Perbandingan Kinerja Metode Iteratif dan Metode Rekursif dalam Algoritma Binary Search

Erba Lutfina*[1], Fachrian Luthfi Ramadhan[2]
[1,2]Universitas Nasioanal Karangturi; Jl. Raden Patah No.182-192, Rejomulyo, Kec. Semarang Timur, Kota Semarang, (024) 3545882
e-mail: *[1]erbalutfina@gmail.com, [2]fachrianlr@gmail.com

Abstrak

Penelitian ini membahas metode dan algoritma yang biasa digunakan dalam proses pencarian. Metode dan algoritma yang dibahas dalam paper ini adalah algoritma binary search dengan metode Rekursif dan metode iteratif. Pada masing-masing algoritma akan diberikan contoh program dalam bahasa pemrograman Python agar bisa memberi sedikit gambaran tentang implementasinya. Setiap algoritma akan dianalisis untuk mengetahui metode mana yang menghasilkan kompleksitas notasi big-O, penggunaan memori dan waktu akses yang paling efisien. Dari penelitian ini diketahui bahwa metode iteratif memiliki hasil yang lebih baik daripada metode rekursif. Hal ini dibuktikan dengan hasil waktu eksekusi metode rekursif sebesar 1.0601 s, 1.414 s, 4.515 s serta 51.4 MiB, 400.4 MiB, 3884.3 MiB untuk penggunaan memori, sedangkan algoritma iteratif hanya membutuhkan 0.0102 s, 0.0106 s, 0.0584 s untuk waktu eksekusi dan penggunaan memori berjumlah 44.3 MiB, 323.5 MiB, 3116.2 MiB.

Kata kunci—Algoritma binary search, notasi Big-O, Rekursif, Iteratif.

Abstract

This paper discusses the methods and algorithms that commonly used in the search process. Methods and algorithms discussed in this paper is a binary search algorithm with recursive and iterative method. In each algorithm will be given an examples using Python programming languages in order to give some idea about the implementation. Each algorithm will be analyzed to find out which method produces the most efficient complexity of big-O notation, memory usage and access time. From this research it is known that the Iterative method has a better result than the Recursive methods. Shown by the results of the recursive method with 1.0601 s, 1.414 s, 4.515 s for execution time and 51.4 MiB, 400.4 MiB, 3884.3 MiB for memory usage, while the iterative algorithm only requires 0.0102 s, 0.0106 s, 0.0584 s for the execution time and 44.3 MiB, 323.5 MiB, 3116.2 MiB memory usage.

Keywords—Binary Search Algorithm, Big-O notation, Recursive, Iterative.

1. PENDAHULUAN

Terdapat berbagai cara untuk memecahkan masalah menggunakan program komputer. Misalnya, ada beberapa cara untuk melakukan pencarian data, salah satunya adalah algoritma *binary search*. Algoritma *binary search* merupakan sebuah list yang sudah diurutkan dan kemudian dibagi menjadi dua bagian [1]. Dalam pemrosesan data sebuah list menggunakan algoritma *binary search* dapat diimplementasikan dengan cara iteratif dan rekursif [2]. Semua algoritma ini memiliki kelebihan dan kekurangan serta dapat digunakan untuk menyelesaikan masalah tertentu. Untuk mengetahui algoritma mana yang digunakan ketika terdapat beberapa

solusi untuk memecahkan masalah akan dilakukan pemilihan algoritma. Pemilihan algoritma dilakukan dengan melakukan analisis beberapa algoritma agar diperoleh algoritma yang paling mangkus. Analisis algoritma dalam penelitian ini mengacu pada analisis kompleksitas dari kedua algoritma yang berbeda berdasarkan penggunaan memori, waktu akses dan notasi *Big-O* [3].

Terdapat berbagai penelitian yang mengungkapkan perbandingan pada proses pencarian menggunakan rekursif dan iteratif. Seperti penelitian yang dilakukan oleh Mirolo [4] pada tahun 2012 dengan membandingkan metode rekursif dan iteratif untuk mengetahui metode mana yang menghasilkan penulisan program yang mudah dipahami dan lebih efisien. Penelitian dilakukan pada mahasiswa *undergraduate* dengan menggambarkan *task copy* dengan menggunakan versi iteratif dan rekursif. Hasil penelitian menunjukkan bahwa pada proses pencarian, iteratif lebih mudah dipahami dan lebih efisien dibanding rekursif. Mauricio [5] pada tahun 2015 melakukan penelitian untuk menganalisis perbedaan antara proses rekursif dan iteratif pada *Visual Hierarchical Processing*. Penelitian tersebut mengusulkan dua metode baru yaitu *Visual Recursion Task* (VRT) yang digunakan untuk mewakili level hirarki baru, dan metode *Embedded Iteration Task* (EIT) yang mewakili penambahan elemen pada level hirarki yang sudah ada. Hasil penelitian menunjukkan bahwa metode rekursif menghasilkan korelasi yang lebih baik dibanding dengan metode iteratif kecuali pada *spatial short-term memory* dan *memory usage*. Pada tahun 2016 Kaser [6] melakukan penelitian yang fokus pada peningkatan *running time* ketika jumlah inputan menjadi sangat besar. Notasi *big-O* digunakan untuk membandingkan tingkat peningkatan waktu tiap algoritma terhadap tingkat pertumbuhan fungsi. Selanjutnya penelitian yang dilakukan oleh Sklyarov [7] yang menganalisis dan membandingkan implementasi rekursif dan iteratif dari sirkuit FPGA untuk berbagai masalah. Terdapat 2 tipe rekursif yg digunakan yaitu *cyclic* dan *binary search*. Eksperimen dilakukan pada 4 masalah yang berbeda. Dalam eksperimen 9 contoh diuji secara acak dan memberikan hasil bahwa algoritma rekursif meningkatkan waktu eksekusi dan sumber daya perangkat keras yang dibutuhkan. Tetapi pada sebagian besar aplikasi, implementasi iteratif menghasilkan hasil yang lebih baik.

Dari berbagai analisis penelitan diatas, perbandingan algoritma sering dilakukan untuk mencari algoritma mana yang lebih mangkus dan cocok pada data yang digunakan. Oleh karena itu penelitian mengenai perbandingan metode rekursif dan iteratif pada algoritma *binary search* akan dilakukan untuk mengetahui algoritma mana yang sesuai dan lebih mangkus. Penelitian ini bertujuan untuk menganalisis kinerja algoritma binary search menggunakan metode iteratif maupun rekursif. Serta membandingkan hasil mengenai waktu eksekusi dan penggunaan memori pada proses pencarian data menggunakan bahasa pemrograman Python.

2. METODE PENELITIAN

2.1 Dataset

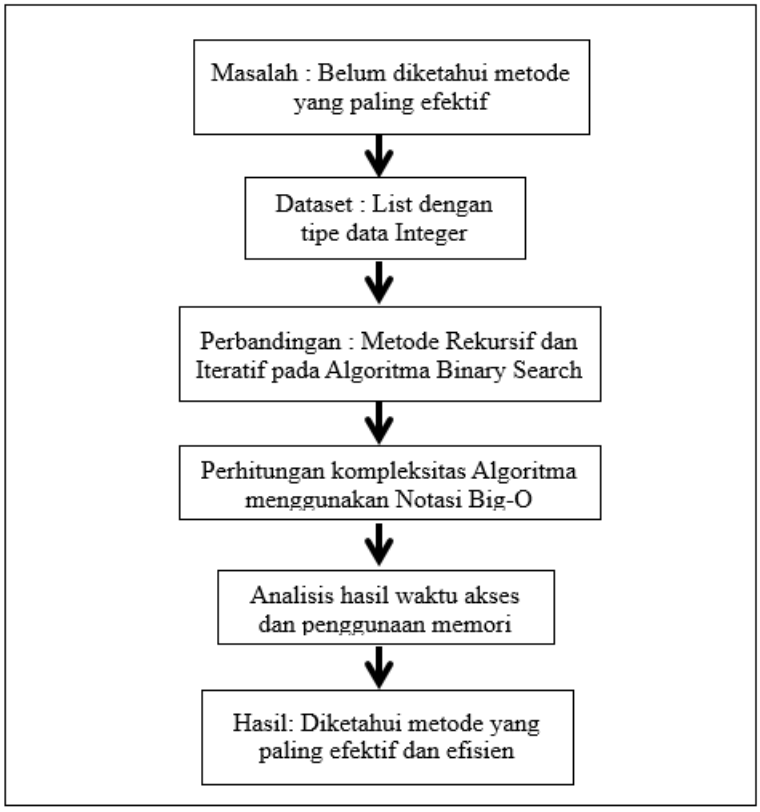
Pada penelitian ini dataset yang digunakan adalah data *list*. Data *list* tersebut menjadi parameter dari fungsi binary search iteratif dan rekursif. Data *list* yang digunakan adalah data integer, masing-masing sebanyak 1 juta, 10 juta dan 100 juta data dengan 3 data pencarian. Value dari data list tersebut digenerate secara random. Selanjutnya dilakukan looping sebanyak 1 juta, 10 juta, dan 100 juta data, lalu proses dilanjutkan dengan mengambil nilai interger secara random.

2.2 Strategi Penelitian

Metode penelitian yang digunakan dalam penelitian ini adalah metode kualitatif . Dalam penelitian ini metode kualitatif dilakukan untuk menemukan suatu dasar pengetahuan sebagai tolak ukur atau acuan pada penerapan kasus yang sama. Penelitian ini biasanya dilakukan di mana penyelesaian masalah perlu dilakukan, dan hasilnya diperlukan sebagai acuan, tolak ukur serta pembanding.

Pada proses implementasi, penelitian ini mengusulkan proses perhitungan kompleksitas algoritma *binary search* menggunakan metode rekursif dan iteratif menggunakan bahasa pemrograman *Python*. Penelitian ini memiliki tujuan untuk membandingkan performa yang

dihasilkan dari tiap algoritma untuk mengetahui algoritma mana yang paling mangkus. Tahap implementasi terbagi menjadi dua fase yaitu penerapan metode rekursif dan iteratif pada algoritma *binary search*, dan perbandingan performa yang dihasilkan dari kedua algoritma menggunakan notasi *Big-O*. Berikut merupakan penjabaran langkah-langkah dari metode yang diusulkan:



Gambar 1 Strategi Penelitian

3. HASIL DAN PEMBAHASAN

3.1 Algoritma Binary Search Menggunakan Metode Rekursif

Pembahasan pada penelitian ini diawali dengan algoritma *binary search* menggunakan metode rekursif untuk proses pencarian item. Dalam penelitian ini algoritma *binary search* diimplementasikan dengan menggunakan bahasa pemrograman *Python*. Di bawah ini merupakan *pseudocode* dari proses *binary search* menggunakan metode rekursif [8].

Tabel 1 *Pseudocode binary search* rekursif

Binary Search Rekursif
Inputan A[0..N-1],Value, Low, High

If (high < low)
  then return not found
mid ← (low + high) / 2
if (A[mid] > value)
  then return BinarySearch(A, Value, Low, mid-1)
else if (A[mid] < value)
  then return BinarySearch(A, Value, mid+1, high)
else
  return mid

Dari *pseudocode* di atas dapat diketahui bahwa:

- Fungsi *BinarySearch* memiliki *list* dan variabel *value* sebagai argumen. Fungsi mencari *value* dalam rentang A[0..N-1].
- *Base case* terdiri dari pengujian apakah *high* kurang dari *low*. Jika tidak, dikembalikan nilai *not found*.
- *Mid* dihitung sebagai dasar dari rata-rata awal dan akhir.
- Jika elemen pada indeks *mid* kurang dari *value*, *BinarySearch* dipanggil lagi dengan mengembalikan nilai *mid* + 1 dan jika lebih dari *value* maka *high = mid*. Jika tidak, *mid* dikembalikan sebagai indeks dari elemen yang ditemukan.

Source code yang didapat dari algoritma *binary search* rekursif diatas ditunjukkan sebagai berikut:

```
def BinarySearch(self, L, n):
  if len(L) == 0:
    return False
  else:
    mid = len(L)/2
    if n == L[mid]:
      return True
    else:
      if n > L[mid]:
        return self.BinarySearch(L[mid+1:], n)
      else:
        return self.BinarySearch(L[:mid], n)
```

Gambar 2 *Source code binary search* rekursif

Dari program diatas kita dapat menghitung kompleksitas waktu *binary search* rekursif. Untuk menghitung kompleksitas waktu, kita anggap *T(n)* adalah waktu yang dibutuhkan pada fungsi *BinarySearch*. Langkah pertama dilakukan pengecekan pada kondisi *if len(L) == 0* yang bernilai satu perbandingan. Pengecekan kondisi kedua sebesar satu perbandingan pada inisialisasi *mid = len(L) / 2*. Pengecekan kondisi *if n == L[mid]* sebesar satu perbandingan dan *if n == L[mid]* sebesar satu perbandingan. Selanjutnya fungsi *BinarySearch* membagi *problem* menjadi dua bagian yaitu *return self.BinarySearch(L[mid+1:], n)* dan *return self.BinarySearch(L[:mid], n)* yang bernilai sebesar (n/2) perbandingan.

Analisis kompleksitas waktu dilakukan pada kasus terbaik, terburuk dan *average* adalah sebagai berikut:

- Kompleksitas waktu pada kasus terbaik bernilai *O(1)* atau konstan. Terjadi apabila *value* yang dicari berada di tengah *list* serta setiap melewati blok *while* dihitung sebagai satu perbandingan.
- Pada kasus *average*, pencarian selalu dibagi menjadi 2 bagian. Perilaku ini selalu terjadi sehingga kompleksitas waktu memiliki orde sebesar *log n.* Penjabaran secara matematis ditunjukkan sebagai berikut:

$T(n) = T(n/2) + 4$
Dengan $T(n/2) = T(n/2^2)+1$, maka
$T(n) = (T(n/2^2) + 1) + 4$
$= T(n/2^2) +5$
$= T(n/2^3) +6$
$= T(n/2^4) +7$
.
.
$= [T(n / 2^k) + k]$

Dari penjabaran diatas didapatkan nilai T(n) sebesar :

$$T(n) = [T(n / 2^k + k\ ] \ (1)$$

Asumsikan $n / 2k = 1$
Jadi $n = 2k$ maka $k = \log n$
$T(N) = [T(n / 2^k) + k]$
$= T (16 / 16) + \log n$
$= T(1) + \log n$
$= O (\log n)$

- Pada kasus terburuk, item yang dicari tidak terdapat sama sekali di dalam *list*. Sehingga pada setiap iterasi *binary search, range* penerima dibagi dua. Pembagian dapat dilakukan sebanyak *O (log n)* kali.

3.2 Algoritma Binary Search Menggunakan Metode Iteratif

Pembahasan algoritma ini menggunakan metode iteratif atau berulang untuk mengimplementasikan algoritma *binary search*. Di bawah ini adalah *pseudocode* pencarian biner menggunakan metode iteratif [9].

Tabel 2 *Pseudocode binary search* iteratif

Binary Search Iteratif
Inputan A[0..N-1],Value

low ← 0
high ← N-1
while (low <= high) **do**
{
mid ← (low + high) / 2
if (A[mid] >= value)
high ← mid - 1
else
low ← mid + 1
}
return low

Dari *pseudocode* di atas diketahui bahwa :

- Fungsi *BinarySearch_left* memiliki argumen *List* dan *value.*
- Variabel *low* diset ke 0 dan *high* diset ke panjang dari *List*.
- Variabel *low* melacak elemen pertama di bagian *List* yang sedang dicari sementara *high* melacak elemen setelah bagian akhir yang dicari.
- *While loop* dibuat dan melakukan iterasi selama *low* < *high*.
- *Mid* dihitung dari rata-rata awal dan akhir.

- Jika elemen pada indeks *mid* kurang dari *value*, maka *low* bernilai *mid + 1* dan jika lebih besar dari *value*, maka *high* bernilai *mid*. Jika tidak, *mid* dikembalikan sebagai indeks dari elemen yang ditemukan.

Kita dapat mengimplementasikan algoritma *binary search* secara iteratif dalam bahasa pemrograman *python*. Kode *binary search* rekursif adalah sebagai berikut:

```
def BinarySearch2(self, L, n, nmin, nmax):
  while (nmin<=nmax):
    mid = (nmin+nmax)/2
    if n > L[mid]:
      nmin = mid + 1
    elif n < L[mid]:
      nmax = mid - 1
    else :
      return True
  return False
```

Gambar 3 *Source code binary search* Iteratif

Dari program diatas kita dapat menghitung kompleksitas waktu *binary search* iterative. Langkah pertama dilakukan pengecekan pada kondisi *While (nmin <= nmax).* Pengecekan kondisi kedua sebesar satu langkah pada inisialisasi *mid = (nmix + nmax) / 2*. Pengecekan kondisi *if n> L[mid]* dan *else* sebanyak 4 langkah.

Analisis kompleksitas waktu dilakukan pada kasus terbaik, terburuk dan *average*.

- Kompleksitas waktu pada kasus terbaik bernilai *O(1)* atau konstan. Terjadi apabila *value* yang dicari berada di tengah *list* serta setiap melewati blok *while* dihitung sebagai satu perbandingan.
- Sedangkan pada kasus *average, value* yang tidak berada pada *list* tidak akan dicari dan probabilitas pada pencarian untuk setiap elemen adalah seragam. Sehingga kompleksitas pada *average case* bernilai *O (log n).* Penjabaran secara matematis ditunjukkan sebagai berikut:

  $T(n) = T(n/2) + \Theta(1)$
  $= T(n/2) + \Theta(1) + \Theta(1)$
  $= T(n/4) + 2\ \Theta(1)$
  $= T(n/8) + 3\ \Theta(1)$
  $= T(n/16) + 4\ \Theta(1)$
  .
  .
  $= [T(n / 2^k) + k * \Theta(1)]$

  Dari penjabaran diatas didapatkan nilai T(n) sebesar :

  $$T(n) = [T(n / 2^k + k * \Theta(1)\ ] \quad (2)$$

  Asumsikan $n / 2^k = 1$
  Jadi $n = 2^k$ maka $k = \log n$
  $T(N) = [T(n / 2^k) + k * \Theta(1)]$
  $= T (16 / 16) + \log n * \Theta(1)$
  $= O (\log n)$
- Pada kasus terburuk, item yang dicari tidak terdapat sama sekali di dalam *list*. Sehingga pada setiap iterasi *binary search, range* penerima dibagi dua. Pembagian dapat dilakukan sebanyak *O (log n)* kali.

3.3 Hubungan Kompleksitas Waktu dengan Waktu Akses Eksekusi Program

Dalam penelitian ini telah dilakukan pengujian terhadap kedua algoritma untuk menemukan memori yang digunakan dan waktu yang dieksekusi. Dalam penelitian ini digunakan beberapa kasus yang berisi bilangan bulat dengan jumlah data yang berbeda yaitu 1 juta, 10 juta dan 100 juta data yang kemudian dilakukan pencarian pada salah satu item pada daftar tersebut.

Perbandingan hasil kinerja dari kedua algoritma ditunjukkan pada tabel berikut:

Tabel 3 Perbandingan *binary search* iteratif dan rekursif

	1 juta		10 juta		100 juta	
	iteratif	rekursif	iteratif	rekursif	iteratif	rekursif
Waktu Proses	0.0102 s	1.0601 s	0.0106 s	1.414 s	0.0584 s	4.515 s
Penggunaan Memori	44.3 MiB	51.4 MiB	323.5 MiB	400.4 MiB	3116.2 MiB	3884.3 MiB

Dari gambar diatas dapat kita lihat bahwa waktu eksekusi program dari *binary search* iteratif berturut-turut membutuhkan 0.0102 s, 0.0106 s, 0.0584 s. Sedangkan untuk waktu eksekusi *binary search* rekursif membutuhkan 1.0601 s, 1.414 s, 4.515 s. Dari perbandingan tersebut terlihat bahwa waktu eksekusi *binary search* iteratif lebih cepat dibandingkan dengan binary search rekursif. Meskipun kedua algoritma memiliki kompleksitas waktu sebesar *O(log n),* namun *binary search* iteratif cenderung memiliki faktor konstan yang lebih rendah karena tidak melibatkan manipulasi *call stack.*

Kedua algoritma memiliki penggunaan ruang memori yang berbeda. *Binary search* iteratif memiliki kompleksitas ruang sebesar *O(1),* sedangkan *binary search* rekursif dapat mencapai sebesar *O(log n)* dikarenakan *binary search* rekursif melibatkan manipulasi *call stack.* Dari gambar diatas *binary search* iteratif berturut-turut menggunakan 44.3 MiB, 323.5 MiB, 3116.2 MiB. Sedangkan *binary search* rekursif menggunakan sebanyak 51.4 MiB, 400.4 MiB, 3884.3 MiB. Perbandingan tersebut menunjukkan bahwa kompleksitas ruang *binary search* iteratif yaitu *O(1)* lebih besar dari kompleksitas rekursif yaitu *O(log n)* dengan bukti seperti diatas.

4. KESIMPULAN

Berdasarkan hasil percobaan dapat disimpulkan bahwa algoritma *binary search* iteratif lebih efisien daripada algoritma *binary search* rekursif. *Binary search* iteratif memiliki kompleksitas ruang sebesar *O(1),* sedangkan *binary search* rekursif dapat mencapai *sebesar O(log n).* Untuk kompleksitas waktu kedua algoritma memiliki kompleksitas sebesar *O(log n),* namun *binary search* iteratif cenderung memiliki faktor konstan yang lebih rendah. Hal ini dibuktikan oleh eksperimen menggunakan data *list* yang dipesan dari 1 juta, 10 juta hingga 100 juta data. *Binary search* rekursif menghasilkan waktu eksekusi lebih lama dan penggunaan memori yang lebih besar jika dibandingkan dengan *binary search* iteratif yang menghasilkan waktu eksekusi dan penggunaan memori yang lebih kecil.

5. SARAN

Untuk pengembangan penelitian kedepan, penelitian dapat dilakukan dengan menggunakan bahasa pemrograman lain serta dilakukan penambahan algoritma yang sudah ada untuk mengetahui algoritma mana yang paling mangkus.

DAFTAR PUSTAKA

[1] D. Roy and A. Kundu, “A Comparative Analysis of Three Different Types of Searching Algorithms in Data Structure,” vol. 3, no. 5, pp. 6626–6630, 2014.

[2] R. Mccauley, B. Hanks, S. Fitzgerald, and L. Murphy, “Recursion vs . Iteration : An Empirical Study of Comprehension Revisited,” pp. 350–355, 2015.

[3] V. P. Parmar, “Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array , Dynamic Array and Linked List,” vol. 121, no. 3, pp. 13–17, 2015.

[4] C. Mirolo, “Is Iteration Really Easier to Learn than Recursion for CS1 Students ?,” pp. 99–104, 2012.

[5] M. D. Martins, I. P. Martins, and W. T. Fitch, “A novel approach to investigate recursion and iteration in visual hierarchical processing,” 2015.

[6] O. Kaser, “Analyzing Code with Θ , O and Ω,” pp. 1–18, 2016.

[7] D. Mihhailov, V. Sklyarov, and A. Sudnitson, “Parallel FPGA-based Implementation of Recursive Sorting Algorithms,” pp. 121–126, 2010.

[8] S. Grissom, L. Murphy, and S. Fitzgerald, “Paper vs . Computer-based Exams : A Study of Errors in Recursive Binary Tree Algorithms,” pp. 6–11, 2016.

[9] M. Drmota and W. Szpankowski, “A Master Theorem for Discrete Divide and Conquer Recurrences,” vol. V, pp. 1–52.