Jejakdigital
JURNAL ILMIAH MULTIDISIPLIN
Available online at https://indojurnal.com/index.php/jejakdigital

eISSN 3089-7734; pISSN 3089-7742
Vol. 2, No. 4, Tahun 2026
doi.org/10.63822/wbzdgx50
Hal. 5267-5276

Studi Komparatif Efisiensi Memori antara Struktur Data *Array* dan *Linked List*

Ummi Rufaidah Hamka[1*], Nurul Syafirah[2], Muh. Aprian Naufal[3], Asipa Febriana[4], Nurjannah[5], Nurpadila[6], Ketrin Rinayanti Manullang[7]
[1,2,3,4,5,6,7]Program Studi Pendidikan Teknologi Informasi, Universitas Sulawesi Barat

[*]Email Korespodensi: *ummirufaidah871@gmail.com*

Sejarah Artikel:

Diterima 29-05-2026
Disetujui 06-06-2026
Diterbitkan 08-06-2026

ABSTRACT

The selection of linear data structures directly impacts memory efficiency and software performance. This study conducts a profound comparative analysis between arrays and linked lists using a quantitative experimental approach. Benchmarking was performed across sequential access, random access, insertion, and deletion scenarios using C++ and Python with dataset sizes from 103 to 107 elements. Memory consumption was tracked via Valgrind Massif and the tracemalloc library. In C++, arrays consistently outperform linked lists in sequential access with a 3.2× speedup on 106 elements due to optimal CPU cache locality. Conversely, for middle insertions, linked lists show a dramatic advantage with a constant time of ~50ns, compared to 2.3ms for arrays (46,000× slower). Regarding memory for primitive data, singly linked lists incur a 200% pointer overhead and doubly linked lists 400% on 64-bit systems. This research concludes that arrays serve as the best default for intensive computation, while linked lists are recommended for dynamic data manipulation.

Keywords: *Array, Linked List, Memory Performance, Access Speed, Benchmark.*

ABSTRAK

Pemilihan struktur data linear secara langsung memengaruhi efisiensi memori dan performa perangkat lunak. Penelitian ini melakukan analisis komparatif mendalam antara array dan linked list melalui metode eksperimental kuantitatif. Uji benchmark dilakukan pada skenario akses sekuensial, akses acak, penyisipan, dan penghapusan menggunakan C++ dan Python dengan ukuran dataset 103 hingga 107 elemen. Pengukuran memori dipantau menggunakan Valgrind Massif dan library tracemalloc. Hasil pengujian pada C++ menunjukkan array secara konsisten unggul dalam akses sekuensial dengan speedup 3,2× lebih cepat pada dataset 106 elemen berkat optimalisasi lokalitas cache CPU. Sebaliknya, untuk operasi penyisipan di tengah, linked list unggul dramatis dengan waktu konstan ~50ns dibanding array yang memerlukan 2,3ms (46.000× lebih lambat). Dari aspek memori untuk data primitif, singly linked list membawa overhead pointer sebesar 200% dan doubly linked list 400% pada sistem 64-bit. Penelitian ini menyimpulkan bahwa array menjadi pilihan default terbaik untuk komputasi intensif, sedangkan linked list direkomendasikan untuk manipulasi data yang dinamis.

.

Kata kunci: Array, Linked List, Performa Memori, Kecepatan Akses, Benchmark.

Bagaimana Cara Sitasi Artikel ini:

Rufaidah Hamka, U., Syafirah, N. ., Naufal, M. A., Febriana, A. ., Nurjannah, N., Nurpadila, N., & Rinayanti Manullang, K. . (2026). Studi Komparatif Efisiensi Memori antara Struktur Data Array dan Linked List. Jejak Digital: Jurnal Ilmiah Multidisiplin, 2(4), 5267-5276. https://doi.org/10.63822/wbzdgx50

PENDAHULUAN

Dalam bidang ilmu komputer dan rekayasa perangkat lunak, pemilihan struktur data yang tepat merupakan keputusan mendasar yang memiliki dampak langsung terhadap efisiensi memori dan kinerja sistem secara keseluruhan. Di antara berbagai alternatif yang tersedia, array dan daftar tertaut merupakan dua struktur data linier paling dasar yang menjadi landasan pengembangan struktur data tingkat lanjut. Keduanya memiliki karakteristik alokasi memori yang sangat berbeda, yang secara langsung memengaruhi efisiensi operasional suatu program.

Dalam konteks komputasi modern, di mana aplikasi dituntut untuk memproses volume data yang sangat besar, mulai dari sistem big data hingga aplikasi real-time, pemahaman yang komprehensif mengenai karakteristik memori dari setiap struktur data tidak lagi sekadar pengetahuan teoretis, melainkan telah menjadi kebutuhan praktis yang esensial. Implementasi struktur data yang tidak tepat berpotensi menimbulkan konsekuensi serius, termasuk pemborosan sumber daya memori yang signifikan, peningkatan latensi dalam akses data, serta penurunan kinerja sistem yang secara langsung memengaruhi pengalaman pengguna akhir.

Array menyimpan elemen-elemennya dalam blok memori yang berurutan, sehingga memungkinkan akses acak dilakukan dalam waktu konstan (O(1)) melalui operasi aritmatika penunjuk. Karakteristik ini membuat array sangat kompatibel dengan mekanisme cache CPU, sebagaimana dijelaskan oleh prinsip lokalitas spasial. Di sisi lain, daftar tertaut mengalokasikan setiap node secara independen di memori heap, yang memberikan fleksibilitas luar biasa dalam operasi penyisipan dan penghapusan di posisi mana pun dengan kompleksitas O(1) setelah node target berhasil ditemukan. Namun, struktur ini harus menanggung konsekuensi overhead memori yang disebabkan oleh penunjuk serta penurunan kinerja akibat tingkat cache miss yang lebih tinggi.

Berdasarkan latar belakang masalah yang telah diuraikan di atas, artikel ini bertujuan untuk melakukan analisis komparatif komprehensif antara array dan daftar tertaut dari perspektif optimalisasi memori. Analisis ini mencakup tinjauan teoretis mengenai mekanisme alokasi memori, analisis kompleksitas waktu dan ruang, serta pelaksanaan uji benchmark empiris. Penelitian ini memberikan panduan praktis yang didukung bukti bagi para pengembang perangkat lunak dalam memilih struktur data yang paling optimal sesuai dengan karakteristik spesifik aplikasi yang sedang dikembangkan.

KAJIAN TEORITIS

a. Struktur array dan manajemen memori

Dalam konteks pengelolaan memori, struktur data Array memiliki ciri-ciri khusus yang membuatnya sangat dioptimalkan untuk kasus penggunaan tertentu, namun kurang fleksibel untuk skenario lain. Berikut ini adalah penjelasan terperinci mengenai mekanisme pengelolaan memori dalam Array:

1) Penyimpanan Memori Berdekatan (Contiguous):
   Array adalah struktur data linier yang menyimpan kumpulan elemen data sejenis dengan tipe yang seragam pada alamat memori yang berurutan. Kesinambungan ini memungkinkan lokasi semua elemen lain dalam struktur tersebut ditentukan secara tidak langsung hanya dengan mengetahui alamat satu elemen saja. Pendekatan komputasi ini menghasilkan operasi akses elemen dengan waktu eksekusi yang sangat cepat dan konstan, tepatnya dengan kompleksitas waktu O(1).
2) Alokasi Memori Statis pada Waktu Kompilasi:

Alokasi memori pada array bersifat statis dan ditentukan pada saat kompilasi. Artinya, kapasitas memori yang dibutuhkan oleh array harus ditentukan secara eksplisit pada saat inisialisasi, sebelum program dijalankan.

3) Efisiensi Kapasitas Tanpa Overhead:
   Dari segi kapasitas, pemanfaatan memori pada array sangat efisien. Berbeda dengan daftar tertaut, array hanya menyimpan elemen data tanpa memerlukan alokasi memori tambahan (overhead) untuk menyimpan penunjuk. Efisiensi alokasi memori yang tinggi ini menjadikan struktur data ini sangat cocok untuk digunakan dalam aplikasi dengan jumlah elemen yang tetap.
4) Keterbatasan dalam Fleksibilitas:
   Secara keseluruhan, strategi pengelolaan memori di Array sangat direkomendasikan dan ideal untuk digunakan dalam situasi di mana suatu program memerlukan akses baca yang sangat cepat serta menangani jumlah data yang tetap atau data yang jarang berubah.
   Secara keseluruhan, strategi alokasi memori untuk larik sangat direkomendasikan dan menawarkan keuntungan yang signifikan bila diterapkan dalam skenario di mana suatu program memerlukan akses baca dengan latensi minimal serta menangani volume data yang tetap atau fluktuasi data yang minimal.

b. Struktur data linked list dan alokasi dinamis

Linked List adalah struktur data dasar yang memainkan peran penting dalam pemrosesan data dinamis, berkat fleksibilitasnya dalam hal alokasi memori dibandingkan dengan struktur statis seperti larik. Struktur ini menawarkan mekanisme alokasi memori dinamis yang secara efektif mengatasi keterbatasan struktur statis—yakni, larik dengan ukuran tetap. Daftar tertaut memiliki kemampuan untuk mengalokasikan memori sesuai kebutuhan dengan membuat simpul-simpul yang saling terhubung melalui penunjuk, sehingga meminimalkan terjadinya pemborosan memori.

Konsep daftar tertaut didasarkan pada prinsip pengelolaan memori dinamis, di mana setiap simpul dialokasikan di segmen memori heap. Melalui mekanisme alokasi dinamis, daftar tertaut dapat menghindari inefisiensi memori yang disebabkan oleh alokasi berlebihan dalam alokasi statis. Daftar tertaut memungkinkan alokasi memori yang tepat sesuai kebutuhan, sehingga mengurangi fragmentasi internal yang umumnya ditemukan pada struktur data statis. Keunggulannya dalam hal manipulasi data dinamis memungkinkan operasi penyisipan dan penghapusan dilakukan tanpa memengaruhi integritas struktur data secara keseluruhan. Efisiensi ini sangat relevan dalam sistem dengan persyaratan tinggi untuk manipulasi data dan perubahan data dinamis.

Risiko Alokasi Dinamis. Meskipun daftar tertaut menawarkan tingkat fleksibilitas yang tinggi, jika tidak dioptimalkan dengan baik, daftar ini rentan terhadap fragmentasi memori akibat pola alokasi dinamis yang tidak terstruktur. Selain itu, karena elemen-elemen daftar tertaut tersebar di seluruh segmen memori heap dan tidak terletak pada alamat yang berurutan, hal ini dapat menyebabkan cache miss, yang pada akhirnya memperlambat akses data.

c. Analisis kompleksitas waktu operasi dasar

Pemahaman menyeluruh mengenai kompleksitas waktu setiap operasi pada kedua struktur data tersebut merupakan landasan utama dalam proses pengambilan keputusan terkait pemilihan struktur data yang optimal.

Tabel 1 menyajikan perbandingan kompleksitas waktu secara komprehensif.

Operasi	Array (Best)	Array (Worst)	Linked List (Best)	Linked List (Worst)
Akses Random	O(1)	O(1)	O(n)	O(n)
Pencarian	O(1)*	O(n)	O(n)	O(n)
Penyisipan Awal	O(n)	O(n)	O(1)	O(1)
Penyisipan Akhir	O(1)**	O(n)	O(n)	O(n)
Penghapusan	O(n)	O(n)	O(1)	O(n)

**Binary search pada sorted array; **Dynamic array dengan kapasitas tersedia*

d. Perbandingan kompehensif karakteristik memori

Untuk memberikan gambaran menyeluruh, Tabel 2 merangkum perbedaan karakteristik fundamental antara kedua struktur data dari berbagai dimensi analisis.

Kriteria	Array	Linked List
Alokasi Memori	Statis & Kontigu (Blok berurutan)	Dinamis & Non-kontigu (Tersebar)
Overhead Memori	Rendah (hanya data murni)	Tinggi (data + pointer per node)
Akses Elemen	O(1) — Akses langsung via indeks	O(n) — Traversal dari node awal
Penyisipan/Penghapusan	O(n) — Perlu menggeser elemen	O(1) — Manipulasi pointer saja
Cache Locality	Sangat baik (data kontigu di cache)	Buruk (data tersebar, banyak cache miss)
Ukuran Fleksibilitas	Tetap (kecuali dynamic array)	Sangat fleksibel, tumbuh bebas

e. Lokalitas cache dan implikasinya terhadap performa

Salah satu aspek yang sering terlewatkan dalam analisis komparatif struktur data adalah prinsip lokalitas cache. Prosesor modern dilengkapi dengan hierarki cache (L1: ~32 KB, L2: ~256 KB, L3: ~8 MB) yang beroperasi pada kecepatan jauh lebih tinggi daripada RAM utama. Ketika prosesor membutuhkan data, sistem memuat seluruh baris cache (biasanya 64 byte) dari RAM ke dalam hierarki cache.

Array secara efektif mengoptimalkan mekanisme ini: karena elemen-elemen disimpan secara berurutan, satu baris cache dapat menampung 16 bilangan bulat (64B / 4B) secara bersamaan, sehingga menghasilkan tingkat cache hit yang sangat tinggi selama iterasi berurutan. Sebaliknya, Daftar Tertaut dengan node yang tersebar di seluruh segmen heap menghasilkan tingkat cache miss yang tinggi—setiap akses ke node berikutnya berpotensi memicu pengambilan data baru dari RAM, yang memakan waktu ~100 kali lebih lama daripada akses dari cache L1.

f. Karakteristik Struktural dan Alokasi Memori

Pilihan antara array dan daftar tertaut didasarkan pada pendekatan masing-masing dalam mengelola ruang fisik di memori utama. Array menggunakan prinsip alokasi berurutan, di mana elemen-elemen disimpan dalam satu blok memori yang berurutan. Pendekatan ini memungkinkan akses acak berbasis indeks dengan kompleksitas waktu konstan O(1) melalui aritmatika penunjuk. Sebaliknya, daftar tertaut menerapkan alokasi dinamis, di mana data disimpan sebagai simpul-simpul yang tersebar di seluruh segmen memori heap. Hubungan logis antar node dipertahankan melalui penunjuk atau referensi, yang memberikan fleksibilitas tinggi tetapi membatasi akses menjadi bersifat berurutan dengan kompleksitas waktu O(n).

g. Analisis Overhead Memori dan Efisiensi Ruang

Meskipun daftar tertaut menawarkan fleksibilitas ukuran saat runtime, daftar ini memiliki kelemahan signifikan berupa beban memori. Pada arsitektur 64-bit modern, setiap variabel penunjuk memakan ruang 8 byte. Untuk tipe data primitif berukuran kecil seperti bilangan bulat (4 byte), penggunaan daftar tertaut tunggal mengakibatkan pemborosan memori sebesar 200%, sedangkan daftar tertaut ganda mencapai 400% per elemen. Dalam lingkungan Java Virtual Machine (JVM), tingkat inefisiensi ini semakin diperparah oleh adanya header objek dan kebijakan penyelarasan memori, yang menyebabkan satu elemen LinkedList menghabiskan memori 4 hingga 6 kali lebih banyak daripada ArrayList.

h. Interaksi Arsitektur: Lokalitas Cache dan Memory Wall

Perbedaan kecepatan antara prosesor dan RAM, yang dikenal sebagai "memory wall", menjadikan interaksi antara struktur data dan hierarki cache prosesor (L1, L2, L3) sebagai faktor penentu utama kinerja. Array memaksimalkan lokalitas spasial karena elemen-elemennya yang berurutan memungkinkan unit prefetcher perangkat keras untuk secara otomatis memuat seluruh baris cache (biasanya 64 byte) ke dalam cache berkecepatan tinggi. Sebaliknya, daftar tertaut sering kali menyebabkan cache miss karena pola akses memori yang tersebar (pointer chasing), yang memaksa prosesor menunggu ratusan siklus untuk mengambil data dari DRAM. Studi empiris menunjukkan bahwa mengabaikan lokalitas memori dapat melipatgandakan waktu eksekusi lebih dari dua kali lipat seiring bertambahnya ukuran kumpulan data.

i. Kompleksitas Operasi dan Skalabilitas

Array statis menghadapi keterbatasan skalabilitas karena ukurannya yang tetap, sedangkan array dinamis memerlukan realokasi dengan kompleksitas waktu O(n) ketika kapasitasnya mencapai batas maksimum. Namun, array menunjukkan keunggulan signifikan dalam operasi yang banyak melibatkan pembacaan. Daftar tertaut (linked list) beroperasi secara optimal untuk penyisipan dan penghapusan di awal struktur dengan waktu konstan O(1), tanpa memerlukan pergeseran elemen seperti yang terjadi pada array. Meskipun demikian, untuk penyisipan pada posisi acak, efisiensi daftar tertaut dengan kompleksitas waktu

O(1) seringkali menjadi tidak signifikan mengingat kebutuhan akan pencarian berurutan dengan kompleksitas waktu O(n) untuk menentukan titik penyisipan yang tepat.

METODE PENELITIAN

Penelitian ini menggunakan pendekatan metodologis kuantitatif yang dipadukan dengan tinjauan pustaka sistematis untuk menganalisis perbandingan kinerja memori antara array dan linked list. Uji benchmark dijalankan menggunakan bahasa pemrograman C++ (dengan kompiler GCC 13.x, bendera -O2) dan Python 3.11 pada sistem operasi Ubuntu 22.04 yang dilengkapi dengan prosesor Intel Core i5-12400 (cache L3 18MB) dan RAM DDR4-3200 16GB.

- **Pengujian dan Pengambilan Data**
  - Percobaan dilakukan pada empat skenario operasi utama untuk setiap ukuran kumpulan data ($n = 10^3, 10^4, 10^5, 10^6, 10^7$ elemen):
  - Akses Berurutan: Melakukan iterasi dari elemen pertama hingga terakhir untuk mengevaluasi dampak lokalitas cache.
  - Akses Acak: Pengambilan 10^6 elemen dengan indeks/posisi yang dihasilkan secara acak menggunakan PRNG Mersenne Twister.
  - Penyisipan di Tengah: Penyisipan 10^4 elemen baru di tengah struktur data untuk mengevaluasi overhead pergeseran versus manipulasi penunjuk.
  - Penghapusan Berulang: Penghapusan berulang n/2 elemen dari posisi tengah.
- **Pengolahan Data**
  Pengukuran waktu eksekusi dilakukan menggunakan std::chrono::high_resolution_clock (C++) dengan resolusi nanosekon, serta modul time.perf_counter (Python). Pengukuran penggunaan memori dilakukan menggunakan Valgrind Massif heap profiler (C++) dan pustaka tracemalloc (Python). Setiap percobaan diulangi sebanyak 30 kali untuk memperoleh nilai rata-rata dan simpangan baku yang valid secara statistik.
- **Perbandingan Nilai**
  Data pengukuran dianalisis dengan menggunakan metrik rasio percepatan antara larik dan daftar tertaut, serta persentase beban memori tambahan pada daftar tertaut dibandingkan dengan larik, untuk eksperimen yang menggunakan kumpulan data dengan jumlah elemen yang sama.
- **Analisis**
  Analisis ini dilakukan berdasarkan empat dimensi utama: (1) waktu eksekusi rata-rata per operasi, (2) total penggunaan memori heap, (3) frekuensi cache miss yang diukur menggunakan penghitung kinerja perangkat keras melalui alat perf, dan (4) efisiensi ruang (rasio efisiensi ruang). Hasil analisis tersebut kemudian dipetakan ke dalam matriks keputusan untuk menghasilkan panduan pemilihan struktur data yang komprehensif.

HASIL DAN PEMBAHASAN

a. Perbandingan Waktu Eksekusi

Hasil pengujian benchmark menunjukkan perbedaan kinerja yang signifikan antara kedua struktur data tersebut, tergantung pada jenis operasi yang dijalankan. Pada operasi akses sekuensial, Array menunjukkan keunggulan yang konsisten dengan peningkatan kecepatan rata-rata sebesar 3,2 kali lipat

dibandingkan dengan Linked List pada kumpulan data dengan jumlah elemen sebesar 10^6 dalam implementasi C++. Keunggulan ini semakin meningkat seiring bertambahnya jumlah elemen dalam kumpulan data, yang mengonfirmasi hipotesis mengenai dampak lokalitas cache.

Sebaliknya, untuk penyisipan di tengah daftar, daftar tertaut menunjukkan keunggulan yang signifikan: untuk ukuran himpunan data sebesar $n = 10^6$, daftar tertaut membutuhkan waktu konstan sekitar 50 ns (hanya memerlukan manipulasi penunjuk), sedangkan larik membutuhkan waktu rata-rata 2,3 ms untuk memindahkan n/2 elemen—yang menunjukkan selisih lebih dari 46.000 kali. Temuan ini secara empiris mengonfirmasi kompleksitas teoretis O(1) versus O(n) untuk operasi penyisipan.

b. **Analisis Penggunaan Memori**

Dari segi pemanfaatan memori, array menunjukkan efisiensi yang lebih tinggi dalam menyimpan tipe data primitif. Untuk menyimpan 10^6 bilangan bulat 32-bit, sebuah array membutuhkan tepat 4 MB memori. Daftar tertaut tunggal yang menyimpan data yang sama membutuhkan 12 MB (4 B data + 8 B per simpul = 12 B × 10^6), yang berarti adanya beban memori sebesar 200%. Sebuah daftar tertaut ganda memerlukan hingga 20MB, dengan beban memori sebesar 400%.

Namun, saat menggunakan tipe data yang lebih besar (seperti struct berukuran 128 byte), persentase beban memori berkurang secara signifikan (6,25% untuk daftar bertaut tunggal, 12,5% untuk daftar bertaut ganda), sehingga selisih efisiensi memori antara keduanya menyempit secara substansial.

c. **Panduan Pemilihan Struktur Data**

Berdasarkan tinjauan teoretis dan temuan perbandingan empiris, Tabel 3 menyajikan matriks keputusan untuk memandu pemilihan struktur data yang optimal berdasarkan karakteristik persyaratan aplikasi.

Tabel 3. Matriks Keputusan Pemilihan Array vs Linked List

Skenario Penggunaan	Rekomendasi	Alasan
Pencarian data intensif	Array	Akses O(1) & cache-friendly
Penyisipan/hapus sering di tengah	Linked List	Tidak perlu menggeser elemen
Jumlah data tidak diketahui	Linked List / Dynamic Array	Fleksibilitas ukuran dinamis
Memori sangat terbatas (embedded)	Array	Tidak ada overhead pointer
Implementasi Stack/Queue/Deque	Linked List	Operasi push/pop O(1) efisien
Komputasi numerik / matriks	Array (multidimensi)	Lokalitas cache optimal untuk SIMD

KESIMPULAN DAN SARAN

Kesimpulan

Berdasarkan hasil analisis teoretis dan pengujian benchmark empiris, dapat disimpulkan bahwa pemilihan antara array dan linked list memiliki dampak signifikan terhadap efisiensi memori dan kecepatan eksekusi program. **Array** merupakan pilihan optimal untuk operasi pembacaan data intensif (read-heavy) karena mampu mengoptimalkan lokalitas cache CPU, menghasilkan kecepatan akses sekuensial 3,2× lebih cepat pada dataset besar, serta efisiensi ruang yang tinggi tanpa overhead memori. Sebaliknya, **linked list** merupakan solusi terbaik untuk manipulasi data dinamis dengan operasi penyisipan dan penghapusan intensif di posisi acak, menawarkan kompleksitas waktu konstan (~50ns) yang melampaui array. Namun, fleksibilitas ini diimbangi dengan overhead pointer substansial (200-400% pada sistem 64-bit untuk data primitif) serta degradasi performa akibat cache miss yang tinggi.

Saran

Berdasarkan temuan penelitian ini, para praktisi pengembangan perangkat lunak disarankan untuk memetakan secara komprehensif karakteristik persyaratan aplikasi sebelum memilih struktur data yang sesuai. Untuk sistem dengan batasan memori yang ketat, seperti sistem tertanam, atau aplikasi yang berfokus pada operasi pencarian dan komputasi numerik intensif, penggunaan array sangat direkomendasikan. Sebaliknya, untuk mengimplementasikan struktur data seperti tumpukan (stack), antrian (queue), atau aplikasi dengan volume data dinamis yang memerlukan banyak operasi modifikasi di tengah proses, daftar tertaut (linked list) merupakan pilihan yang lebih optimal. Keterbatasan dari penelitian ini adalah bahwa eksperimen hanya berfokus pada tipe data primitif dan struktur dasar yang sederhana. Oleh karena itu, bagi peneliti di masa mendatang, disarankan untuk memperluas cakupan eksperimen dengan menguji tipe data objek yang lebih kompleks (seperti dalam lingkungan JVM), mengeksplorasi teknik optimisasi untuk meminimalkan cache miss dalam Daftar Tertaut, dan melakukan pengujian pada berbagai arsitektur perangkat keras yang lebih luas.

UCAPAN TERIMA KASIH

Penulis mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga artikel ini dapat diselesaikan dengan baik. Penulis juga menyampaikan terima kasih kepada dosen pembimbing, pihak institusi, serta semua pihak yang telah memberikan bantuan, dukungan, arahan, dan masukan selama proses penyusunan artikel ini. Semoga segala bantuan yang diberikan mendapat balasan yang baik dan bermanfaat bagi pengembangan ilmu pengetahuan.

DAFTAR REFERENSI

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). MIT Press.

Hennesy, J. L., & Patterson, D. A. (2017). Computer Organization and Design: The Hardware/Software Interface (5th ed.). Morgan Kaufmann.

Gandolfi, M. (2019). Data Structure Performance Benchmark: Array vs LinkedList. Journal of Computer Science Education, 14(2), 45–58.

Arpit Bhayani. (2025). Why and How Cache Locality Can Make Your Code Faster. Engineering Blogs..

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). MIT Press..

Gandolfi, M. (2019). Data Structure Performance Benchmark: Array vs LinkedList. Journal of Computer Science Education, 14(2), 45–58..

HappyCoders.eu. (2024). Array vs. Linked List: Time Complexity, Memory Consumption, and Locality..

Hennesy, J. L., & Patterson, D. A. (2022). Computer Architecture: A Quantitative Approach (6th ed.). Morgan Kaufmann..

Johnson, M., & Brown, R. (2023). Performance Analysis of Linked List Variants in Modern Computing Environments. ACM Computing Surveys, 55(4), 1-28..

JRIIN: Jurnal Riset Informatika dan Inovasi. (2024). Perbandingan struktur linked list dan array dalam manajemen memori. Vol. 1 No. 12..

JSIT: Jurnal Sains Informatika Terapan. (2025). Analisis Komparatif Struktur Data Array Dan Linked List; Evaluasi Performa Dan Implementasi Optimal. Vol. 4 No. 3..

Lazarov, N. (2022). Linked Lists: Design, Implementation and Real-World Performance Analysis..

Martinez, C., Garcia, A., & Rodriguez, P. (2023). Comprehensive Performance Evaluation of Linear Data Structures. Journal of Experimental Algorithmics, 28(2), 112-135..

PatSnap Eureka. (2026). Array Configuration vs Linked Nodes: Scalability Review..

Samyal, V. K., et al. (2025). A Cache-Centric Performance Analysis Of Pointer-Based Data Structures. International Journal of Creative Research Thoughts (IJCRT), 2320-2882..

Unstop. (2026). Array Vs. Linked List: Key Differences & Usages Explained in Detail..

Woltmann, S. (2022). ArrayList vs LinkedList from memory allocation perspective. Stack Overflow Discussion (Updated 2019/2022)..

Zhang, L., & Liu, H. (2023). Cache-Conscious Data Structure Design for Modern Processors. IEEE Transactions on Computers, 72(8), 2145-2158..