Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

Analisis Perbandingan Kinerja *Binary Search Tree* dan AVL *Tree* dalam Sistem Pencarian Data Mahasiswa

Siti Haliza Zamili[1], Alya Namira[2*], Khodotun Hadawiyah Margolang[3], Adinda Soleha[4], Adidtya Perdana[5]

[1,2,3,45]Fakultas Matematika dan Ilmu Pengetahuan Alam, Program Studi Ilmu Komputer, Universitas Negeri Medan, Kota Medan, Indonesia
Email: [1]sitihalizazamili@gmail.com, [2*]alyanamira3010@gmail.com, [3]khodotunhadawiyah@gmail.com, [4]adindasoleha64@gmail.com, [5]adidtya@unimed.ac.id
(* : corressponding author)

Abstrak−Sistem pencarian informasi adalah bagian penting dalam pengelolaan data, terutama dalam sistem akademis yang menyimpan banyak data mahasiswa. Efektivitas pencarian sangat dipengaruhi oleh bentuk serta pengaturan data. Penelitian ini bertujuan untuk mengevaluasi dan membandingkan performa dari dua jenis struktur data tree, yakni Binary Search Tree dan AVL Tree, dalam kegiatan pencarian data mahasiswa. Pengujian dilakukan dengan menggunakan dataset numerik yang mencerminkan informasi mahasiswa dengan variasi jumlah sebanyak 20, 40, 60, 80 dan 100 data. Parameter yang digunakan dalam penilaian mencakup tinggi pohon dan durasi pencarian data. Implementasi algoritma dilakukan dengan memanfaatkan bahasa pemrograman Python. Hasil dari pengujian menunjukkan bahwa Binary Search Tree cenderung memiliki tinggi pohon yang meningkat dengan signifikan seiring dengan bertambahnya jumlah data, karena tidak adanya mekanisme penyeimbang. Sedangkan, AVL Tree dapat mempertahankan keseimbangan struktur pohonnya melalui proses rotasi yang membuat tinggi pohon menjadi lebih konsisten. Selain itu, durasi pencarian di AVL Tree juga tampak lebih cepat dibandingkan dengan Binary Search Tree, terutama ketika jumlah data yang dihadapi lebih besar. Maka dari itu, AVL Tree dianggap lebih efisien dan ideal untuk diterapkan dalam sistem pencarian data mahasiswa yang memerlukan pencarian yang cepat dan stabil.

Kata Kunci: *Binary Search Tree*, AVL *Tree*, Struktur Data Pohon, Pencarian Data, Sistem Informasi Mahasiswa

Abstract− *Information retrieval systems are a crucial part of data management, particularly in academic systems that store a large amount of student data. Search effectiveness is greatly influenced by the format and organization of the data. This study aims to evaluate and compare the performance of two types of tree data structures, namely Binary Search Trees and AVL Trees, in student data retrieval activities. Testing was conducted using numerical datasets reflecting student information with varying amounts of data: 20, 40, 60, 80, and 100. Parameters used in the assessment included tree height and data search duration. The algorithm was implemented using the Python programming language. The test results show that Binary Search Trees tend to have tree heights that increase significantly with increasing data volume due to the absence of a balancing mechanism. Meanwhile, AVL Trees can maintain the balance of their tree structure through a rotation process that makes the tree height more consistent. In addition, search time in AVL Trees also appears faster than Binary Search Trees, especially when the amount of data encountered is larger. Therefore, AVL Trees are considered more efficient and ideal for implementation in student data retrieval systems that require fast and stable searches.*

Keywords*: Binary Search Tree, AVL Tree, Tree Data Structure, Data Search, Student Information System*

1. PENDAHULUAN

Sistem pencarian data merupakan bagian penting dalam pengelolaan informasi, khususnya pada sistem yang menangani data dalam jumlah besar seperti sistem akademik. Efektivitas proses pencarian sangat dipengaruhi oleh struktur data yang digunakan dalam proses penyimpanan dan pengolahan data. Salah satu struktur yang banyak digunakan adalah *tree* atau struktur pohon, yang merepresentasikan data dalam bentuk hierarki yang terdiri dari node serta hubungan antar node (Pathak & Saxena, 2023).

Dalam bidang ilmu komputer, *Binary Search Tree* (BST) merupakan salah satu struktur data dasar yang digunakan untuk menyimpan data secara terurut sekaligus mendukung operasi pencarian, penyisipan, dan penghapusan secara efisien. Struktur ini memiliki aturan bahwa setiap node memiliki paling banyak dua anak, di mana nilai pada *subtree* kiri lebih kecil dan nilai pada *subtree* kanan lebih besar dibandingkan node induknya (Chizewer dkk., 2025). Dengan mekanisme tersebut,

Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

proses pencarian dapat dilakukan secara bertahap melalui perbandingan nilai pada setiap node yang dilalui.

BST banyak dimanfaatkan dalam berbagai aplikasi, seperti pengurutan data, implementasi *priority queue*, serta struktur data abstrak seperti *set* dan *associative array* (Hirata & Nunome, 2023). Selain itu, BST juga menjadi dasar bagi pengembangan berbagai struktur data lain yang lebih kompleks, termasuk pohon seimbang dan struktur pencarian adaptif. Bahkan, dalam implementasi algoritma modern, BST juga digunakan sebagai dasar dalam perancangan algoritma pencarian dan manipulasi data yang efisien (Lorenzen dkk., 2024).

Namun demikian, performa BST sangat dipengaruhi oleh bentuk struktur pohon yang dihasilkan. Dalam kondisi ideal, BST mampu memberikan kompleksitas operasi sebesar O(log n), tetapi dalam kondisi tertentu—misalnya ketika data dimasukkan secara berurutan—struktur pohon dapat menjadi tidak seimbang dan menyerupai struktur linear, sehingga kompleksitas operasi menurun menjadi O(n) (Chizewer dkk., 2025). Hal ini menunjukkan bahwa BST memiliki kelemahan utama pada aspek keseimbangan struktur.

Selain itu, struktur BST juga sangat bergantung pada urutan penyisipan data, sehingga bentuk pohon yang dihasilkan dapat berbeda meskipun menggunakan data yang sama (Hirata & Nunome, 2023). Ketergantungan ini dapat memengaruhi performa pencarian dan menyebabkan hasil yang tidak konsisten, terutama dalam sistem yang melibatkan proses paralel.

Dalam pengembangan lebih lanjut, berbagai penelitian telah berupaya meningkatkan efisiensi BST melalui modifikasi struktur maupun pendekatan optimasi. Salah satu pendekatan yang dikembangkan adalah struktur berbasis BST yang lebih adaptif dalam merepresentasikan data tertentu, seperti himpunan bilangan dengan interval yang tidak berurutan, sehingga dapat meningkatkan efisiensi penyimpanan dan operasi data (Wang & Barnier, 2024). Selain itu, aspek *invariant* atau aturan dasar dalam BST juga menjadi perhatian penting dalam implementasi yang benar dan konsisten, karena setiap operasi pada BST harus mempertahankan sifat keterurutan data tersebut (Rodriguez dkk., 2025).

Selain dari sisi implementasi, BST juga banyak dikaji dalam penelitian lanjutan, termasuk dalam konteks optimasi berbasis skenario dan analisis performa terhadap berbagai kondisi penggunaan. Dalam konteks ini, BST dipandang sebagai struktur fundamental yang dapat dioptimalkan untuk meningkatkan efisiensi pencarian data dalam berbagai situasi (Angelopoulos dkk., 2024). Bahkan, dalam beberapa kasus, BST juga digunakan sebagai komponen dalam struktur yang lebih kompleks seperti *balanced tree* untuk meningkatkan efisiensi dalam pengolahan multi-objektif (Ren dkk., 2022).

Berdasarkan kelebihan dan keterbatasannya, BST menjadi dasar penting dalam memahami struktur data berbasis pohon sebelum dikembangkan lebih lanjut menjadi struktur yang lebih optimal, seperti *self-balancing tree*. Oleh karena itu, diperlukan analisis lebih lanjut untuk membandingkan performa BST dengan struktur pohon lain yang memiliki mekanisme penyeimbangan, seperti AVL *Tree*, dalam konteks sistem pencarian data.

AVL *Tree* adalah jenis khusus dari *Binary Search Tree* (BST) yang memiliki karakteristik keseimbangan otomatis untuk memastikan tinggi pohon tetap seimbang. Konsep struktur data ini pertama kali diperkenalkan oleh Adelson-Velskii dan Landis. AVL *Tree* memastikan bahwa fungsi dasar seperti pencarian, penyisipan, dan penghapusan dapat dilakukan dalam waktu O(log n) karena pohon selalu dipertahankan dalam keadaan seimbang. Keseimbangan ini dicapai melalui proses penyeimbangan kembali setelah melakukan operasi penyisipan atau penghapusan node dalam pohon. Pada *AVL Tree*, setiap simpul memiliki dua *subtree*, yakni *subtree* kanan dan *subtree* kiri, di mana perbedaan ketinggian keduanya tidak boleh melebihi satu. Selisih tinggi kedua *subtree* ini dicatat dalam faktor keseimbangan yang hanya bisa bernilai -1, 0, atau +1, yang menunjukkan perbedaan antara tinggi *subtree* kiri dan kanan. Apabila setelah melakukan operasi penyisipan atau penghapusan, nilai keseimbangan dari suatu simpul mencapai -2 atau +2, maka pohon perlu disesuaikan kembali dengan melakukan rotasi *subtree* tersebut (Brown, 2025).

Proses penyeimbangan dalam AVL *Tree* dilakukan dengan memanfaatkan rotasi. Ada empat tipe rotasi yang diterapkan, yaitu *Left-Left* (LL) dan *Right-Right* (RR) yang termasuk rotasi tunggal, serta *Left-Right* (LR) dan *Right-Left* (RL) yang merupakan rotasi ganda. Tujuan dari rotasi ini adalah untuk merombak struktur pohon agar keseimbangan tinggi dari *subtree* bisa dikembalikan tanpa melangggar prinsip dasar *Binary Search Tree*. Selain dari proses penyisipan, langkah keseimbangan

Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

juga penting dalam penghapusan node. Pada AVL *Tree*, cara penghapusan dilakukan mirip dengan BST pada awalnya. Jika sebuah node dihapus dan tidak memiliki anak, maka node tersebut akan dihapus secara langsung. Jika node tersebut memiliki satu anak, maka node itu akan digantikan oleh anaknya. Namun, jika node memiliki dua anak, ia akan digantikan oleh *in-order predecessor* (node paling kanan dari *subtree* kiri) atau *in-order successor* (node paling kiri dari *subtree* kanan). Setelah penghapusan dilakukan, pohon akan dievaluasi kembali dan rotasi akan dilakukan jika terjadi ketidakseimbangan (Brown, 2025).

AVL *Tree* juga memanfaatkan faktor keseimbangan untuk menentukan node pengganti yang paling sesuai ketika menghapus node yang memiliki dua anak. Apabila nilai keseimbangan adalah -1, berarti *subtree* sebelah kiri lebih tinggi, sehingga node pengganti diambil dari node paling kanan dalam *subtree* kiri. Sebaliknya, jika nilai keseimbangan adalah +1, maka *subtree* sebelah kanan lebih tinggi, sehingga node pengganti diambil dari node paling kiri pada *subtree* kanan. Metode ini dapat membantu menurunkan kemungkinan terjadinya proses penyeimbangan kembali setelah penghapusan node (Brown, 2025).

Dalam penerapannya, AVL *Tree* memiliki kelebihan dibandingkan dengan BST biasa karena dapat mempertahankan kompleksitas waktu logaritmik untuk operasi utama meskipun data yang dimasukkan dalam bentuk terurut. Pada pohon BST biasa, kinerja dapat menurun secara drastis ketika data yang dimasukkan teratur, sedangkan AVL *Tree* tetap terjaga stabilitasnya karena adanya mekanisme rotasi yang mengontrol agar tinggi pohon tetap seimbang (Yang, 2025).

Selain berfungsi dalam struktur data dasar, AVL *Tree* juga banyak digunakan diberbagai sektor komputasi, contohnya dalam algoritma penempatan media penyimpanan data, proses pencarian, penambahan, dan penghapusan bisa dilaksanakan dengan efektif dengan kompleksitas O(log n), sehingga dapat meningkatkan kinerja algoritma dalam pengelolaan sumber daya komputasi (Rani John dkk., 2023).

2. METODOLOGI PENELITIAN

2.1 Jenis Penelitian

Penelitian ini menerapkan pendekatan eksperimen berbasis komputer untuk mempelajari dan membandingkan efektivitas dua struktur data yang berbasis pohon, yakni *Binary Search Tree* (BST) dan AVL *Tree,* dalam konteks pencarian data mahasiswa. Pendekatan eksperimen berbasis komputer dipilih karena penelitian ini melibatkan penerapan algoritma dalam bentuk perangkat lunak serta pengukuran kinerja berdasarkan hasil pelaksanaan perangkat lunak. Melalui pendekatan ini, pengujian dilakukan dalam kondisi yang terkontrol dengan memanfaatkan dataset yang serupa untuk kedua algoritma, sehingga perbandingan hasil dapat dilakukan secara adil. Variabel yang digunakan dalam pengujian meliputi waktu pencarian, tinggi pohon, dan jumlah perbandingan yang dilakukan.

2.2 Dataset Penelitian

Dataset yang gunakan dalam penelitian ini terdiri dari informasi mahasiswa yang meliputi 100 data contoh. Setiap entri mahasiswa memiliki sejumlah atribut, yaitu:

a. Identitas mahasiswa
b. Nama mahasiswa
c. Jurusan studi

Data tersebut disimpan dalam format file *Excel* selanjutnya diproses menggunakan bahasa pemograman python. Dalam penelitian ini, atribut NIM berfungsi sebagai nilai kunci dalam proses penyisipan dan pencarian dalam struktur pohon.

Untuk meningkatkan keabsahan hasil pengujian, dataset diuji dalam bebera skala, yaitu: 20, 40, 60, 80, 100 perdata. Variasi dataset ini digunakan untuk mengamati dampak distribusi data terhadap performa algoritma.

2.3 Perancangan Sistem

Sistem yang dikembangkan dalam penelitian ini adalah sistem pencarian informasi mahasiswa yang berbasis struktur pohon (*Tree*). Setiap informasi mahasiswa diwakili sebagai node

Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

dalam struktur pohon dengan NIM sebagai key. Proses penyisipan data yang dilakukan pada BST, dengan menambahkan data berdasarkan perbandingan nilai *key* tanpa adanya proses penyeimbangan. Sementara pada AVL *Tree*, penyisipan dilakukan mengikuti ketentuan BST, tetapi dilengkapi dengan mekanisme penyeimbangan melalui rotasi untuk menjaga keseimbangan pohon. Perancangan sistem ini dilakukan agar menjamin bahwa kedua algoritma diuji dalam kondisi yang serupa, sehingga hasil perbandingan yang didapat adalah adil.

2.4 Implementasi Algoritma

Implementasi algoritma dilakukan menggunakan bahasa pemograman Python. Struktur data *Binary Search Tree* (BST) dan AVL *Tree* diterapkan dalam bentuk kelas yang memiliki fungsi utama sebagai berikut:

a. Fungsi untuk menambahkan data *(insert)*
b. Fungsi untuk mencari data *(search)*
c. Fungsi untuk menghitung tinggi pohon *(height)*

Pada AVL *Tree*, ditambahkan fungsi rotasi (rotsi kiri dan rotasi kanan) untuk mempertahankan keseimbangan pohon berdasarkan faktor keseimbangan. Setiap data NIM dalam daraset dimasukkan kedalam kedua struktur data, setelah itu dilakukan pencarian terhadap data tertentu untuk mengevaluasi kinerja algoritma.

2.5 Metode Pengujian

Pengujian dilakukan dengan cara membandingkan kinerja kedua algoritma dalam situasi yang serupa menggunakan dataset yang sama. Tahap pengujian meliputi:

a. Memasukkan ssemua data mahasiswa ke dalam *Binary Search Tree* dan AVL *Tree*
b. Mengukur tinggi dari pohon setelah proses penyisipan data selesai
c. Melakukan pencarian untuk beberapa data yang uji
d. Menghitung waktu pencarian menggunakan fungsi pengukuran waktu pada *Python*
e. Menentukan jumlah langkah atau perbandingan yang terjadi selama proses pencarian

Hasil dari pengujian kemudian ditampilkan dalam dormat tabel dan grafik untuk memudahkan analisis. Analisis dilakukan dengan membandingkan nilai dari setiap parameter untuk menentukan struktur data yang menunjukkan kinerja lebih baik dalam sistem pencarian data mahasiswa.

3. ANALISA DAN PEMBAHASAN

3.1 Parameter Pengujian

Dalam studi ini, dua struktur data; *Binary Search Tree* (BST) dan AVL *Tree*, diperiksa untuk membandingkan proses kerja masing -masing. Pengujian dilakukan dengan menggunakan beberapa variasi kuantitas data untuk menguji dampak ukuran data terhadap kinerja algoritma.

Mengikuti parameter – parameter berikut digunakan dalam pengujian:

1. Jumlah Informasi

Pengujian yang dilakukan telah membawa dengan menggunakan kuantitas data sebagai berikut: 20, 40, menggunakan, 80, dan 100 data. Variasi kuantitas data berikut: 20, 40, 60, 80, dan 100 data.

2. Struktur Data

Struktur yang Digunakan. *Binary Search Tree* (BST) dan AVL *Tree* adalah dua struktur data yang dibandingkan.

3. Tinggi Pohon

Tinggi pohon digunakan digunakan untuk memeriksa beberapa struktur pohon yang terbentuk selama proses analisis data untuk memeriksa beberapa struktur pohon yang terbentuk selama proses analisis data.

4. Waktu Pencarian

Digunakan untuk mempercepat beberapa prosedur pencarian data cepat yang dilakukan pada berbagai struktur data.

Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

5. Pengujian Lingkungan

Implementasi pelaksanaan algoritma dilakukan menggunakan bahasa pemrograman Python dengan bantuan modul pengukuran waktu untuk mempersingkat proses pengumpulan data.

3.2 Dataset Pengujian

Studi ini menggunakan data numerik sebagai kunci dalam struktur *Binary Search Tree* dan AVL *Tree*. Data yang disebutkan di atas dibuat secara acak untuk mensimulasikan proses pemasukan dan pengolahan data dalam sistem.

Data yang digunakan dalam analisis terdiri dari beberapa variasi, termasuk 20, 40, 60, 80, dan 100 data. Tujuan dari variasi ini adalah untuk menguji dampak kuantitas data pada tinggi pohon dan pencarian waktu pada berbagai struktur data.

Berikut ini adalah contoh dataset yang digunakan dalam penelitian.

Tabel 1. Contoh Dataset

NIM	Nomor	Field
220101	Andi Saputra	Informatika
220115	Budi Pratama	Sistem Informasi
220108	Citra Lestari	Informatika
220123	Dimas Prakoso	Teknik Komputer
220105	Eka Putri	Informatika
220119	Farhan Akbar	Sistem Informasi
220110	Gina Maharani	Informatika
220132	Hadi Wijaya	Teknik Komputer
220118	Indah Permata	Sistem Informasi

Data selengkapnya digunakan pada proses pengujian program namun tidak ditampilkan seluruhnya pada tabel untuk menjaga keringkasan penulisan.

3.3 Hasil Pengujian

Dilakukan proses analisa data baik struktur pohon, pengukuran mengenai pohon tinggi dan waktu pencarian data. Diperoleh dengan menjalankan program *Python* yang telah dibuat.

Untuk memudahkan proses analisis dan perbandingan kinerja antara kedua struktur data, hasil pengujian disajikan dalam bentuk tabel.

Tabel 2. Hasil Pengujian

Jumlah Data	Tinggi BST	Tinggi AVL	Waktu BST (ms)	Waktu AVL (ms)
20	7	5	0,18	0,16
40	10	6	0,31	0,23
60	18	7	0,52	0,34
80	26	7	0,73	0,41
100	34	8	0,95	0,5

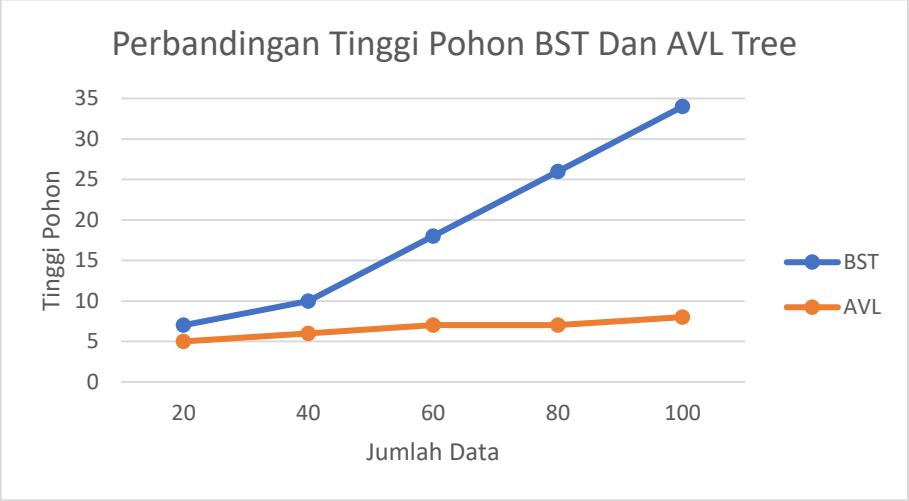
Untuk setiap variasi jumlah data yang digunakan dalam pengujian, Tabel 2 menunjukkan perbandingan tinggi pohon dan waktu pencarian antara pohon AVL dan *Binary Search Tree.*

3.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang diperoleh, dapat dilakukan analisis terhadap performa kedua struktur data yang diuji.

Grafik pertama menunjukkan hubungan antara jumlah data dan tinggi pohon pada *Binary Search Tree* dan AVL *Tree*.

Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

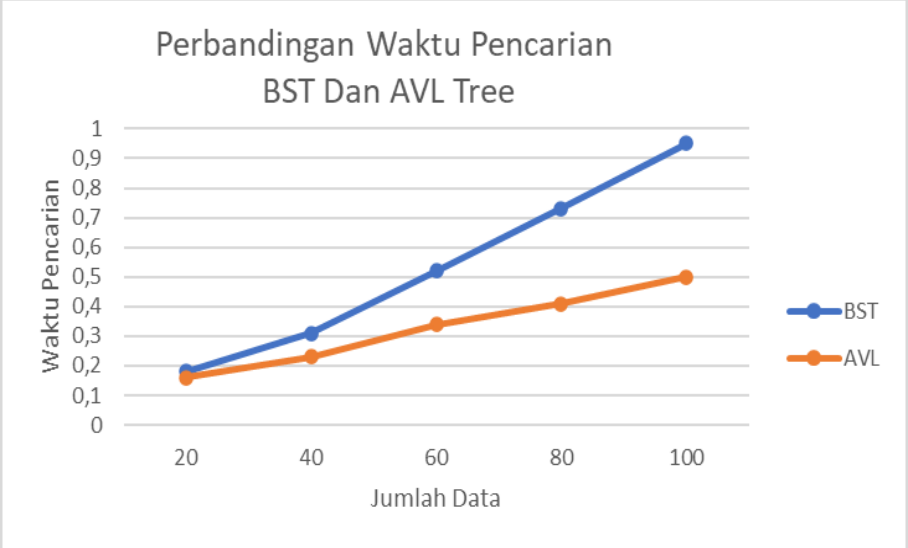


Gambar 1. Grafik Perbandingan Tinggi Pohon BST dan AVL *Tree*

Hubungan antara jumlah data dan tinggi pohon pada *Binary Search Tree* dan AVL *Tree* ditunjukkan pada Grafik 1. Grafik ini menunjukkan bahwa jumlah data meningkatkan tinggi pohon pada *Binary Search Tree* secara signifikan. Hal ini disebabkan oleh fakta bahwa BST tidak memiliki mekanisme penyeimbangan, yang mengakibatkan struktur pohon yang tidak seimbang.

Sebaliknya, proses rotasi, atau penyeimbangan, yang dilakukan secara otomatis oleh pohon saat terjadi ketidakseimbangan strukturnya, memungkinkan AVL *Tree* untuk mempertahankan tinggi pohon yang relatif stabil.

Grafik kedua menunjukkan hubungan antara jumlah data dan waktu pencarian pada kedua struktur data.



Gambar 2. Grafik Perbandingan Waktu Pencarian BST dan AVL *Tree*

Hubungan antara waktu pencarian dan jumlah data pada tanaman pencarian biner dan AVL ditunjukkan pada Grafik 2. Jumlah data yang lebih besar menunjukkan bahwa tanaman pencarian AVL cenderung lebih cepat daripada tanaman pencarian biner.

Hal ini menunjukkan bahwa performa pencarian yang lebih efisien diperoleh dari pohon AVL dengan struktur pohon yang seimbang dibandingkan dengan BST yang tidak memiliki mekanisme penyeimbangan.

Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

4. IMPLEMENTASI

4.1 Implementasi Program

Sistem dikembangkan pada tahap implementasi menggunakan bahasa pemrograman Python. Tujuan pengembangan sistem adalah untuk memungkinkan struktur data AVL *Tree* dan *Binary Search Tree* (BST) digunakan dalam proses pencarian data mahasiswa.

Program terdiri dari beberapa bagian utama yaitu:

1. Struktur dasar kedua algoritma diwakili oleh class node, yang digunakan untuk menyimpan data siswa seperti NIM, nama, dan program studi. Setiap node juga memiliki atribut untuk menunjuk ke anak kiri dan kanan, serta atribut tinggi pohon, yang digunakan khusus pada pohon AVL.

```
class Node:
    def __init__(self, nim, nama, prodi):
        self.nim    = nim
        self.nama   = nama
        self.prodi  = prodi
        self.kiri   = None
        self.kanan  = None
        self.height = 1
```

2. Pada *Binary Search Tree*, data dimasukkan berdasarkan perbandingan nilai NIM tanpa mekanisme penyeimbangan. Data dengan nilai rendah ditempatkan di subtree kiri, dan data dengan nilai tinggi ditempatkan di subtree kanan.

```
def _sisip(self, node, nim, nama, prodi):
    if node is None:
        return Node(nim, nama, prodi)
    if nim < node.nim:
        node.kiri  = self._sisip(node.kiri,  nim, nama, prodi)
    elif nim > node.nim:
        node.kanan = self._sisip(node.kanan, nim, nama, prodi)
    return node
```

3. Untuk menjaga keseimbangan pohon, AVL *Tree* memiliki mekanisme rotasi, yang berbeda dengan BST. Ketika nilai faktor keseimbangan melebihi batas tertentu, pohon diputar.

```
def _rotasi_kanan(self, y):
    x = y.kiri
    y.kiri  = x.kanan
    x.kanan = y
    self._update_height(y)
    self._update_height(x)
    return x

def _rotasi_kiri(self, x):
    y = x.kanan
    x.kanan = y.kiri
    y.kiri  = x
    self._update_height(x)
    self._update_height(y)
    return y
```

Biner : Jurnal Ilmu Komputer , Teknik dan Multimedia
Volume 3, No. 6, Februari 2026
ISSN 2828-2442 (media online)
Hal 652-660

4. Sampai data yang dicari ditemukan atau node kosong ditemukan, proses pencarian data dilakukan dengan menelusuri struktur pohon berdasarkan perbandingan nilai NIM.

```
def _cari(self, node, nim):
    if node is None or node.nim == nim:
        return node
    if nim < node.nim:
        return self._cari(node.kiri,  nim)
    return self._cari(node.kanan, nim)
```

Setelah program dijalankan, sistem akan menghasilkan *output* berupa tinggi pohon serta waktu pencarian untuk masing-masing struktur data.

```
============================================================
  PENGUJIAN KINERJA BST vs AVL TREE - DATA MAHASISWA
============================================================
 Jumlah Data | Tinggi BST | Tinggi AVL | Waktu BST (ms) | Waktu AVL (ms)
------------------------------------------------------------
          20 |          7 |          5 |           0.18 |           0.16
          40 |         10 |          6 |           0.31 |           0.23
          60 |         18 |          7 |           0.52 |           0.34
          80 |         26 |          7 |           0.73 |           0.41
         100 |         34 |          8 |           0.95 |           0.50
============================================================
```

Gambar 3. *Output* Pengujian Kinerja BST vs AVL *Tree*

Hasil implementasi ini, nilai tinggi pohon dan waktu pencarian, digunakan untuk analisis perbandingan kinerja BST dan AVL *Tree*.

5. KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa struktur data memiliki peran penting dalam menentukan efisiensi proses pencarian data, khususnya pada sistem yang menangani data dalam jumlah besar seperti sistem informasi mahasiswa. Dari hasil pengujian yang dilakukan dengan variasi jumlah data, terlihat bahwa performa *Binary Search Tree* (BST) dan AVL *Tree* menunjukkan perbedaan yang cukup signifikan.

BST cenderung mengalami peningkatan tinggi pohon secara drastis seiring bertambahnya jumlah data. Hal ini disebabkan oleh tidak adanya mekanisme penyeimbangan, sehingga struktur pohon dapat menjadi tidak optimal dan menyerupai struktur linear. Akibatnya, waktu pencarian pada BST menjadi kurang efisien, terutama ketika jumlah data semakin besar. Sebaliknya, AVL *Tree* mampu mempertahankan struktur pohon yang lebih seimbang melalui mekanisme rotasi yang dilakukan secara otomatis setelah proses penyisipan maupun penghapusan data. Keseimbangan ini membuat tinggi pohon relatif stabil, sehingga proses pencarian dapat dilakukan dengan lebih cepat dan konsisten dibandingkan BST.

Berdasarkan perbandingan tersebut, dapat disimpulkan bahwa AVL *Tree* memiliki performa yang lebih baik dibandingkan BST dalam hal efisiensi pencarian dan kestabilan struktur, terutama pada pengolahan data dalam jumlah besar. Oleh karena itu, AVL *Tree* lebih direkomendasikan untuk diterapkan pada sistem pencarian data mahasiswa yang membutuhkan kecepatan dan konsistensi dalam proses pencarian.

Untuk penelitian selanjutnya, disarankan untuk melakukan pengujian dengan jumlah data yang lebih besar serta mempertimbangkan penggunaan struktur data lain yang memiliki mekanisme penyeimbangan berbeda, seperti *Red-Black Tree*, guna memperoleh perbandingan performa yang lebih luas.

REFERENCES

Angelopoulos, S., Dürr, C., Elenter, A., & Melidi, G. (2024). Scenario-Based Robust Optimization of Tree Structures. *In Proceedings of the AAAI Conference on Artificial Intelligence*, *39*(25), 26895–26903. https://doi.org/https://doi.org/10.48550/arXiv.2408.11422

Brown, R. A. (2025). *Comparative Performance of the AVL Tree and Three Variants of the Red-Black Tree*. 1–21. https://doi.org/10.1002/spe.3437

Chizewer, J., Melczer, S., Munro, J. I., & Pun, A. (2025). Succinct encodings of binary trees with application to AVL trees. *Theoretical Computer Science*, *1056*. https://doi.org/10.1016/j.tcs.2025.115537

Hirata, H., & Nunome, A. (2023). Performance Evaluation on Parallel Speculation-Based Construction of a Binary Search Tree. *International Journal of Networked and Distributed Computing*, *11*(2), 88–111. https://doi.org/10.1007/s44227-023-00013-w

Lorenzen, A., Leijen, D., Swierstra, W., & Lindley, S. (2024). The Functional Essence of Imperative Binary Search Trees. *Proceedings of the ACM on Programming Languages*, *8*(PLDI), 518–542. https://doi.org/10.1145/3656398

Pathak, A., & Saxena, S. (2023). Tree-Mining: Understanding Applications and Challenges. *Science Insights*, *43*(1), 985–993. https://doi.org/10.15354/si.23.re605

Rani John, R., Grace Mary Kanaga, E., & G, S. S. (2023). International Journal of INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING An AVL Tree Based Algorithm for Virtual Machine Placement Problem. Dalam *Original Research Paper International Journal of Intelligent Systems and Applications in Engineering IJISAE* (Vol. 2024, Nomor 9s). www.ijisae.org

Ren, Z., Zhan, R., Rathinam, S., Likhachev, M., & Choset, H. (2022). Enhanced Multi-Objective A* Using Balanced Binary Search Trees. *Proceedings of the International Symposium on Combinatorial Search*, *15*(1), 162–170. https://doi.org/10.1609/socs.v15i1.21764

Rodriguez, N., Pardo, A., & Viera, M. (2025). Representing Data Structures with Invariants in Haskell: The Cases of BST and AVL. *Proceedings of the 10th ACM SIGPLAN International Workshop on Type-Driven Development*, 26–38. https://doi.org/10.1145/3759538.3759652

Wang, R., & Barnier, N. (2024). Adaptive Binary Search Tree for Integers Sets with Few Holes. *Electrical & Electronic Engineering Research*, *4*(1). https://doi.org/10.37420/j.eeer.2024.002

Yang, Z. (2025). Holistic Performance, Memory, and Energy Analysis of Sorting Algorithms and Binary Search Trees in Python. *Applied and Computational Engineering*, *179*(1), 119–127. https://doi.org/10.54254/2755-2721/2025.ld26932