Pencarian Solusi *Tower of Hanoi* Menggunakan Divide and Conquer

Frederik Imanuel Louis - 13520163[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520163@std.stei.itb.ac.id

***Abstrak*—Tower of Hanoi adalah teka-teki kombinatorial yang seringkali dibahas dalam ilmu Matematika dan Informatika. Dengan induksi, dapat dibuktikan bahwa banyak langkah minimum untuk menyelesaikan permainan ini bertumbuh eksponensial dengan bertambahnya ukuran teka-teki. Algoritma divide and conquer dapat digunakan untuk mencari urutan langkah tersebut secara efisien dengan kompleksitas waktu asimptotik yang berbanding lurus dengan banyaknya langkah minimum.**

Kata Kunci—Induksi; Kompleksitas; Algoritma;

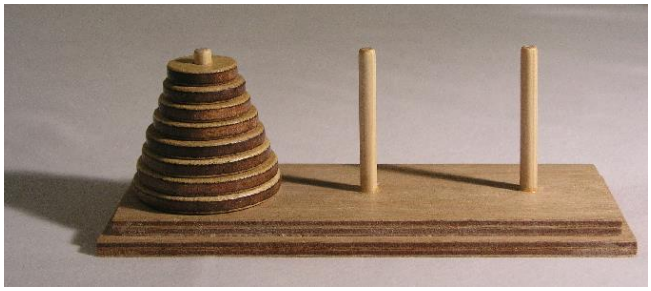
I. PENDAHULUAN

Kemampuan komputer modern telah jauh melampaui kemampuan manusia dalam melakukan komputasi sederhana. Hal tersebut memungkinkan manusia untuk menggunakan komputer untuk mensimulasikan permasalahan dalam skala yang sebelumnya tidak dapat dicapai. Salah satu aplikasi simulasi yang sering digunakan adalah simulasi permasalahan permainan kombinatorial, dimana komputasi dalam skala besar tidak jarang dibutuhkan.

Permainan kombinatorial adalah salah satu cabang ilmu Kombinatorika yang mempelajari permainan yang memiliki *perfect information*, dimana semua pemain memiliki informasi lengkap mengenai kondisi dan keadaan permainan. Selain itu, permainan kombinatorial bersifat deterministik, yang berarti semua langkah akan memiliki konsekuensi yang pasti. Hal tersebut berarti permainan yang melibatkan lemparan dadu atau keberuntungan tidak termasuk dalam permainan kombinatorial. Salah satu cabang permainan kombinatorial dimana permainan hanya dimainkan oleh satu orang disebut *combinatorial puzzle*.

Tower of Hanoi merupakan salah satu contoh *combinatorial puzzle* yang dimainkan oleh satu pemain. Pada permainan tersebut, pemain diberikan sebuah menara piringan yang harus dipindahkan dari satu pasak ke pasak lainnya, tetapi piringan yang lebih besar tidak boleh diletakkan diatas piringan yang lebih kecil, dan piringan harus dipindahkan satu per satu.

Permasalahan yang seringkali dibahas dalam ilmu Informatika adalah bagaimana urutan langkah menyelesaikan *Tower of Hanoi* menggunakan banyak langkah minimum. Pada makalah ini, akan dibahas bagaimana cara menentukan banyak langkah minimum tersebut, serta bagaimana cara mencari urutan langkah penyelesaian *Tower of Hanoi* menggunakan *Divide and Conquer*.



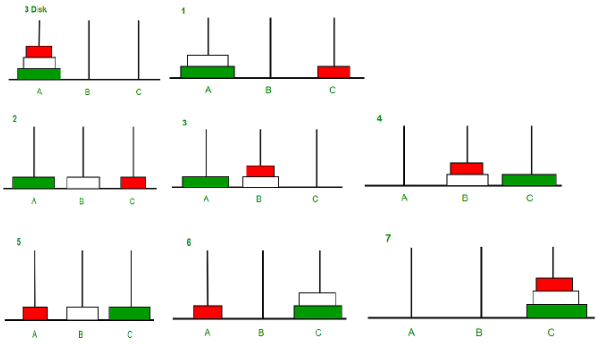
Gambar 1.1 Tower of Hanoi
Sumber: wikimedia.org

II. LANDASAN TEORI

A. *Tower of Hanoi*

Tower of Hanoi adalah sebuah permainan klasik yang bertujuan memindahkan sebuah menara dari satu pasak ke pasak lainnya. Pada permainan klasik *Tower of Hanoi*, terdapat tiga pasak A, B, dan C, serta terdapat n buah piringan yang membentuk sebuah menara pada salah satu pasak. Semua piringan memiliki ukuran yang berbeda, dan menara awal adalah piringan yang disusun dari piringan terbesar ke terkecil. Piringan hanya boleh dipindahkan satu per satu, dan saat memindahkan piringan, tidak ada piringan yang boleh diletakkan diatas piringan yang lebih kecil dari piringan tersebut. Dengan aturan tersebut, seluruh menara harus dipindahkan dari satu pasak ke pasak lainnya.

Dalam ilmu Matematika, permasalahan yang sering dibahas adalah berapa banyak langkah minimum yang diperlukan untuk menyelesaikan permainan *Tower of Hanoi* relatif terhadap banyak piringan pada menara. Kemudian, dalam dunia Informatika, permasalahan yang muncul adalah apa saja urutan langkah yang diperlukan agar permainan *Tower of Hanoi* dapat diselesaikan dengan banyak langkah minimum tersebut. Permasalahan urutan langkah tersebutlah yang akan dibahas pada makalah ini.



Gambar 2.1 Contoh Penyelesaian Tower of Hanoi
Sumber: geeksforgeeks.org

B. *Induksi Matematika*

Induksi matematika adalah salah satu teknik yang dapat digunakan untuk membuktikan suatu pernyataan matematis. Induksi dapat membuktikan suatu pernyataan yang bersifat umum dari suatu pernyataan yang bersifat khusus. Induksi melibatkan suatu basis yang menyatakan bahwa pernyataan yang ingin dibuktikan benar untuk beberapa nilai n. Kemudian, induksi membutkikan suatu langkah induksi yang membuktikan jika pernyataan benar untuk suatu nilai n, maka pernyataan juga akan benar untuk nilai lainnya.

Terdapat beberapa jenis induksi matematika berdasarkan variasi basis dan langkah induksi. Induksi lemah membuktikan basis untuk satu nilai n, dan kemudian memiliki langkah induksi yang membuktikan bahwa jika pernyataan benar untuk n, maka pernyataan juga benar untuk $n + 1$. Induksi kuat membuktikan basis benar untuk semua nilai $0 \leq i \leq n$, dan kemudian memiliki langkah induksi yang membuktikan bahwa pernyataan benar untuk $0 \leq i \leq n + 1$.

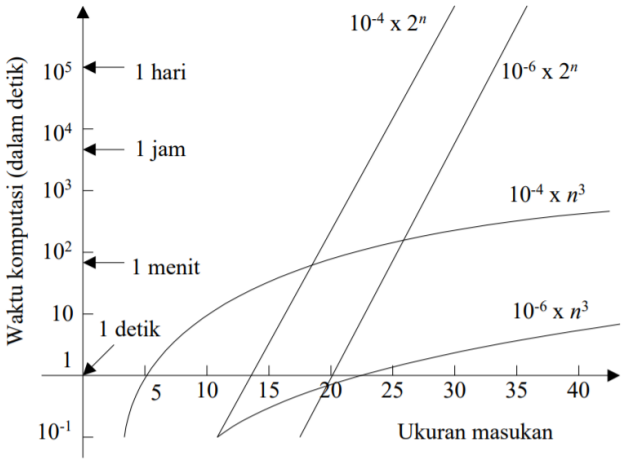
Berikut merupakan pembuktian induksi lemah pada pernyataan $n^2 \geq n$ untuk $n \geq 0$.

1. Basis:
$$n = 0 \rightarrow 0 = n^2 \geq n = 0$$
2. Langkah induksi:
$$(n+1)^2 = n^2 + 2n + 1$$
$$n^2 \geq n \rightarrow n^2 + 2n + 1 \geq 3n + 1$$
$$n \geq 0 \rightarrow 3n + 1 \geq n + 1$$
3. Kesimpulan:
Pernyataan benar untuk $n = 0$, dan jika $n^2 \geq n$ untuk suatu n benar, maka $(n + 1)^2 \geq n + 1$ benar, sehingga pernyataan benar untuk semua n bilangan asli.

Dari bukti tersebut, terlihat bahwa induksi dapat membuktikan suatu pernyataan secara tidak langsung, dengan menunjukkan bahwa suatu basis dan langkah induksi benar.

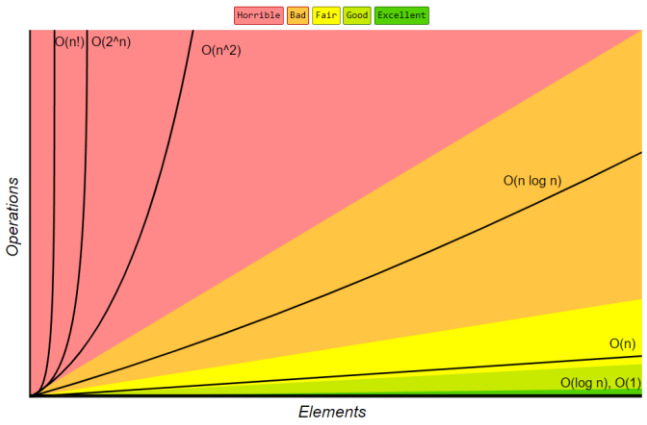
C. *Kompleksitas Waktu*

Kompleksitas waktu suatu algoritma adalah salah satu metrik pengukuran yang dapat mengukur efisiensi algoritma. Kompleksitas waktu diukur dari banyaknya operasi komputasi primitif yang dilakukan suatu algoritma. Operasi primitif yang dimaksud antara lain operasi baca/tulis (input/output), aritmetika, assignment, pemanggilan fungsi, dan sebagainya. Komputer modern dapat melakukan hingga 10^9 operasi primitif tersebut dalam satu detik.



Gambar 2.2 Grafik perbandingan ukuran masukan terhadap waktu komputasi algoritma
Sumber: Slide Materi Kuliah

Kompleksitas waktu asimptotik mengukur peningkatan kompleksitas waktu terhadap peningkatan ukuran input. Suatu algoritma umumnya dinilai dari kompleksitas waktu asimptotik yang dimilikinya. Kompleksitas waktu persis tidak diperlukan sebab jumlah operasi primitif yang dilakukan suatu algoritma pada umumnya tidak dibutuhkan untuk menilai apakah suatu algoritma feasible atau tidak. Hal tersebut dapat diperkirakan menggunakan kompleksitas waktu asimptotik, yang seringkali dilambangkan dengan notasi *Big-O*. Suatu algoritma dengan kompleksitas waktu asimptotik $O(f(N))$ dapat berjalan dalam waktu berorder paling besar $f(N)$, dengan N adalah besarnya masukan.



Gambar 2.3 Grafik perbandingan efisiensi beberapa notasi Big-O
Sumber: bigocheatsheet.com

Efisiensi satu algoritma dinilai dari orde kompleksitas waktu asimptotiknya. Semakin kecil orde yang dimilikinya, maka semakin efisien algoritma tersebut dalam menyelesaikan permasalahan dengan cepat.

Tabel 2.1 Kompleksitas beberapa algoritma umum

Kompleksitas	Algoritma
$O(1)$	Operasi primitif
$O(\log(N))$	Binary search
$O(sqrt(N))$	Cek keprimaan suatu bilangan
$O(N)$	Linear search,
$O(N\log(N))$	Merge sort, Heap sort
$O(N^2)$	Dynamic Programming, Dijkstra
$O(N^3)$	Floyd-Warshall, Perkalian Matriks
$O(2^N)$	Enumerasi subset
$O(N!)$	Enumerasi permutasi

D. Divide and Conquer

Divide and Conquer adalah salah satu metode penyelesaian masalah yang sering digunakan dalam ilmu komputer. Metode ini membagi sebuah permasalahan menjadi beberapa sub permasalahan yang serupa dengan permasalahan awal, yang kemudian akan terus dibagi lagi sampai permasalahan memiliki ukuran yang diinginkan. Terdapat beberapa tahapan dari divide and conquer, yaitu:

1. Divide
   Tahap ini membagi permasalahan menjadi beberapa sub permasalahan lebih kecil yang akan masing-masing diselesaikan.
2. Conquer
   Tahap ini akan menyelesaikan permasalahan jika ukuran permasalahan sudah memenuhi permintaan.
3. Combine
   Tahap ini akan menggabungkan solusi-solusi tiap sub problem menjadi satu solusi yang utuh.

Perhatikan bahwa tiap penyelesaian permasalahan akan menyelesaikan tiap sub masalah yang identik. Oleh karena itu, divide and conquer umumnya diimplementasikan sebagai fungsi yang rekursif, dimana fungsi *solver* masalah akan kemudian memanggil dirinya sendiri dengan parameter yang berbeda.

Salah satu algoritma populer yang sering digunakan bersama strategi divide and conquer adalah algoritma binary search yang bekerja dengan prinsip yang sama. Algoritma binary search adalah algoritma yang dapat menemukan letak suatu elemen dalam suatu array yang sudah terurut. Algoritma ini akan mengecek elemen tengah array. Jika elemen tersebut merupakan elemen yang ingin dicari, maka pencarian berhenti. Jika bukan, maka algoritma binary search akan dijalankan lagi pada kiri atau kanan array. Jika elemen tengah tersebut lebih kecil dari elemen yang ingin dicari, maka akan dilakukan binary search dari tengah sampai akhir array. Sebaliknya, jika elemen tengah tersebut lebih besar dari elemen yang ingin dicari, maka akan dilakukan binary search dari awal sampai tengah array.

```
122  //Binary Search
123  ll binser(ll l,ll r,ll val){
124      if(l>=r)return l;
125      ll mid=(l+r)/2;
126      //printf(".%d",mid);
127      if(arr[mid]==val)return mid;
128      if(arr[mid]>val&&arr[mid-1]<val)return mid;
129      if(arr[mid]>val) return binser(l,mid-1);
130      if(arr[mid]<val) return binser(mid+1,r);
131  }
```

Gambar 2.4 Algoritma Binary Search dalam C++
Sumber: Dokumen Pribadi

Pencarian dengan binary search tersebut menghasilkan algoritma dengan kompleksitas waktu asimptotik $O(\log(N))$. Algoritma tersebut jauh lebih efisien dari algoritma linear search, yang membutuhkan waktu $O(N)$. Algoritma tersebut merupakan salah satu contoh pengaplikasian divide and conquer dalam penyelesaian suatu permasalahan dengan efisien. Pembagian masalah menjadi beberapa sub masalah yang lebih kecil dapat mengurangi operasi primitif yang perlu dilakukan, sehingga masalah dapat diselesaikan dengan lebih efisien.

III. Pencarian Solusi *Tower of Hanoi* Menggunakan Divide and Conquer

A. Menentukan Banyak Langkah Minimum Penyelesaian Tower of Hanoi

Permainan *Tower of Hanoi* dengan n buah piringan dapat diselesaikan dengan cara berikut:

1. Pindahkan $n-1$ piringan teratas dari pasak asal ke pasak perantara (bukan pasak destinasi)
2. Pindahkan piringan terbesar dari pasak asal ke pasak destinasi
3. Pindahkan $n-1$ piringan teratas dari pasak perantara ke pasak destinasi

Aksi 1 dan 3 sebenarnya juga merupakan permainan *Tower of Hanoi* dengan $n-1$ piringan, dan asal atau destinasi pasak yang diubah. Perhatikan bahwa ini juga merupakan aksi paling optimal yang dapat dilakukan karena piringan terbesar harus dipindahkan minimal satu kali dari pasak asal ke pasak destinasi, dan dalam aksi tersebut, piringan terbesar memang hanya dipindahkan tepat satu kali. Misalkan banyak langkah minimum untuk menyelesaikan permainan *Tower of Hanoi* dengan banyak piringan n adalah T_n. Maka, dari ketiga aksi tersebut, dapat disimpulkan bahwa:

$$T_n = 2T_{n-1} + 1$$

Dari persamaan T_n tersebut, akan dibuktikan dengan induksi bahwa $T_n = 2^n - 1$.

1. Basis Induksi:

Jika hanya terdapat satu piringan ($n = 1$), maka jelas bahwa langkah minimum yang diperlukan untuk memindahkan menara dari asal ke tujuan adalah satu. Hal ini juga memenuhi $T_1 = 2^1 - 1 = 1$.

2. Langkah Induksi:

Asumsikan bahwa $T_k = 2^k - 1$. Maka, didapat bahwa $T_{k+1} = 2T_k + 1 = 2 * (2^k - 1) + 1 = 2^{k+1} - 1$. Maka, berdasarkan basis dan langkah induksi, klaim $T_n = 2^n - 1$ benar.

Maka, diperoleh bahwa langkah minimum yang diperlukan untuk menyelesaikan *Tower of Hanoi* dengan tinggi menara atau banyak piringan n adalah $2^n - 1$.

B. *Solusi Naif Tower of Hanoi*

Permainan *Tower of Hanoi* dapat diselesaikan secara *brute force* untuk ukuran menara (n) kecil. Solusi dapat dicari dengan melakukan pencarian semua kemungkinan langkah dari tiap posisi, dan mencetak langkah jika jawaban ditemukan. Misalkan ketiga pasakan bernama $A, B,$ dan C Dalam tiap posisi, terdapat enam kemungkinan langkah, yaitu memindahkan piringan dari $A \rightarrow B, A \rightarrow C, B \rightarrow C, B \rightarrow A, C \rightarrow A,$ dan $C \rightarrow B$. Kemudian, akan dibentuk state permainan baru dari gerakan yang dicoba. Berikut merupakan sedikit cuplikan penyelesaian *Tower of Hanoi* menggunakan brute force dalam C++:

```
string moves;
map<pair<stack<int>,stack<int>>, int> visited;

bool bruteForce(stack<int> src, stack<int> dest, stack<int> aux){
    //flag visit
    visited[mp(src,dest)]=1;
    // check if src and aux is empty (solved)
    if(src.empty() && aux.empty())return true;
    // move from src
    if(!src.empty()){
        int disk = src.top();
        src.pop();
        // to dest
        if (dest.empty()||disk<dest.top()){
            dest.push(disk);

            if(!visited[mp(src,dest)]&&bruteForce(src,dest,aux)){
                moves = "Move top disk from src to dest\n" + moves;
                return true;
            }
            dest.pop();
        }

        //to aux
        if (aux.empty()||disk<aux.top()){
            aux.push(disk);
            if(!visited[mp(src,dest)]&&bruteForce(src,dest,aux)){
                moves = "Move top disk from src to aux\n" + moves;
                return true;
            }
            aux.pop();
        }
        src.push(disk);
    }
```

Gambar 3.1 Cuplikan fungsi bruteForce dalam penyelesaian Tower of Hanoi dengan algoritma Brute Force dalam C++
Sumber: Dokumen Pribadi

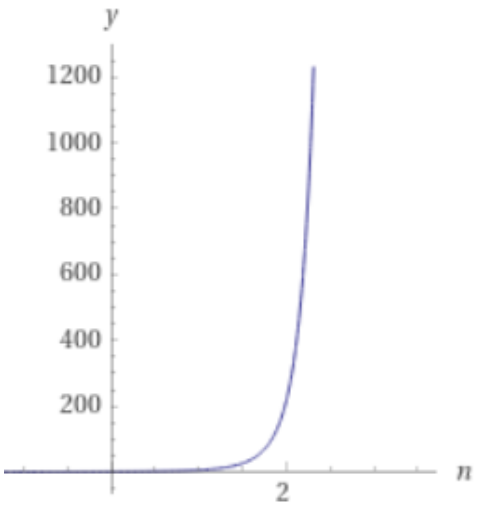
```
int main(){
    int n;
    cout<<"Masukkan banyak piringan: "; cin>>n;
    stack<int> src;
    for(int i=n; i>0; i--)src.push(i);
    stack<int> dest;
    stack<int> aux;
    cout<<"Solusi:"<<endl;
    auto start = high_resolution_clock::now();
    bruteForce(src, dest, aux);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout<<moves;
    cout<<"Waktu pemrosesan: "<<duration.count()<<" mikrosekon";
}
```

Gambar 3.2 Cuplikan fungsi main dalam penyelesaian Tower of Hanoi dengan algoritma Brute Force dalam C++
Sumber: Dokumen Pribadi

```
Masukkan banyak piringan: 3
Solusi:
Move top disk from src to dest
Move top disk from src to aux
Move top disk from dest to src
Move top disk from src to aux
Move top disk from src to dest
Move top disk from aux to src
Move top disk from src to dest
Move top disk from aux to src
Move top disk from dest to src
Move top disk from src to aux
Move top disk from src to dest
Move top disk from aux to src
Move top disk from src to dest
Waktu pemrosesan: 1001 mikrosekon
--------------------------------
Process exited after 0.4685 seconds with return value 0
Press any key to continue . . .
```

Gambar 3.3 Solusi Tower of Hanoi untuk n=3
Sumber: Dokumen Pribadi

Karena tiap iterasi harus mengecek hingga enam langkah, maka algoritma ini memiliki kompleksitas worst case $O(6^{banyak\ langkah})$. Jika kita mengasumsikan bahwa algoritma dapat menemukan solusi terpendek, maka algoritma *brute force* akan memiliki kompleksitas worst case $O(6^{2^n-1})$ dengan n adalah banyak piringan menara Hanoi. Terlihat bahwa algoritma *brute force* ini sangat tidak efisien, dimana algoritma hanya *feasible* untuk komputer modern untuk $n < 8$. Tetapi, jika pengecekan selalu benar, maka algoritma ini akan memiliki kompleksitas dengan best case $O(banyak\ langkah)$, dimana banyak langkah minimum yang mungkin adalah $2^n - 1$. Maka, algoritma brute force ini mempunyai kompleksitas waktu best case $O(2^n - 1)$ dan worst case $O(6^{2^n-1})$.



Gambar 3.4 Plot banyak operasi primitif algoritma Brute Force terhadap ukuran masukan
Sumber: Dokumen Pribadi

C. Pencarian Solusi Tower of Hanoi Menggunakan Divide and Conquer

Solusi *Tower of Hanoi* dapat dicari secara lebih efektif menggunakan teknik divide and conquer. Seperti pada pembuktian langkah minimum solusi *Tower of Hanoi*, permainan *Tower of Hanoi* dengan n buah piringan dapat diselesaikan dengan cara berikut:

1. Pindahkan $n-1$ piringan teratas dari pasak asal ke pasak perantara (bukan pasak destinasi)
2. Pindahkan piringan terbesar dari pasak asal ke pasak destinasi
3. Pindahkan $n-1$ piringan teratas dari pasak perantara ke pasak destinasi

Langkah tersebut dapat diterjemahkan ke algoritma divide and conquer dengan fungsi pencari solusi *Tower of Hanoi*: $hanoi(n, src, dest, aux)$, dengan rincian:

1. Divide
   Jika $n \geq 2$, selesaikan tiap sub permasalahan berikut:
   a. $hanoi(n-1, src, aux, dest)$
   b. $hanoi(1, src, dest, aux)$
   c. $hanoi(n-1, aux, dest, src)$
2. Conquer
   Jika $n = 1$, pindahkan piringan dari src ke $dest$.
3. Combine
   Gabung seluruh solusi menjadi langkah yang terurut.

Jika pemanggilan fungsi dilakukan secara terurut sesuai pada tahap Divide, dan jika solusi hanya ingin dicetak, maka masing-masing langkah dapat langsung dicetak pada tahap Conquer.

```cpp
#include<bits/stdc++.h>
#include<chrono>
using namespace std;
using namespace std::chrono;

void dnc(int n, const string& src, const string& dest, const string& aux){
    if(n==1){
        cout<<"Move top disk from "<<src<<" to "<<dest<<endl;
        return;
    }
    dnc(n-1, src, aux, dest);
    dnc(1, src, dest, aux);
    dnc(n-1, aux,dest,src);
}

int main(){
    int n;
    string src = "src", dest = "dest", aux = "aux";
    cout<<"Masukkan banyak piringan: "; cin>>n;
    cout<<"Solusi:"<<endl;
    auto start = high_resolution_clock::now();
    dnc(n, src, dest, aux);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout<<"Waktu pemrosesan: "<<duration.count()<<" mikrosekon";
}
```

Gambar 3.5 Penyelesaian Tower of Hanoi dengan algoritma Divide and Conquer dalam C++
Sumber: Dokumen Pribadi

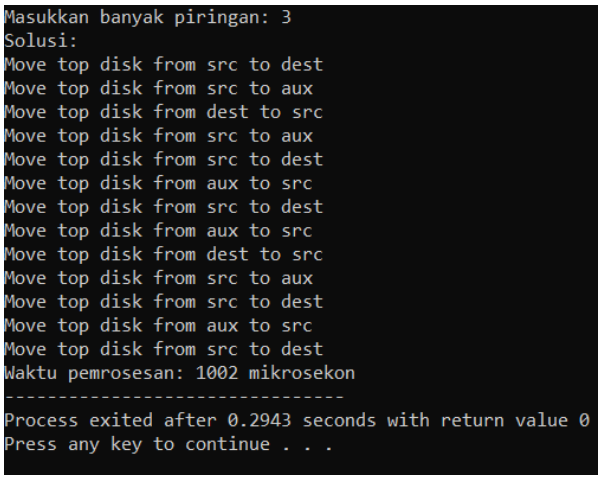
Solusi divide and conquer memberikan kompleksitas waktu asimptotik $O(banyak\ langkah)$, yaitu $O(2^n - 1)$. Orde solusi ini jauh lebih efisien dari solusi yang dihasilkan secara brute force, dimana solusi ini masih feasible hingga $n < 30$.

```
Masukkan banyak piringan: 5
Solusi:
Move top disk from src to dest
Move top disk from src to aux
Move top disk from dest to aux
Move top disk from src to dest
Move top disk from aux to src
Move top disk from aux to dest
Move top disk from src to dest
Move top disk from src to aux
Move top disk from dest to aux
Move top disk from dest to src
Move top disk from aux to src
Move top disk from dest to aux
Move top disk from src to dest
Move top disk from src to aux
Move top disk from dest to aux
Move top disk from src to dest
Move top disk from aux to src
Move top disk from aux to dest
Move top disk from src to dest
Move top disk from aux to src
Move top disk from dest to aux
Move top disk from dest to src
Move top disk from aux to src
Move top disk from aux to dest
Move top disk from src to dest
Move top disk from src to aux
Move top disk from dest to aux
Move top disk from src to dest
Move top disk from aux to src
Move top disk from aux to dest
Move top disk from src to dest
Waktu pemrosesan: 13069 mikrosekon
--------------------------------
Process exited after 0.4288 seconds with return value 0
Press any key to continue . . .
```

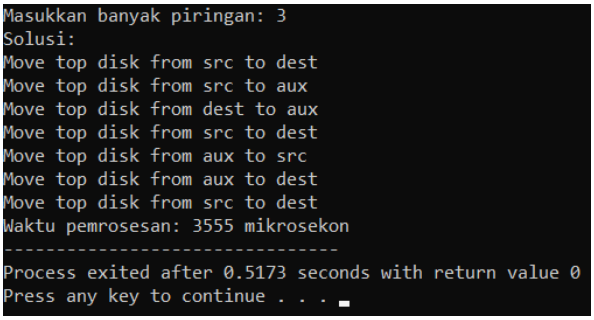
Gambar 3.6 Solusi optimal Tower of Hanoi untuk n=5
Sumber: Dokumen Pribadi

D. Perbandingan algoritma Divide and Conquer dan Brute Force dalam penyelesaian Tower of Hanoi

Untuk nilai n kecil, dapat terlihat bahwa tidak terdapat perbedaan jauh dalam kinerja algoritma brute force dan divide and conquer. Untuk beberapa kasus, algoritma brute force bahkan dapat memiliki execution time yang lebih rendah meskipun menghasillkan solusi dengan banyak langkah yang lebih banyak. Hal tersebut disebabkan pemanggilan fungsi yang dilakukan algoritma brute force bisa jadi lebih sedikit dibanding algoritma divide and conquer jika langkah yang diperiksa kebetulan langsung menuju ke solusi. Hal tersebut berbeda dengan divide and conquer yang melakukan banyak panggil fungsi yang sudah pasti.

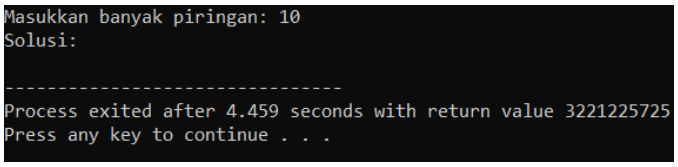


Gambar 3.7 Kinerja algoritma Brute Force untuk n=3
Sumber: Dokumen Pribadi

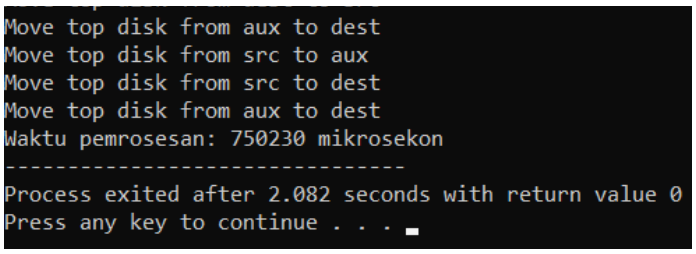


Gambar 3.8 Kinerja algoritma Divide and Conquer untuk n=3
Sumber: Dokumen Pribadi

Namun, untuk n yang lebih besar, algoritma Brute Force sudah tidak bisa menghasilkan solusi. Hal tersebut mengillustrasikan pengaruh kompleksitas waktu asimptotik seiring bertambahnya ukuran masukan.



Gambar 3.9 Kinerja algoritma Brute Force untuk n=10
Sumber: Dokumen Pribadi



Gambar 3.10 Kinerja algoritma Divide and Conquer untuk n=10
Sumber: Dokumen Pribadi

IV. KESIMPULAN

Permasalahan *Tower of Hanoi* dapat diselesaikan dengan banyak langkah minimum $2^n - 1$ dengan n adalah banyak piringan. Dengan algoritma divide and conquer, kita dapat mencari urutan langkah tersebut dengan kompleksitas waktu asimptotik $O(2^n - 1)$. Solusi tersebut merupakan solusi yang jauh lebih efisien dari solusi brute force yang memiliki kompleksitas waktu asimptotik terburuk $O(6^{2^n - 1})$.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa,
2. orang tua penulis,
3. Bapak dan Ibu dosen pengampu mata kuliah Strategi Algoritma IF2251, dan
4. teman-teman penulis

yang telah mendukung penulis selama proses penyusunan makalah ini.

VI. REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf Diakses pada 14 Mei 2022
[2] https://www.bigocheatsheet.com/. Diakses pada 14 Mei 2022
[3] https://devopedia.org/algorithmic-complexity. Diakses pada 15 Mei 2022
[4] Sriram, Pranav A. (2014), Olympiad Combinatorics.
[5] *Dutta, Prajit K. (1999), Strategies and games: theory and practice*, MIT Press, ISBN 978-0-262-04169-0.
[6] Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). *Competitive programming 3*. Lulu Independent Publish.
[7] Wengrow, J. (2020). *A Common-Sense Guide to Data Structures and Algorithms*. Pragmatic Bookshelf.

PERNYATAAN

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

Bandung, 21 Mei 2022

Frederik Imanuel Louis, 13520163