Optimasi Pencarian Data Santri Menggunakan Algoritma Binary Search Tree di PPM Al-Islam

Siti Badriya[1], Hilyatul Aulia Azzahra[2], Bima Azis Kusuma[3]

[1,2,3] *Teknik Informatika, Sekolah Tinggi Teknologi Indonesia, Cirebon, Jawa Barat, Indonesia*
Email: sitibadrya38@gmail.com

Abstract **—Data management and student data retrieval at Pondok Pesantren Modern Al-Islam are currently conducted manually using spreadsheets, which leads to inefficiencies as the volume of student data increases each year. This study aims to improve the efficiency of student data retrieval by implementing the Binary Search Tree (BST) data structure as an alternative solution. The research methodology includes problem identification, design of a Binary Search Tree structure based on the Student Identification Number (NIS) attribute, and system implementation using the C++ programming language. System testing was carried out by comparing the performance of the Binary Search Tree search process with linear search methods using various student data scenarios. The results show that the Binary Search Tree provides significantly better search efficiency than linear search, with an average time complexity of O(log n) compared to O(n) for conventional methods. In addition, the InOrder traversal of the Binary Search Tree produces sorted student data output, which is beneficial for automating administrative reporting processes. This study concludes that the implementation of Binary Search Tree is effective in improving the efficiency and accuracy of student data retrieval and can serve as a technical foundation for developing a more structured and efficient Islamic boarding school information system.**

Keywords— Binary Search Tree, Data Structure, Search Algorithm, Student Data, Time Complexity

Abstrak **— Pengelolaan dan pencarian data santri pada Pondok Pesantren Modern Al-Islam saat ini masih dilakukan secara manual menggunakan spreadsheet, sehingga proses pencarian menjadi kurang efisien seiring bertambahnya jumlah data setiap tahun. Penelitian ini bertujuan untuk meningkatkan efisiensi pencarian data santri dengan menerapkan struktur data Binary Search Tree (BST). Metodologi penelitian meliputi identifikasi permasalahan, perancangan struktur Binary Search Tree berdasarkan atribut Nomor Induk Santri (NIS), serta implementasi sistem menggunakan bahasa pemrograman C++. Pengujian sistem dilakukan dengan membandingkan kinerja pencarian menggunakan Binary Search Tree dan metode pencarian linear pada berbagai skenario data santri. Hasil penelitian menunjukkan bahwa Binary Search Tree memberikan efisiensi pencarian yang lebih baik dibandingkan metode pencarian linear, dengan kompleksitas waktu rata-rata O(log n) dibandingkan O(n) pada metode konvensional. Selain itu, traversal InOrder pada Binary Search Tree mampu menghasilkan data santri yang tersusun secara terurut, sehingga mendukung otomatisasi proses pelaporan administrasi. Penelitian ini menyimpulkan bahwa penerapan Binary Search Tree efektif dalam meningkatkan efisiensi dan ketepatan pencarian data santri serta layak digunakan sebagai dasar pengembangan sistem informasi pesantren yang lebih terstruktur dan efisien.**

Kata Kunci— Algoritma Pencarian, Binary Search Tree, Data Santri, Kompleksitas Waktu, Struktur Data

I. PENDAHULUAN

Perkembangan teknologi informasi saat ini semakin maju, termasuk dalam hal pengelolaan dan pencarian data. Pencarian (searching) merupakan aktivitas yang sering dilakukan dalam kehidupan sehari-hari, seperti mencari kata pada text editor atau menemukan data tertentu dalam sebuah dokumen. Dalam ilmu komputer, dikenal beberapa metode pencarian, di antaranya Sequential Search, Binary Search, Interpolation Search, Direct Search, dan Hash Search. Setiap metode memiliki kelebihan serta kekurangan masing-masing.

Penggunaan komputer dalam pengolahan data kini menjadi kebutuhan penting demi efisiensi dan ketepatan informasi. Meskipun banyak aplikasi siap pakai seperti Microsoft Office atau OpenOffice yang mampu mengolah data dalam jumlah besar, kebutuhan tertentu seringkali memerlukan sistem yang lebih khusus, terutama dalam proses pencarian yang cepat dan akurat [1].

Pada Pondok Pesantren Modern Al-Islam, proses pencatatan dan pencarian data santri masih dilakukan secara manual atau menggunakan spreadsheet sederhana. Ketika jumlah santri bertambah, proses pencarian menjadi semakin lambat dan tidak efisien. Hal ini menunjukkan perlunya penerapan metode pencarian yang lebih optimal.

Binary Search adalah salah satu metode pencarian yang memiliki efisiensi tinggi, namun metode ini hanya dapat bekerja jika data telah terurut. Untuk konteks

Submit: 20/01/2026 *Revisi: 22/01/2026* *Accepted: 25/01/2026*

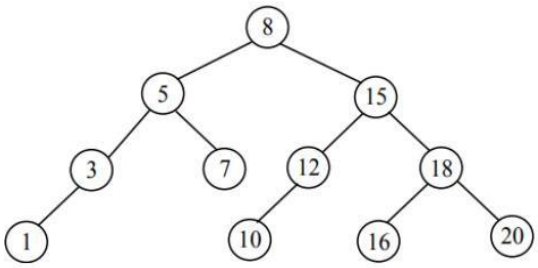
penyimpanan dan pencarian data yang lebih dinamis, struktur data *Binary Search Tree* menjadi solusi yang lebih tepat. *Binary Search Tree* merupakan bentuk terurut dari *Binary Tree* (Pohon Biner), di mana setiap simpul memiliki maksimal dua anak, yaitu anak kiri *(left child)* dan anak kanan *(right child)*. Penempatan data dalam *Binary Search Tree* ditentukan berdasarkan perbandingan nilai dengan simpul induk, sehingga proses pencarian, penyisipan, dan penghapusan dapat dilakukan dengan cepat [2], [3].

Dengan menerapkan Binary Search Tree pada pencarian data santri, sistem informasi pesantren dapat bekerja lebih efisien dan akurat. Implementasi Binary Search Tree diharapkan dapat mengatasi kendala pencarian data yang selama ini terjadi, serta menjadi pondasi penting dalam pengembangan sistem informasi Pondok Pesantren Modern Al-Islam ke depannya.

II. TINJAUAN PUSTAKA

A. Pohon Pencarian Biner (Binary Search Tree)

Binary Search Tree merupakan struktur data pohon biner yang memiliki karakteristik khusus karena memungkinkan proses pencarian, penambahan, dan penghapusan data dilakukan secara efisien. Setiap node dalam Binary Search Tree memiliki nilai lebih besar daripada semua node pada subpohon kiri dan lebih kecil daripada semua node pada subpohon kanan. Struktur ini membuat Binary Search Tree sangat sesuai untuk pengelolaan data yang terus bertambah, seperti data santri di Pondok Pesantren Modern Al-Islam

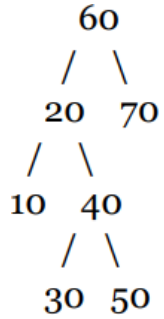


Gambar 1. Contoh Binary Search Tree

Operasi utama pada *Binary Search Tree* terdiri dari penyisipan, pencarian, dan penghapusan. Penyisipan data dimulai dari simpul akar. Jika pohon kosong, data baru menjadi simpul akar. Jika pohon tidak kosong, nilai data dibandingkan dengan node saat ini dan ditempatkan pada subpohon kiri jika nilainya lebih kecil atau pada subpohon kanan jika nilainya lebih besar. Penyisipan biasanya dilakukan secara rekursif, sehingga memudahkan implementasi program komputer. Pencarian data dapat dilakukan secara rekursif maupun iteratif, dengan membandingkan nilai node dengan kunci yang dicari dan menelusuri subpohon kiri atau kanan sesuai aturan *Binary Search Tree* hingga data ditemukan atau tidak ada node yang dapat dikunjungi. Penghapusan node memerlukan penyesuaian struktur pohon agar aturan *Binary Search Tree* tetap terpenuhi; jika node yang dihapus memiliki anak, posisinya digantikan oleh node yang sesuai dari subpohon terkait.

Traversal digunakan untuk mengunjungi semua node dalam pohon. Traversal *PreOrder* mengunjungi node saat ini terlebih dahulu, kemudian anak kiri dan anak kanan. Traversal *InOrder* mengunjungi anak kiri, node saat ini, dan anak kanan; metode ini menghasilkan urutan data yang terurut dari kecil ke besar. Traversal *PostOrder* mengunjungi anak kiri, kemudian anak kanan, dan terakhir node saat ini, sedangkan traversal *LevelOrder* mengunjungi node per level dari atas ke bawah dan dari kiri ke kanan.

Sebagai contoh, pertimbangkan *Binary Search Tree* sederhana yang menyimpan data santri dengan atribut NIS dan nama. Struktur pohonnya dapat digambarkan sebagai berikut:



Gambar 2. Binary Search Tree Diagram

Jika dilakukan traversal *InOrder*, urutan NIS yang dihasilkan adalah 10, 20, 30, 40, 50, 60, 70. Traversal *PreOrder* menghasilkan urutan 60, 20, 10, 40, 30, 50, 70, sedangkan *PostOrder* menghasilkan 10, 30, 50, 40, 20, 70, 60. Traversal *LevelOrder* menampilkan 60, 20, 70, 10, 40. 30, 50. Contoh ini menunjukkan bagaimana BST memungkinkan penelusuran data yang terstruktur dan efisien.

Selain itu, *Binary Search Tree* dapat digunakan untuk merepresentasikan ekspresi matematika atau logika. Notasi yang umum digunakan adalah prefix, infix, dan postfix. Notasi prefix menempatkan operator sebelum operand, notasi infix menempatkan operator di antara operand, dan notasi postfix menempatkan operator setelah operand. Representasi ini memanfaatkan konsep traversal pohon dan menjadi dasar evaluasi ekspresi dalam berbagai aplikasi komputasi [6]

B. Kompleksitas Waktu BST (Binary Search Tree)

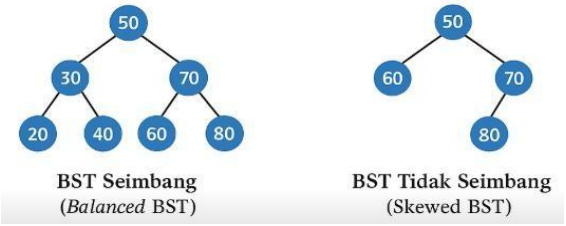
Binary Search Tree (BST) adalah struktur data hierarkis yang memungkinkan operasi pencarian, penyisipan, dan penghapusan dilakukan secara efisien. Kompleksitas setiap operasi bergantung pada tinggi pohon h, dengan n menyatakan jumlah node dalam pohon. Pada *Binary Search Tree* yang seimbang, tinggi pohon kira-kira setara dengan logaritma basis dua dari jumlah node, sehingga operasi pencarian, penyisipan, dan penghapusan dapat dilakukan dalam waktu rata-rata *O(log n)*. Sebaliknya, jika pohon

tidak seimbang, misalnya node dimasukkan secara berurutan, tinggi pohon dapat mencapai n, sehingga kompleksitas operasi menurun menjadi *O(n).*

Traversal pada *Binary Search Tree*, termasuk *PreOrder, InOrder, PostOrder,* dan *LevelOrder*, mengunjungi setiap node tepat satu kali. Oleh karena itu, kompleksitas traversal tetap *O(n)* baik pada pohon seimbang maupun tidak seimbang [2], [7], [9]. Perbedaan ini menggambarkan bagaimana struktur pohon mempengaruhi efisiensi operasional. *Binary Search Tree* yang seimbang memiliki hampir semua level terisi penuh sehingga proses pencarian atau penyisipan memerlukan jumlah langkah minimal. Sebaliknya, pada *Binary Search Tree* tidak seimbang, node tersusun dalam satu garis lurus sehingga operasi memerlukan langkah sebanyak jumlah node, yang jauh lebih lambat [2], [3].

. Tabel 1. Kompleksitas Waktu Bst

Operasi	Rata-rata (Balanced)	Terburuk (Skewed)
Pencarian (Search)	O(log n)	O(n)
Penyisipan (Insert)	O(log n)	O(n)
Penghapusan (Delete)	O(log n)	O(n)
Traversal (Pre/In/Post/ Level)	O(n)	O(n)



Gambar 3. Diagram Ilustratif

Pohon *Binary Search Tree* seimbang ini memiliki tinggi pohon $h \approx log_2(n)$, sehingga operasi pencarian, penyisipan, dan penghapusan dapat dilakukan dalam waktu *O(log n).* Sedangkan Pada pohon *Binary Search Tree* tidak seimbang, tinggi pohon h = n, sehingga operasi utama menjadi linear, yaitu *O(n).* Traversal tetap *O(n)* karena setiap node tetap dikunjungi sekali [2], [3].

C. Relevansi BST (Binary Search Tree) untuk Sistem Informasi

Penerapan struktur data *Binary Search Tree* memiliki peran yang sangat penting dalam pengembangan sistem informasi data santri di Pondok Pesantren Modern Al-Islam. *Binary Search Tree* adalah solusi yang efektif untuk mengatasi masalah pencarian data yang masih dilakukan secara manual atau menggunakan *spreadsheet* sederhana, di mana prosesnya menjadi lambat *(inefisien)* seiring bertambahnya jumlah santri.

1) Kecepatan Pencarian yang Optimal

Keunggulan utama *Binary Search Tree* terletak pada kecepatannya dalam menemukan data. Berbeda dengan pencarian sekuensial pada daftar biasa yang harus menelusuri data satu per satu *(O(n)), Binary Search Tree* dirancang untuk memangkas ruang pencarian di setiap langkahnya.

Pada *Binary Search Tree* yang tersusun baik, pencarian data santri berdasarkan Nomor Induk Santri (NIS) rata-rata hanya membutuhkan waktu logaritmik, yaitu *O(log n)* [2], [3]. Efeknya sangat signifikan: jika jumlah santri mencapai ribuan, *Binary Search Tree* dapat menemukan data hanya dalam sedikit langkah saja. Efisiensi waktu ini sangat krusial untuk meningkatkan ketepatan dan kecepatan pelayanan administrasi [1].

2) Penanganan Data yang Terus Bertambah (Dinamis)

Data santri selalu bersifat dinamis, bertambah setiap tahun ajaran baru. *Binary Search Tree* mampu mengelola perubahan data ini tanpa mengganggu kinerja sistem secara keseluruhan.

Ketika data santri baru dimasukkan, *Binary Search Tree* dapat menyisipkan (insert) data tersebut ke posisi yang tepat dalam waktu rata-rata *O(log n)* [3].

Struktur ini memastikan bahwa penambahan data dapat dilakukan dengan cepat tanpa perlu melakukan proses pengurutan ulang seluruh daftar, yang merupakan kendala pada metode pencarian biner pada array tradisional [4].

3) Otomatisasi Laporan Data yang Terurut

Binary Search Tree secara alami menyimpan data santri dalam urutan yang teratur berdasarkan kunci NIS. Dengan menggunakan metode Traversal *InOrder*, sistem dapat menghasilkan daftar santri yang sudah terurut rapi dari NIS terkecil hingga terbesar secara otomatis.

Proses traversal ini mengunjungi semua data secara efisien (O(n)) [6]. Hal ini memudahkan staf dalam pembuatan laporan atau daftar resmi tanpa memerlukan proses pengurutan tambahan.

4) Pondasi Pengembangan Sistem Jangka Panjang

Implementasi *Binary Search Tree* merupakan fondasi yang baik untuk sistem informasi yang akan dikembangkan lebih lanjut. Meskipun Binary Search Tree sederhana memiliki risiko kinerja menurun (worst-case) jika datanya dimasukkan secara tidak teratur, konsep dasarnya memungkinkan transisi ke struktur yang lebih canggih (seperti Self-Balancing Tree).

Struktur lanjutan ini dapat menjamin bahwa waktu pencarian akan selalu stabil pada *O(log n)*, yang sangat penting untuk memastikan sistem informasi pesantren tetap efisien dan andal di masa depan [2].

D. Penelitian Terkait

Penelitian mengenai optimasi pencarian data telah banyak dilakukan. Implementasi teknik *Binary Search Tree* (BST) pada pencarian data penduduk menunjukkan hasil yang lebih efektif dibandingkan pencarian sekuensial [1]. Selain itu, penggunaan *Binary Search Tree* dalam sistem informasi sekolah terbukti membantu pengorganisasian data dinamis yang terus bertambah [6]. Penelitian lain juga menekankan pentingnya struktur data *non-linear* dalam merepresentasikan hierarki data secara optimal [5].

Meskipun beberapa penelitian sebelumnya telah menerapkan Binary Search Tree dalam sistem kependudukan dan inventaris [1], perbedaan mendasar penelitian ini dengan penelitian terdahulu terletak pada kontekstualisasi struktur data terhadap nomor induk santri (NIS) yang bersifat statis dan unik di lingkungan pesantren. Selain itu, penelitian ini mengintegrasikan InOrder Traversal tidak hanya untuk pencarian, tetapi sebagai mekanisme otomatisasi pengurutan data untuk laporan bulanan akademik di PPM Al-Islam, yang sebelumnya tidak dieksplorasi secara mendalam pada studi kasus serupa

III. METODOLOGI PENELITIAN

A. Gambaran Umum Sistem dan Permasalahan

Pada sistem administrasi Pondok Pesantren Modern Al- Islam, proses pencatatan dan pencarian data santri sampai sekarang masih dilakukan secara manual menggunakan *spreadsheet* seperti Microsoft Excel. Setiap kali dibutuhkan informasi tertentu, pencarian Nomor Induk Santri (NIS), pengecekan kelas, atau data kamar santri, petugas harus menelusuri data satu per satu secara linear. Proses ini membutuhkan waktu yang relatif lama dan berpotensi menimbulkan kesalahan, terutama ketika jumlah data santri terus meningkat setiap tahun.

Berdasarkan permasalahan tersebut, pada penelitian ini dipilih struktur data *Binary Search Tree* sebagai solusi pencarian data santri [1], [2]. *Binary Search Tree* dipilih karena mampu menyederhanakan proses penelusuran data berdasarkan kunci tertentu, yaitu NIS, serta memungkinkan proses pencarian, penyisipan, dan penghapusan data dilakukan secara lebih efisien dibandingkan pencarian linear.

B. Jenis dan Pendekatan Penelitian

Penelitian ini menggunakan pendekatan deskriptif dan eksperimental. Pendekatan deskriptif digunakan untuk menggambarkan kondisi sistem pencarian data santri yang berjalan saat ini serta karakteristik data yang digunakan. Sementara itu, pendekatan eksperimental digunakan untuk menguji kinerja struktur data *Binary Search Tree* dalam melakukan pencarian data santri dan membandingkannya dengan metode pencarian linear.

Data yang digunakan dalam penelitian ini merupakan data santri Pondok Pesantren Modern Al-Islam. Teknik pengumpulan data dilakukan dengan cara studi dokumentasi, yaitu mengambil data yang telah tersedia pada sistem administrasi pesantren dalam bentuk *spreadsheet*.

Atribut data santri yang digunakan dalam penelitian ini meliputi:

1) Nomor Induk Santri (NIS)
2) Nama Santri
3) Kelas
4) Asal Daerah (Domisili)
5) Angkatan

Dari atribut tersebut, NIS digunakan sebagai kunci utama (key) dalam proses pembentukan dan pencarian pada struktur *Binary Search Tree.*

C. Perancangan Struktur Binary Search Tree

Pada tahap ini dilakukan perancangan struktur data *Binary Search Tree* untuk merepresentasikan data santri. Setiap santri dimodelkan sebagai sebuah node dalam *Binary Search Tree* yang memiliki beberapa atribut data, dengan NIS sebagai nilai kunci.

Aturan penyusunan node dalam *Binary Search Tree* mengikuti prinsip sebagai berikut [2], [3]:

1) Jika nilai NIS santri lebih kecil dari NIS pada node induk, maka node ditempatkan pada subpohon kiri.
2) Jika nilai NIS santri lebih besar dari NIS pada node induk, maka node ditempatkan pada subpohon kanan.

Struktur ini memastikan bahwa data santri tersimpan secara terurut berdasarkan NIS dan memungkinkan proses pencarian dilakukan dengan membagi ruang pencarian pada setiap langkah.

D. Implementasi Program

Implementasi struktur data *Binary Search Tree* pada penelitian ini dilakukan menggunakan bahasa pemrograman C++ [2], [3]. Pemilihan bahasa C++ didasarkan pada kemampuannya dalam mendukung pemrograman terstruktur dan manipulasi struktur data secara efisien.

Fungsi-fungsi utama yang diimplementasikan dalam program meliputi:

1) Fungsi *insertNode*, digunakan untuk menambahkan data santri baru ke dalam *Binary Search Tree* sesuai dengan aturan penyusunan *Binary Search Tree*.
2) Fungsi *searchNode*, digunakan untuk mencari data santri berdasarkan NIS tertentu.
3) Fungsi *displayInOrder***,** digunakan untuk menampilkan seluruh data santri menggunakan *traversal InOrder* sehingga data ditampilkan dalam urutan NIS dari yang terkecil hingga terbesar.

Implementasi fungsi-fungsi tersebut mencerminkan konsep dasar operasi pada *Binary Search Tree*, yaitu penyisipan, pencarian, dan traversal.

E. Teknik Pengujian Sistem

Pengujian sistem dilakukan untuk mengetahui kinerja metode *Binary Search Tree* dalam pencarian data santri dengan membandingkannya terhadap pencarian linear sebagaimana dijelaskan pada literatur algoritma pencarian [7], [8], [10]. Pengujian dilakukan dengan menggunakan data santri yang telah dikumpulkan dan dimasukkan ke dalam struktur *Binary Search Tree*. Tahapan pengujian meliputi:

1) Menguji proses pencarian data santri menggunakan Binary Search Tree berdasarkan NIS tertentu.
2) Menguji proses pencarian data santri menggunakan metode pencarian linear pada daftar data.
3) Membandingkan waktu pencarian antara metode *Binary Search Tree* dan pencarian linear.

Hasil pengujian dianalisis untuk melihat perbedaan efisiensi waktu pencarian antara kedua metode tersebut, sehingga dapat diketahui sejauh mana penerapan Binary Search Tree memberikan peningkatan kinerja dibandingkan metode pencarian konvensional.

F. Alur Penelitian

Secara umum, alur penelitian ini dapat dijelaskan sebagai berikut:

a) Identifikasi permasalahan pencarian data santri pada sistem administrasi pesantren.
b) Studi literatur terkait algoritma pencarian, *Binary Search Tree*, dan kompleksitas waktu.
c) Pengumpulan data santri.
d) Perancangan struktur *Binary Search Tree*.
e) Implementasi *Binary Search Tree* dalam bentuk program.
f) Pengujian dan analisis hasil pencarian data.
g) Penarikan kesimpulan berdasarkan hasil penelitian.

Dengan tahapan tersebut, penelitian ini diharapkan mampu menunjukkan bahwa penerapan *Binary Search Tree* dapat meningkatkan efisiensi pencarian data santri pada sistem informasi Pondok Pesantren Modern Al-Islam, sejalan dengan hasil penelitian dan teori yang telah dikemukakan pada literatur sebelumnya [1], [2], [3].

IV. HASIL DAN PEMBAHASAN

Bab ini membahas hasil demonstrasi implementasi serta analisis kinerja struktur data *Binary Search Tree* yang digunakan untuk pencarian data santri. Pembahasan difokuskan pada pembuktian efisiensi waktu pencarian menggunakan *Binary Search Tree* dengan kompleksitas rata-rata *O(log n),* dibandingkan dengan metode pencarian linear yang memiliki kompleksitas *O(n)* dan sebelumnya digunakan dalam pengelolaan data santri.

A. Perhitungan dan Analisis Manual (Demonstrasi Proses Pencarian)

Untuk membuktikan efisiensi *Binary Search Tree*, dilakukan demonstrasi pencarian data secara manual pada model pohon yang dibentuk dari data santri fiktif dengan urutan NIS:

{45, 20, 60, 10, 30, 50, 70}

Data tersebut disisipkan ke dalam *Binary Search Tree* menggunakan aturan penyisipan *Binary Search Tree*, yaitu nilai yang lebih kecil ditempatkan pada subpohon kiri dan nilai yang besar ditempatkan pada subpohon kanan.

1) Skenario pencarian

data ditemukan (NIS = 50)

Tujuan: mencari data santri dengan kunci NIS 50.

Proses pencarian dilakukan dengan mengikuti aturan *Binary Search Tree*, sehingga ruang pencarian dipersempit pada setiap langkah.

Tabel 2. Pencarian Data Ditemukan

Langkah ke-	NIS Node Saat Ini	Perbandingan	Tindak Lanjut
1	45 (Root)	50 > 45	Lanjut ke subpohon kanan
2	60	50 < 60	Lanjut ke subpohon kiri
3	50	50 = 50	Data ditemukan

Analisis:

Data santri dengan NIS 50 berhasil ditemukan hanya dalam 3 langkah pencarian. Hal ini menunjukkan bahwa *Binary Search Tree* mampu mengurangi jumlah langkah pencarian secara signifikan dibandingkan pencarian linear.

2) Skenario pencarian

data tidak ditemukan (NIS = 35)

Tujuan: mencari data santri dengan kunci NIS 35, yang tidak terdapat dalam *Binary Search Tree*.

Tabel 3. Pencarian Data Tidak Ditemukan

Langkah ke-	NIS Node Saat Ini	Perbandingan	Tindak Lanjut
1	45 (Root)	35 < 45	Lanjut ke subpohon kiri
2	20	35 > 20	Lanjut ke subpohon kanan
3	30	35 > 30	Lanjut ke subpohon kanan
4	NULL	-	Pencarian dihentikan

Analisis: pencarian dihentikan setelah mencapai node kosong (NULL pointer). Proses ini membutuhkan 4 langkah, yang setara dengan tinggi pohon, dan menunjukkan batas terburuk pencarian *Binary Search Tree.*

3) Perbandingan kinerja pencarian

Berdasarkan kedua skenario di atas,dapat dianalisis perbedaan kinerja sebagai berikut:

a) *Binary Search Tree (BST)*

Dengan jumlah data n = 7, pencarian dapat diselesaikan dalam 3 - 4 langkah, sesuai dengan kompleksitas rata-rata:

$$O(\log_2 n), \log_2 7 \approx 2{,}8$$

b) *Pencarian Linear (Sekuensial)*

Pada struktur data linear, pencarian NIS 50 dapat membutuhkan hingga 6 - 7 langkah, baik pada saat data terurut maupun tidak terurut, dengan kompleksitas:

$$O(n)$$

Perhitungan manual ini membuktikan secara logis bahwa *Binary Search Tree* memberikan efisiensi pencarian yang lebih baik dibandingkan metode pencarian linear, bahkan pada jumlah data yang relatif kecil.

B. Implementasi Program (Demonstrasi Fungsi BST)

Implementasi struktur data *Binary Search Tree* dilakukan menggunakan bahasa pemrograman C++ sesuai dengan metodologi penelitian. Program ini merepresentasikan data santri dengan NIS sebagai kunci utama dan mendukung operasi penyisipan, pencarian, serta penampilan data secara terurut.

1) Struktur Node *Binary Search Tree*

Struktur data santri direpresentasikan dalam sebuah node *Binary Search Tree* yang memuat atribut NIS, nama, kelas, asal daerah, dan angkatan, serta pointer ke anak kiri dan kanan.

```
struct Santri
    { int nis;
    string nama;
    string kelas;
    string asal;
    int angkatan;
    Santri* left;
    Santri* right;
};
```

Kode 1. Struktur Node Binary Search Tree

Penjelasan:
Struktur Santri digunakan sebagai node pada *Binary Search Tree*, dengan atribut nis sebagai kunci utama pencarian. Pointer left dan right digunakan untuk membentuk hubungan hierarkis antar node sesuai aturan *Binary Search Tree*.

2) Implementasi Fungsi *insertNode()*

Fungsi *insertNode()* digunakan untuk menambahkan data santri ke dalam *Binary Search Tree* berdasarkan nilai NIS.

```
Santri* insertNode(Santri* root, Santri*
newNode) {
   if (root == NULL)
       return newNode;
   if (newNode->nis < root->nis)
root->left     =     insertNode(root->left,
newNode);
else
root->right    =    insertNode(root->right,
newNode);
   return root;
}
```

Kode 2. Fungsi Insert Node

Penjelasan:
Proses penyisipan dilakukan secara rekursif dengan membandingkan nilai NIS. Jika nilai lebih kecil, node ditempatkan di subpohon kiri, sedangkan nilai yang lebih besar ditempatkan di subpohon kanan.

3) Implementasi Fungsi *searchNode()*

Fungsi *searchNode()* digunakan untuk mencari data santri berdasarkan NIS.

```
Santri* searchNode(Santri* root, int nis)
{
    if (root == NULL || root->nis == nis)
       return root;
    if (nis < root->nis)
       return searchNode(root->left, nis);
    return searchNode(root->right, nis);
}
```

Kode 3. Fungsi Search Node

Penjelasan :

Fungsi ini mempersempit ruang pencarian pada setiap langkah, sehingga jumlah perbandingan menjadi lebih sedikit dibandingkan pencarian linear.

4) Demonstrasi penyisipan dan *traversalInOrder*

Ketika data NIS:

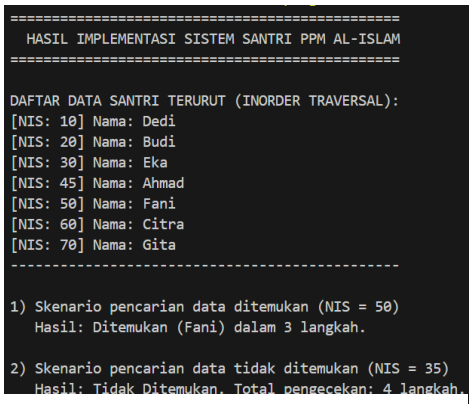
{45, 20, 60, 10, 30, 50, 70}

Disisipkan ke dalam *Binary Search Tree* menggunakan fungsi *insertNode(),* kemudian ditampilkan menggunakan fungsi *displayInOrder(),* maka diperoleh hasil traversal sebagai berikut:

Kunjungan Node	**Urutan Output**
InOrder Traversal	10 → 20 → 30 → 45 → 50 → 60 → 70

Analisis:
Traversal InOrder selalu menghasilkan data dalam keadaan terurut dari nilai terkecil hingga terbesar. Hal ini membuktikan bahwa *Binary Search Tree* tidak hanya efektif untuk pencarian, tetapi juga berguna untuk penyajian data santri secara terurut, misalnya untuk pembuatan laporan administrasi [6].

C. Hasil Output Program



Gambar 4. Output Program

Gambar 4 menunjukkan *output* dari implementasi program, di mana data ditampilkan secara terurut melalui fungsi *InOrder Traversal*. Selain itu, ditunjukkan hasil pengujian dua skenario pencarian yang membuktikan kecepatan akses data menggunakan metode *Binary Search Tree*.

D. Demonstrasi pencarian menggunakan program

Ketika fungsi *searchNode()* dijalankan dengan input NIS 50, program akan mengeksekusi proses perbandingan dan navigasi node sesuai aturan *Binary Search Tree*. Jika data ditemukan, sistem menampilkan informasi lengkap santri yang terkait dengan NIS tersebut, seperti nama, kelas, asal daerah, dan angkatan.

E. Hasil Akhir dan Analisis Kinerja

Implementasi Binary Search Tree berhasil mengubah pengelolaan data santri dari struktur linear menjadi struktur pohon yang terindeks secara hierarkis.

Tabel 4. Hasil Pencarian Spreadsheet & BST

Operasi	**Pencarian Sekuensial (Spreadsheet)**	**Binary Search Tree (BST)**
Pencarian (Rata-rata)	O(n)	O(log n)
Pencarian (Terburuk)	O(n)	O(n) (pohon miring)
Penyisipan Data Baru	Manual / pengurutan ulang	O(log n)

Analisis:
Hasil implementasi dan perhitungan menunjukkan bahwa *Binary Search Tree* memberikan peningkatan kinerja yang signifikan dalam pencarian data santri [1], [2]. Meskipun pada kondisi terburuk *Binary Search Tree* dapat memiliki kompleksitas *O(n),* kompleksitas rata-rata *O(log n)* sangat menguntungkan.

Sebagai ilustrasi, pada 10.000 data santri, pencarian linear dapat membutuhkan hingga 10.000 langkah, sedangkan pencarian menggunakan *Binary Search Tree* rata-rata hanya memerlukan sekitar 14 langkah. Hal ini membuktikan bahwa penerapan *Binary Search Tree* berhasil menjawab permasalahan pencarian data yang lambat dan tidak efisien di Pondok Pesantren Modern Al-Islam.

V. KESIMPULAN

Berdasarkan hasil perancangan, implementasi, dan pengujian yang telah dilakukan, dapat disimpulkan bahwa penerapan struktur data *Binary Search Tree* mampu meningkatkan efisiensi pencarian data santri pada Sistem Informasi Pondok Pesantren Modern Al-Islam. Proses pencarian data yang sebelumnya dilakukan secara linear menggunakan spreadsheet dapat dioptimalkan melalui pemanfaatan struktur Binary Search Tree.

Hasil pengujian menunjukkan bahwa pencarian menggunakan *Binary Search Tree* memiliki kompleksitas waktu rata-rata O(log n), sedangkan pencarian linear memiliki kompleksitas O(n). Dengan menerapkan prinsip pembagian ruang pencarian secara hierarkis dan logika perbandingan (if–then), *Binary Search Tree* mampu mengurangi jumlah langkah pencarian secara signifikan, terutama ketika jumlah data santri semakin besar. Selain itu, *traversal InOrder* pada *Binary Search Tree* menghasilkan data santri yang tersusun secara terurut berdasarkan Nomor Induk Santri (NIS), sehingga memudahkan proses penyajian dan pengelolaan data. Dengan demikian, penerapan konsep logika matematika dan algoritma melalui struktur data *Binary Search Tree* terbukti efektif dalam meningkatkan kinerja pencarian data santri. Struktur Binary Search Tree dapat dijadikan solusi yang tepat untuk mendukung sistem informasi pesantren agar lebih efisien, terstruktur, dan mudah dikembangkan.

Sebagai pengembangan selanjutnya, penelitian ini dapat diperluas dengan menggunakan struktur pohon seimbang, seperti *AVL Tree* atau *Red-Black Tree*, serta mengintegrasikan sistem dengan basis data agar kinerja tetap optimal pada jumlah data yang lebih besar.

DAFTAR PUSTAKA

[1] Malau, E. P., & Sirait, H. P. (2021). "Implementasi teknik binary search tree pada pencarian data penduduk". *Jurnal KAKIFIKOM*.
[2] R. R. Sedgewick, Algorithms, 4th ed. Boston, MA, USA: Addison-Wesley, 2011.
[3] R. Munir, Struktur Data. Bandung, Indonesia: Informatika, 2020.
[4] Politeknik Elektronika Negeri Surabaya (PENS), "Data structure – Bab 8: Pencarian," Modul ajar, 2025.
[5] Politeknik Elektronika Negeri Surabaya, "Tree (pohon)," Modul ajar. [Online]. Available: https://repository.unikom.ac.id/46090/1/TREE.pdf
[6] S. H. N. Ginting et al., Pengantar Struktur Data, Cet. 1. Deli Serdang, Indonesia: PT Mifandi Mandiri Digital, 2023.
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
[8] A. M. Tenenbaum, Y. Langsam, and M. J. Augenstein, Data Structures Using C and C++. Upper Saddle River, NJ, USA: Prentice Hall, 1990.
[9] M. A. Weiss, Data Structures and Algorithm Analysis in C++, 4th ed. Boston, MA, USA: Pearson, 2014.
[10] S. Sahni, Data Structures, Algorithms, and Applications in C++, 2nd ed. New York, NY, USA: McGraw-Hill, 2005.