedua
4. Tentukan nilai akhir μ, σ, dan *n*

Pada langkah kedua, asumsi *Chauvenet's criterion* digunakan. Jika $erfc\left(\frac{|x^* - \mu|}{\sigma}\right) < \frac{1}{2n}$ bernilai benar, maka nilai eksentrik x^* harus dihilangkan. Fungsi *erfc* adalah *complementary error function* yang definisinya adalah sebagai berikut:

$$\text{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt$$

Yang dapat ditulis juga sebagai $\mathbb{P}(Y \in [-x, x])$ dengan $Y \sim N\left(0, \frac{1}{2}\right)$, sehingga dapat kita lihat asumsi normalitas pada langkah kedua. Nilai $\frac{1}{2}$ dapat diinterpretasikan sebagai pemberian 50% poin survival untuk setiap titik atau dalam kata lain, pasti ada titik yang lebih dekat ke nilai rerata dengan jumlah yang sama dengan titik yang lebih jauh ke nilai rerata. Ketika posisinya terlalu jauh, maka sebuah data dianggap sebagai sebuah *outlier* dan harus dibuang dari *database.*

Dengan metode ini, dapat dilihat sebuah fenomena baru yakni *shielded outlier* dimana terdapat sebuah *outlier* yang pada awalnya tidak diperhatikan sebagai *outlier* karena nilai rerata awal pada *dataset* yang mengandung *outlier* pra-buang masih termasuk dekat dengan nilai tersebut namun setelah pembuangan pertama, nilai ini kemudian terdeteksi sebagai *outlier* karena reratanya menjadi normal. Oleh karena itu, metode ini perlu dilakukan berberapa kali hingga benar-benar tidak ditemukan *outlier* pada rerata terbaru.

3) Grubb's Test

Tes Grubb atau yang biasa dikenal sebagai tes nilai maksimal residu ternormalisasi (*maximum normalized residual test*) atau tes deviasi ekstrim terstudifikasi (*extreme studentized deviate test*) adalah metode deteksi *outlier* pada *dataset* univariat. Tes ini juga menggunakan asumsi normalitas (*normality assumption*). Tes Grubb akan menghasilkan tepat satu *outlier* dalam sekali waktu sehingga sebuah titik yang di tes pasti tepat titik maksimum atau titik minimum dari sebuah *dataset,* sehingga tes ini perlu dilakukan berkali-kali untuk membersihkan *dataset* dari *outlier*. Tes kurang baik bila digunakan untuk data set yang isinya lebih dari 6 titik karena akan menganggap hampir seluruh data sebagai *outlier*. Tes ini berjalan dengan dua hipotesis:

- H_0 (*null hypothesis*): Tidak ada *outlier* pada *dataset*
- H_a (*alternative hypothesis*): Terdapat 1 *outlier* pada *dataset*

Tes akan melihat apakah kita perlu memilih hipotesis H_0 atau H_a terlebih dahulu. Bentuk statistik dari tesnya adalah sebagai berikut:

$$G = \frac{max_{i=1,\dots,n}|x_i - \mu|}{\sigma}$$

Nilai G kemudian menjadi tolak ukur. H_a akan dipilih saat $G > \frac{n-1}{\sqrt{n}} \sqrt{\frac{t^2_{\frac{\alpha}{n}, n-2}}{n - 2 + t^2_{\frac{\alpha}{n}, n-2}}}$ sedangkan saat kondisi tidak memenuhi, maka H_0 akan dipilih untuk dieksekusi. Saat H_a dieksekusi, maka setelahnya dapat dilakukan Tes Grubb lagi, sedangkan sebaliknya saat H_0 maka tes berhenti dilakukan.

4) Z-score

Z-score, *standard score,* z-value, *normal scores*, atau *standarized variabel* adalah kriteria yang dapat dibuat untuk melakukan deteksi *outlier*. Nilai Z didefinisikan sebagai berikut:

$$Z = \frac{x - \mu}{\sigma}$$

Dengan *x* adalah titik pada *dataset*, μ adalah nilai rerata, dan σ adalah standard deviasi. *Outlier detection* dengan metode ini merupakan metode yang terus terang. Semua data pada *dataset* dihitung Z-score nya kemudian ditentukan *outlier* dari data-data tersebut apabila $|Z| > 3$, dengan nilai 3 didapatkan dari percobaan empiris yang berkaca pada distribusi normal bahwa 99.7% data berada pada interval $[\mu - 3\sigma, \mu + 3\sigma]$ seperti yang tergambarkan pada Fig 4. Nilai-nilai *outlier* ini kemudian dibuang dari *dataset* sehingga didapatkan *dataset* yang bersih. Secara lebih terinci, langkah-langkah *outlier detection* menggunakan metode Z-score adalah sebagai berikut:

1. Tentukan nilai maksimal atau nilai minimal sebagai x^*
2. Hitung nilai rerata μ dan standard deviasi σ dari sisa data pada *dataset*
3. Tentukan x^* merupakan *outlier* apabila $|x^*| > \mu + 3\sigma$
4. Jika x^**outlier*, buang x^*, kemudian lakukan step pertama hingga keempat kembali pada sisa data

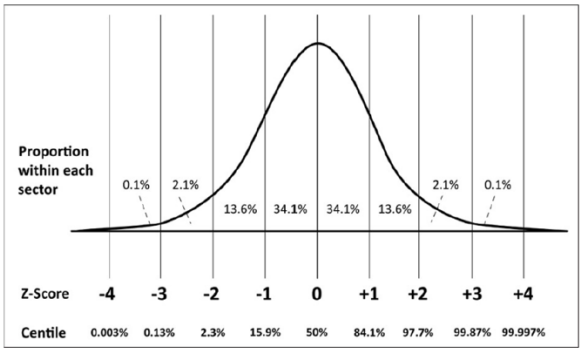


Fig. 4. Distribusi Z-score (Sumber: Maas, Y.R. 2019)

5) MAD (Median absolute deviation)

Metode MAD adalah metode yang biasa digunakan pada data dengan distribusi gaussian. Metode ini juga berlaku untuk *outlier detection* pada data denga distribusi

non-gaussian. Secara umum, MAD didefinisikan sebagai berikut:

$$MAD = bM_i\left(\left|x_i - M_j\left(x_j\right)\right|\right)$$

Dengan $b = \frac{1}{Q(0.75)}$, di mana Q(0.75) adalah kuartal ketiga dari distribusi data; M_i adalah median dari *dataset*; x_i adalah nilai observasi; dan $M_j(x_j)$ adalah median dari titik observasi x_j. Secara lebih detail langkah-langkah penggunaan metode MAD adalah sebagai berikut:

1. Kurangkan semua titik dalam *dataset* dengan nilai median dan simpan nilai mutlaknya pada *set* baru
2. Urutkan nilai-nilai yang baru ini dari terendah ke tertinggi
3. Hitung median dari *set* baru ini yakni $M_i\left(\left|x_i - M_j\left(x_j\right)\right|\right)$
4. Kalikan median baru ini dengan *b* untuk mendapatkan nilai MAD
5. Tentukan apakah sebuah titik x^* bukan *outlier* dengan kriteria sebagai berikut:

   $$M - C \cdot MAD < x^* < M + C \cdot MAD$$

   Nilai *M* merupakan median dari data awal, sedangkan konstanta *C* dapat dipilih dari 3 nilai; 3 untuk *very conservative,* 2.5 untuk *moderately conservative*, dan 2 untuk *poorly conservative*.
6. Apabila x^* berada dalam *range* $(M - C \cdot MAD, M + C \cdot MAD)$, maka x^* bukanlah *outlier* serta kebalikannya.

III. KOMPLEKSITAS ALGORITMA PIERCE'S CRITERION

```
#Peirce normal distribution
set.seed(349)

n = 500;
x = rnorm(n,0,1);                                   (1)
mu = mean(x);                                       (2)
sigma = sd(x);                                      (3)
alpha = 0.1;
xc = qnorm(nthroot(1-alpha,n))                      (4)
outliers <- vector();

a = c(0.4094, 0.4393, 0.4565, 0.4680, 0.477,
0.4842, 0.4905, 0.4973, 0.5046);
b = c(0.991, 0.6069, 0.3725, 0.2036, 0.0701, -
0.0401, -0.1358, -0.2242, -0.3079);
R_values = a*log(n)+b                               (5)

mad1 = sigma*R_values[1]                            (6)
v1 <- vector()
for (i in 1:n){
      (7)
 v1[i] <- abs(x[i]-mu)
}

v2 <- vector()
for (i in 1:n){
      (8)
 if (mad1 < v1[i]){
  v2[i] <- x[i]
 }
}
outliers_val = na.omit(v2)                          (9)
outliers <- append(ourliers_val)                    (10)
```

```
while (length(outliers_val) != 0) {                 (11)
 if (length(outliers_val) < 9) {
  k <- length(outliers_val);
 } else {
  k <- 9
 }

 madk = sigma*R_values[k]
 v1 <- vector()
 for (i in 1:n){                                    (12)
  v1[i] <- abs(x[i]-mu)
 }

 v2 <- vector()
 for (i in 1:n){                                    (13)
  if (madk < v1[i]){
   v2[i] <- x[i]
  }
 }

 outliers_val = na.omit(v2)                         (14)
 outliers <- append(ourliers_val)                   (15)

}
```

Fig. 5. *Pseudo-code Pierce's Criterion dalam Bahasa R* (Sumber: Maas, Y.R. 2019)

Algoritma Pierce's Criterion memiliki *pseudocode* seperti yang dijelaskan pada Fig 5. Algoritma tersebut diambil dari tesis Y.R. Mass, 2019 yang berjudul "Outlier detection in non-Gaussian distributions" dengan gubahan yang diperlukan.

Pada baris yang ditandai nomor (1), terdapat fungsi *rnorm* yang tersedia pada dokumentasi R dengan terimaan input n, mean, dan var yang akan menghasilkan varitas acak sejumlah n dengan rerata mean dan janis variasi var. Pembanggilan fungsi ini memerlukan kompleksitas algoritma sejumlah O(N). Pada bagian bernomor (2), (3), dan (4) terdapat pemanggilan fungsi rerata (mean), standard deviasi (sd), dan qnorm dengan kompleksitas yang sama yakni O(N).

Untuk bagian (5), terdapat perkalian vektor dengan scalar diikuti penjumlahan vektor. Namun karena panjang vektor yang dioperasikan selalu tepat 9, maka kompleksitasnya O(1). Begitu pula untuk operasi bagian (6), kompleksitasnya juga ttetap O(1).

Bagian (7) dan (8) menampilkan *loop* dengan banyak pengulangan N, sehingga keduanya tepat memiliki kompleksitas O(N). Sedangkan pada bagian ke-(9) terdapat penghapusan nilai-nilai NA dari vektro v2. Kasus terburuk unruk bagian ini adalah di mana semua bagian dari v2 bernilai NA, sehingga *deleting* dan *shifting* dilakukan satu demi satu sebanyak N. Total dari kompleksitas waktu terburuk dari bagian ini adalah O(N).

Pada bagian (10), dilakukan penambahan pada elemen terakhir dari *outliers* sebanyak jumlah *outlier* terdeteksi pada *outlier_val*. Algoritma iniakan memakan kompleksitas waktu terburuk pada kasus yang sama dengan *worst-case* bagian ke (9), yakni saat banyak elemen *outliers_val* adalah n. Kompleksitas waktu bagian ini adalah O(N) dalam notasi *big-o*.

Bagian ke-(11) adalah pengulangan untuk bagian nomor (6) s.d. (10) dan pengulangan ini nilai terburuknya adalah ketika *outlier* ditemukan satu demi satu, sehingga *while loop* akan dijalankan sebanyak N-1 kali. Maka *big-o* notation yang representatif sama dengan O(f(n)) dengan f(n) merupakan kompleksitas waktu dari bagian (6) s.d. (10).

Total perhitungan nilai *big-o* daritahap-tahap ini adalah sebagai berikut:

Bagian (1) s.d. (5)

$$O\big(h(n)\big) = O(N) + O(N) + O(N) + O(N) + O(1)$$
$$= O(N)$$

Bagian (6) s.d. (10)

$$O(f(n)) = O(1) + O(N) + O(N) + O(N) + O(N)$$
$$= O(N)$$

Bagian (11) s.d. (15)

$$O\big(g(n)\big) = (N-1) * O\big(f(n)\big)$$
$$= O(N^2)$$

Dari tiap bagian itu, kemudian dihitung *big-O* untuk keseluruhan algoritma sebesar $O(n) = O\big(h(n)\big) + O\big(f(n)\big) + O\big(g(n)\big) = \boldsymbol{O(N^2)}$

IV. KOMPLEKSITAS ALGORITMA CHAUVENET'S CRITERION

Algoritma Chauvenet's Criterion memiliki *pseudocode* seperti pada Fig 6. Algoritma tersebut diambil dari tesis Y.R. Mass, 2019 yang berjudul "Outlier detection in non-Gaussian distributions" dengan gubahan yang diperlukan.

```
#Chauvenet normal distribution
set.seed(501)

n = 500;
x = rnorm(n,0,1);                                    (1)
mu = mean(x);                                        (2)
sigma = sd(x)                                        (3)

v1 <- vector ()
for (i in 1:n){                                      (4)
 v1[i] = (abs(x[i]-mu))/sigma
}

v2 <- vector()
for (i in 1:n){                                      (5)
 if (n*erfc(v1[i])< 0.5){
  v2[i] <- x[i]
 }
}

outliers = na.omit(v2)                               (6)
```

Fig. 6. *Pseudo-code Chauvenet's Criterion dalam Bahasa R* (Sumber: Maas, Y.R. 2019)

Pada algoritma tersebut, dapat dilihat bagian (1), (2), dan (3) memanggil fungsi rnorm, mean, dan sd sebagaimana bagian (1), (2), dan (3) pada algoritma Pierce's Criterion. Oleh karena itu, bagian ini memiliki kompleksitas algorithm dalam *big-o notation* masing-masing sebesar O(N).

Bagian (4) dan (5) masing-masing melakukan *for loop* dengan banyak pengulangan N. Kedua fungsi tersebut memiliki *body* yang memiliki kompleksitas algoritma O(1), sehingga secara keseluran kompleksitas algoritma dari tiap-tiap loop ini adalah O(N). Pada bagian (6), fungsi omit sama dengan bagian (9) pada algoritma Pierce's Criterion, seperti yang diketahui bahwa nilainya O(N).

Oleh karena ini, secara lengkap, perhitungan total *big-O* untuk keseluruhan algorima adalah $O(n) = O(N) + O(N) + O(N) + O(N) + O(N) + O(N) = \boldsymbol{O(N)}$

V. KOMPLEKSITAS ALGORITMA GRUBB'S TEST

Algoritma Grubb's Test memiliki *pseudocode* seperti pada Fig 7. Algoritma tersebut diambil dari tesis Y.R. Mass, 2019 yang berjudul "Outlier detection in non-Gaussian distributions" dengan gubahan yang diperlukan.

```
#Grubb normal distribution
set.seed(212)

n = 500;
x = rnorm(n,0,1);                                    (1)
alpha = 0.1;
xc = qnorm(nthroot(1-alpha,n))                       (2)
#x[n+1] <- xc*1.1
st = n;

for (t in 1:n){                                      (3)
 mu = mean(x);                                       (4)
 sigma = sd(x)                                       (5)
 v1 <- vector ()
 for (i in 1:st){                                    (6)
  v1[i] = abs(x[i]-mu)
 }
 abnormalpoint = max(v1)+mu;                         (7)

 G = max(v1)/sigma;                                  (8)
 alpha = 0.1;
 p = 1-alpha/(2*n);                                  (9)
 df = n-2;
 t = qt(p,df)                                        (10)

 G0 = ((n-1)/sqrt(n))*sqrt(t^2/(n-2+t^2))(11)

 if (G > G0){                                        (12)
  outlier = abnormalpoint;
  setdiff(x,outlier);                                (13)
  st = st - 1
 }
}
```

Fig. 7. *Pseudo-code Grubb's Test dalam Bahasa R* (Sumber: Maas, Y.R. 2019)

Pada algoritma tersebut, dapat dilihat bagian (1) digunakan fungsi rnorm dengan kompleksitas algorima O(N). Sedangkan pada bagian (2), terdapat fungsi qnorm yang memanggil fungsi nthroot, masing-masing memiliki kompleksitas O(N) dan O(1), sehingga total kompleksitasnya sama dengan O(N*1) = O(N).

Bagian (3) selanjutnya melakukan *for loop* sebanyak N kali pengulangan terhadap kompleksitas algoritma *body* nya yang mencakup urutan nomor (4) s.d. (13).

Pada nomor (4) dan (5), fungsi *mean* dan *standard deviation* yang masing-masing memiliki kompleksitas algortima O(N). Selanjutnya bagian (6) melakukan *for loop* sebanyak st kali, di mana $st \leq n$, maka kompleksitas algoritmanya terhitung sebagai O(N). Pada bagian (7) dan (8) terdapat operasi dasar yang memanggil fungsi *max* yang memiliki kompleksitas

algorima O(N), maka bagian (7) dan (8) juga memiliki kompleksitas algoritma sebesar O(N).

Bagian (9), (10), dan (11) masing-masing melakukan operasi aritmatika dasar dengan kompleksitas konstan atau O(1). Dilanjutkan dengan pengecekan *if statement* pada bagian (12) yang memiliki kompleksitas O(N) berdasarkan *body function* nya pada nomor (13) yang memanggil fungsi *setdiff* yang memiliki kompleksitas algoritma O(N).

Dari penjabaran tersebut, *body function* dari *for loop,* yakni bagian (4) s.d. (13) memiliki kompleksitas algoritma $O(f(n)) = O(N) + O(N) + O(N) + O(N) + O(N) + O(1) + O(1) + O(1) + O(N) + O(N) = O(N)$. Karena *loop* (3) dilakukan sebanyak N kali, maka $O(h(n)) = O(N) * O\big(f(n)\big) = O(N^2)$.

Oleh karena itu, secara lengkap kompleksitas algoritma *Grubb's Test* dapat dijelaskan sebagai hasil penjumlahan kompleksitas algoritma bagian (1), (2), dan (3) atau secara matematis dituliskan dalam bentuk $O(n) = O(N) + O(N) + O(N^2) = \boldsymbol{O(N^2)}$

VI. KOMPLEKSITAS ALGORITMA Z-SCORE

```
#Z-score normal distribution
set.seed(66)

n = 500;
x = rnorm(n,0,1);                                    (1)
mu = mean(x);                                        (2)
sigma = sd(x);                                       (3)

Zscores <- vector ()
for (i in 1:n){                                      (4)
  Zscores[i] <- (x[i]-mu)/sigma
}

v <- vector()
for (i in 1:n){                                      (5)
  if (abs(Zscores[i]) > 3){
    v[i] <- x[i]
  }
}

outliers = na.omit(v)                                (6)
```

Fig. 8. *Pseudo-code Z-Score dalam Bahasa R* (Sumber: Maas, Y.R. 2019)

Algoritma Z-score memiliki *pseudocode* seperti yang dijelaskan pada Fig 8. Algoritma tersebut diambil dari tesis Y.R. Mass, 2019 yang berjudul "Outlier detection in non-Gaussian distributions" dengan gubahan yang diperlukan.

Pada bagian (1), (2), dan (3) dari algoritma ini, ditemukan kesamaan dengan algoritma Pierce's Criterion dan Chauvenet's Criterion. Artinya ketiga bagian ini tepat berkompleksitas O(N).

Selanjutnya, pada bagian (4) dan (5) dilakukan *for loop* dengan N kali pengulangan dan *body function* yang berupa operasi biasa yang berkompleksitas O(1). Maka bagian (4) dan (5) masing-masing memiliki kompleksitas sebesar O(N*1) = O(N). Terakhir, bagian (6) merupakan fungsi *na omit* yang memiliki kompleksitas O(N).

Dari perhitungan yang dilakukan, diperoleh kompleksitas algoritma lengkap dari Z-score adalah $O(n) = O(N) + O(N) + O(N) + O(N) + O(N) + O(N) = \boldsymbol{O(N)}$

VII. KOMPLEKSITAS ALGORITMA MAD (MEDIAN ABSOLUTE DEVIATION)

```
#MAD normal distribution
set.seed(666)

n = 500;
x = rnorm(n,0,1)                                     (1)
xs = sort(x)                                         (2)
b = 1/(quantile(xs, 0.75))                           (3)
M = median(xs)                                       (4)

ab <- vector()
for (i in 1:n){                                      (5)
 ab[i] <- abs(xs[i]-M)
}

abs = sort(ab)                                       (6)

Mi = median(abs)                                     (7)
MAD = b*Mi
C = 2.5

v <- vector()
for (i in 1:n){                                      (8)
 if (abs((xs[i]-M)/MAD) > C){
  v[i] <- xs[i]
 }
}

outliers = na.omit(v)                                (9)
```

Fig. 9. *Pseudo-code MAD dalam Bahasa R* (Sumber: Maas, Y.R. 2019)

Algoritma MAD memiliki *pseudocode* seperti yang dijelaskan pada Fig 9. Algoritma tersebut diambil dari tesis Y.R. Mass, 2019 yang berjudul "Outlier detection in non-Gaussian distributions" dengan gubahan yang diperlukan.

Pada algoritma ini, bagian (1) dipanggil fungsi rnorm yang memiliki kompleksitas algoritma O(N). Selanjutnya pada bagian (2), dilakukan *sorting* yang memiliki kompleksitas algoritma ambang bawah $O(N^2)$. Bagian (3) dan (4) memanggil fungsi quantile dan median yang masing-masing memiliki kompleksitas O(N).

Kemudian, pada bagian (5) dilakukan *for loop* dengan N pengulangan untuk *body function* dengan kompleksitas O(1). Maka, kompleksitas *loop* ini adalah O(N). Bagian (6) memanggil fungsi yang sama seperti bagian (2) dan bagian (7) memanggil fungsi yang sama dengan bagian (4). Masing-masing bagian ini memiliki kompleksitas $O(N^2)$ dan O(N). Sedangkan *loop* pada bagian ke-(8) juga memiliki kompleksitas yang sama dengan *loop* pada bagian (5) atau O(N). Terakhir, fungsi *omit* pada bagian (9) memerlukan kompleksitas O(N).

Oleh karena itu, melalui perhitungan yang telah dilakukan diperoleh kompleksitas algoritma lengkap dari Z-score adalah $O(n) = O(N) + O(N^2) + O(N) + O(N) + O(N) + O(N^2) + O(N) + O(N) + O(N) = \boldsymbol{O(N^2)}$

VIII. KESIMPULAN

Berdasarkan semua data yang telah diolah pada bagian III s.d. VII, diperoleh table kompleksitas tiap metode *outlier detection* sebagai berikut:

TABLE II. PERBANDINGAN KOMPLEKSITAS ALTERNATIF ALGORITMA OUTLIER DETECTION UNTUK DISTRIBUSI NON-GAUSS

Metode *Outlier Detection*	**Hasil Pengolahan Data**	
	Kompleksitas Algoritma	***Kelebihan Kualitatif Algoritma***
Pierce's Criterion	$O(N^2)$	Menghilangkan lebih dari 1 outlier dalam sekali keberjalanan algoritma sekaligus memiliki konstanta *normality assumption* yang menyesuaikan sengan asumsi jumlah *outlier* yang akan dideteksi, sehingga lebih presisi
Chauvenet's Criterion	$O(N)$	Akurat dalam menentukan nilai *outlier* meskipun menggunakan *arbitrary assumption*/ asumsi sembarang dalam algoritmanya
Grubb's Test	$O(N^2)$	Metode mudah dipahami dan akurat karena data yang diolah adalah data dengan nilai ekstrim dan dilakukan pengecekan kemungkinan hipotesis sebelum pengolahan dilakukan
Z-score	$O(N)$	Metode deteksi sederhana serta *checking* dilakukan kepada semua poin dengan indikator yang jelas sehingga akurasi cukup tepat
MAD	$O(N^2)$	Metode cukup fleksibel karena juga berlaku untuk *outlier detection* pada data dengan distribusi non-gaussian.

Berdasarkan data yang telah diolah, dapat dilihat bahwa rata-rata metode untuk *outlier detection* memiliki kompleksitas $O(N^2)$ dan $O(N)$. Di antara semua metode, dua metode yang memiliki kompleksitas paling sederhana adalah Chauvenet's Criterion dan Z-score sehingga kedua metode ini dapat dikonsideriasasi untuk dipilih sebagai metode *outlier detection*. Antara Chauvenet's Criterion dan Z-score meskipun kompleksitasnya sama, Z-score memiliki kelebihan secara kualitatif karena *checking* dilakukan secara merata dan tidak menggunakan *arbitrary assumption* sehingga hasilnya akan lebih konsisten. Oleh karena itu, metode Z-score dipertimbangkan sebagai metode dengan kompleksitas algoritma yang paling tepat sebagai *outlier detection* untuk data dengan distribusi non-gauss.

VII. UCAPAN TERIMAKASIH

Penulis bersyukur kepada Tuhan Yang Maha Esa karena telah melimpahkan rahmat-Nya sehingga penulis mampu menyelesaikan makalah berjudul "Analisis Perbandingan Kompleksitas Alternatif Algoritma Outlier Detection untuk Distribusi Non-Gauss". Penulis mengucapkan terima kasih kepada orang tua, bapak/ibu dosen, teman-teman, dan sahabat penulis yang selalu memberikan dukungan, motivasi, dan masukan sehingga penulis mampu menyelesaikan makalah dengan baik.

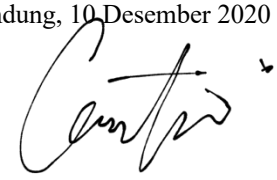
REFERENSI

[1] Maas, Y. 2019. Outlier detection in non-Gaussian distributions.
[2] Subandijo, S. 2011. Efisiensi Algoritma dan Notasi O-Besar. *ComTech: Computer, Mathematics and Engineering Applications*, *2*(2), 849. https://doi.org/10.21512/comtech.v2i2.2835
[3] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. 2002. *Introduction to Algorithms 2nd Edition*. Massachusetts: MIT Press.
[4] Sedgewick, R. 1998. *Algorithms in C++ Part 1-4*. Massachusetts: Addison-Wesley Publisher.
[5] Munir, Rinaldi. 2022. "Kompleksitas Algoritma (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf (Diakses pada tanggal 6 Desember 2021 Pukul 19.20 WIB)
[6] Munir, Rinaldi. 2022. "Kompleksitas Algoritma (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian2.pdf (Diakses pada tanggal 6 Desember 2021 Pukul 19.20 WIB)
[7] GeeksforGeeks. 2022. *Time Complexity and Space Complexity*. https://www.geeksforgeeks.org/time-complexity-and-space-complexity/
[8] GeeksforGeeks. 2022. *Difference between Big Oh, Big Omega and Big Theta*. https://www.geeksforgeeks.org/difference-between-big-oh-big-omega-and-big-theta/
[9] Mishra, P. 2022. *5 Outlier Detection Techniques that every "Data Enthusiast" Must Know*. Medium. https://towardsdatascience.com/5-outlier-detection-methods-that-every-data-enthusiast-must-know-f917bf439210
[10] *Non Normal Distribution*. (2021, September 30). Statistics How To. https://www.statisticshowto.com/probability-and-statistics/non-normal-distributions/
[11] *Normal Distributions (Bell Curve): Definition, Word Problems*. (2022, September 25). Statistics How To. https://www.statisticshowto.com/probability-and-statistics/normal-distributions/
[12] *R Introduction*. (n.d.). https://www.w3schools.com/r/r_intro.asp
[13] *Home - RDocumentation*. (n.d.). https://www.rdocumentation.org
[14] GeeksforGeeks. (2020, June 24). *Remove unnecessary values from an Object in R Programming - na.omit() Function*. https://www.geeksforgeeks.org/remove-unnecessary-values-from-an-object-in-r-programming-na-omit-function/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Desember 2020



Cetta Reswara Parahita
13521133