Analisis Kompleksitas Waktu dari beberapa Algoritma pada Mesin Turing

Abdul Rafi Radityo Hutomo - 13522089
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522089@std.stei.itb.ac.id

***Abstract*—Mesin Turing adalah sebuah permodelan abstrak dari komputer modern yang menjadi standar dari kapabilitas sebuah mesin komputasi dalam ilmu komputer. Mesin Turing dapat menyelesaikan masalah sama halnya dengan mesin komputasi atau komputer pada umumnya. Mesin Turing memiliki mekanisme kerja sendiri yang lebih sederhana daripada komputer pada umumnya. Pada makalah ini dilakukan analisis kompleksitas waktu dari berbagai algoritma yang umum digunakan dengan implementasinya dibuat menggunakan mesin Turing**

***Kata Kunci*—Kompleksitas, Mesin Turing, Algoritma, Automata**

I. PENDAHULUAN

Mesing Turing adalah sebuah konsep abstrak yang dikenalkan oleh Alan Turing pada tahun 1936. Mesin Turing didesain oleh Alan Turing untuk merepresentasikan cara kerja komputer dalam bentuk yang paling dasarnya. Meskipun konsepnya yang minimalis, mesin Turing dapat mencakup seluruh algoritma dan kapabilitas yang dapat diimplementasikan oleh sebuah "komputer". Dalam pengembangan ilmu komputer, Mesin Turing juga dijadikan sebbuah standar, dengan sebuah mesin komputasi diberi istilah 'Turing Complete' apabila mesin tersebut dapat menghitung segala fungsi yang dapat dilakukan oleh sebuah mesin Turing. Oleh karena itu, mengingat besarnya peran mesin Turing dalam perkembangan ilmu komputasi, dibutuhkan pemahaman yang baik mengenai sifat kerja dan karakteristik dari mesin turing.

Gambar 1 : Foto Alan Turing

Analisis Kompleksitas adalah salah satu metode analisis algoritma yang sering digunakan dalam memilih sebuah algoritma. Pada dasarnya, analisis kompleksitas menggambarkan bagaimana sebuah fungsi meningkat nilainya seiring dengan meningkatnya nilai dari parameter fungsi tersebut. Analisis kompleksitas digunakan dalam ilmu komputer untuk menganalisis waktu dan memori yang dibutuhkan dari sebuah algoritma dan menjadi ukuran kesangkilan algoritma tersebut. Secara sederhananya, analisis kompleksitas merupakan salah satu alat untuk seorang informatikawan dalam membandingkan beberapa algoritma dalam menyelesaikan sesuatu permasalahan dan menentukan algoritma yang paling efisien dalam menyelesaikan permasalahan tersebut.

Secara praktik, mesin turing tidak digunakan untuk melakukan proses komputasi. Namun, sebagai representasi yang paling sederhana dari sebuah komputer, mempelajari dan menganalisis implementasi beberapa algoritma dalam mesin Turing dapat meningkatkan pemahaman dan pengetahuan dunia ilmu komputer mengenai mesin Turing dan permasalahan komputasi dan komputabilitas itu sendiri.

II. LANDASAN TEORI

A. Mesin Turing

Sebuah mesin Turing, yang dikonseptualisasikan oleh matematikawan dan logikawan Alan Turing pada tahun 1936, adalah model komputasi teoretis yang menjadi dasar dari ilmu komputer modern. Ini berfungsi sebagai abstraksi dasar untuk memahami batasan dan kemampuan komputasi. Konsep ini dinamai sesuai dengan Alan Turing, yang memainkan peran penting dalam membentuk bidang ilmu komputer teoretis.

Pada intinya, mesin Turing terdiri dari pita tak terhingga yang dibagi menjadi sel, kepala baca/tulis yang dapat bergerak ke kiri atau kanan sepanjang pita, dan unit kontrol yang mengatur perilaku mesin. Pita berfungsi sebagai masukan dan keluaran, dan tak terbatas untuk memungkinkan komputasi yang tidak terbatas.

Notasi untuk mesin Turing sering kali direpresentasikan sebagai 7-tuple:

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Q: himpunan 'states' atau keadaan mesin dengan jumlah

berhingga,

Σ: alfabet masukan, yang merupakan himpunan berhingga dari simbol,

Γ: alfabet pita, yang mencakup Σ dan simbol tambahan untuk pita,

δ: fungsi transisi yang mendefinisikan perilaku mesin,

q_0: keadaan awal atau state awal mesin,

B: simbol pada pita yang menandakan blank atau karakter kosong,

F: Himpunan 'state' atau keadaan yang menerima string,

Mesin Turing beroperasi melalui siklus transisi keadaan yang dipandu oleh fungsi transisi δ. Pada setiap saat tertentu, mesin berada dalam 'state' atau keadaan tertentu, membaca simbol di sel pita saat ini dan kemudian memperbarui keadaan dan pita berdasarkan δ.

Proses ini berlanjut hingga mesin mencapai keadaan penerimaan atau penolakan, pada saat itu mesin berhenti. Jika mesin masuk ke keadaan penerimaan, masukan dianggap diterima; jika masuk ke keadaan penolakan, masukan ditolak.

Bahasa dari sebuah mesin Turing mengacu pada himpunan dari semua rangkaian simbol yang dapat diterima oleh mesin Turing tersebut. Dalam konteks mesin Turing, sebuah bahasa pada dasarnya merupakan kumpulan dari rangkaian simbol yang berasal dari suatu alfabet. Alfabet merupakan himpunan simbol yang digunakan oleh mesin Turing pada pita.

A. Alfabet (Σ): Himpunan simbol yang dapat digunakan oleh mesin Turing. Termasuk di dalamnya adalah simbol-simbol masukan, serta simbol-simbol tambahan pada pita.
B. Rangkaian Simbol: Sebuah rangkaian adalah urutan simbol dari alfabet. Rangkaian ini mewakili masukan yang diproses oleh mesin Turing.
C. Bahasa (L): Bahasa dari sebuah mesin Turing, dilambangkan sebagai L, merupakan himpunan dari semua rangkaian simbol yang dapat diterima oleh mesin Turing tersebut. Secara formal, L adalah suatu sub-himpunan dari himpunan semua rangkaian simbol yang mungkin dari alfabet (Σ^*).

Mesin Turing menerima sebuah rangkaian simbol jika, ketika memproses rangkaian tersebut sesuai dengan aturan transisinya(δ), mesin tersebut mencapai suatu keadaan penerimaan. Himpunan dari semua rangkaian yang diterima membentuk bahasa dari mesin Turing.

Secara matematis, jika M adalah sebuah mesin Turing, maka bahasa L(M) didefinisikan sebagai:

$$L(M) = \{w \in \Sigma^* \mid M \text{ menerima } w\}$$

Dengan, w mewakili sebuah rangkaian simbol alfabet, dan M menerima w jika, ketika w diberikan sebagai masukan kepada M, mesin tersebut memasuki suatu keadaan penerimaan.

Dalam konteks mesin Turing, sebuah bahasa juga dapat didefinisikan berdasarkan penerimaan melalui berhenti tanpa keberadaan keadaan penerimaan yang spesifik 'Acceptance by Halting'. Ini mengimplikasikan bahwa mesin "menerima" suatu rangkaian ketika tidak ada langkah lebih lanjut yang dapat diambil, bukan berdasarkan program yang mencapai suatu keadaan penerimaan yang ditentukan. Ini sering dikaitkan dengan konsep mesin Turing yang mengenali bahasa melalui komputasi total.

Dalam konteks ini, bahasa yang dikenali oleh mesin Turing adalah himpunan dari semua rangkaian simbol di mana mesin berhenti, tanpa memandang apakah mesin berhenti dalam suatu keadaan penerimaan. Secara matematis, jika M adalah mesin Turing, bahasa $L_{halt}(M)$ berdasarkan penerimaan melalui berhenti tanpa keberadaan keadaan penerimaan yang spesifik dapat didefinisikan sebagai:

$$L_{halt}(M) = \{ w \in \Sigma^* \mid M \text{ berhenti pada masukan } w\}$$

Pada konsep ini, terdapat penekanan pada berhentinya, atau *halting* mesin itu sendiri, dan penerimaan tersirat dalam penghentian komputasi. Pendekatan ini memberikan perspektif yang berbeda tentang bagaimana bahasa dapat dikenali oleh mesin Turing.

Meskipun, pada saat ini konsep mesin Turing dianggap sebagai mesin penentu diterima atau ditolaknya sebuah rangkaian simbol, visi awal Alan Turing terhadap mesin Turing adalah sebuah komputer dari mesin komputasi integer dengan integer direpresentasikan sebagai kumpulan 0 atau 1 yang dirangkai secara kontigu dan hasil komputasi didapatkan dari hasil perubahan panjang pita oleh mesin atau penulisan blok baru pada posisi yang berbeda pada pita.

Dengan memandang mesin Turing sebagai sebuah mesin komputasi, dapat diimplementasikan beberapa algoritma-algoritma yang umum digunakan dalam proses komputasi.

B. Analisis Kompleksitas

Dalam menentukan algoritma untuk menyelesaikan sebuah permasalahan, tidak cukup untuk menilai benar atau salahnya hasil dari algoritma tersebut, melainkan juga perlu dinilai efisiensi dari algoritma tersebut. Dalam menentukan efisiensi suatu algoritma dapat ditinjau berdasarkan waktu yang dibutuhkan dalam eksekusi program tersebut dan besar memori yang diperlukan untuk mengeksekusi algoritma tersebut.

Dalam mengukur kecepatan suatu algoritma, ada beberapa pendekatan yang bisa diambil. Salah satu pendekatan yang paling sederhana tetapi kurang akurat adalah dengan menghitung waktu eksekusi dari program tersebut dimulai hingga menghasilkan keluarannya. Namun, waktu eksekusi tentunya akan berbeda tergantung dengan perangkat keras yang digunakan serta kondisi perangkat keras yang digunakan pada waktu tersebut. Sehingga, metode tersebut kuranglah efektif dalam mengukur kecepatan suatu algoritma karena hasilnya tidak cukup umum untuk dijadikan acuan dan dibandingkan dengan algoritma lainnya.

Pada praktiknya, seorang programmer akan mengukur kecepatan suatu program dengan menghitung jumlah langkah dasar yang perlu diekseskusi sebuah program untuk menyelesaikan suatu permasalahan.

Jumlah langkah dasar yang dibutuhkan untuk menyelesaikan

suatu masalah tentunya tidak akan konstan, melainkan akan bergantung dengan besarnya masukkan yang diterima. Pada makalah ini, digunakan notasi T(n) untuk menyatakan fungsi yang menghitung jumlah operasi dasar yang dibutuhkan suatu algoritma untuk menyelesaikan eksekusinya yang bergantung dengan ukuran parameter n. Nilai n pada kasus ini dapat berupa sebuah angka yang diinput pengguna, panjang rangkaian simbol, panjang array atau jumlah elemen yang diproses.

Dalam melakukan analisis kompleksitas algoritma, seringkali tidak dibutuhkan nilai eksak dari langkah yang dibutuhkan untuk menyelesaikan algoritma tersebut. Pada praktiknya para pemrogram menggunakan notasi *Big O* yang menunjukkan laju meningkatnya nilai dari suatu fungsi seiring meningkatnya input atau parameternya. Langkah penyelesaian suatu algoritma juga tentunya dapat bervariasi bergantung pada data yang diprosesnya. Sebagai contoh, suatu algoritma pencarian dalam sebuah array akan membutuhkan langkah yang lebih singkat ketika elemen yang dicari berada pada indeks awal array, tetapi di sisi lain akan membutuhkan langkah yang lebih banyak ketika elemen yang dicari ada di akhir array. Dalam menangani kasus tersebut nilai *Big O* yang digunakan biasanya adalah pada kasus terburuk, yaitu kasus yang membuat langkah yang dibutuhkan untuk menyelesaikan algoritma tersebut paling banyak.

Berikut diberikan beberapa nilai *Big O* yang sering ditemui dalam analisis kompleksitas suatu algoritma diurutkan dari yang tercepat hingga yang paling lambat beserta contoh algoritmanya.

Tabel 1 : Beberapa kompleksitas algoritma

Notasi	Tipe	Contoh Algoritma
O(1)	Konstan	Mengakses elemen array
O(log n)	Logaritmik	Binary Search pada array terurut
O(n)	Linear	Linear Search
O(n log(n))	Linearitmik	Merge Sort
$O(n^2)$	Kuadratik	Bubble Sort
$O(c^n)$	Eksponensial	Travelling Salesman Problem menggunakan Dynamic Programming
O(n!)	Faktorial	Travelling Salesman Problem menggunakan brute force

C. Kompleksitas Dari Sebuah Mesin Turing

Kompleksitas dari sebuah mesin Turing diukur dari jumlah langkah pergeseran sel yang dilakukan oleh mesin tersebut. Pergeseran sel pada mesin turing akan terjadi setiap kali dilakukan perubahan keadaan(*state*), isi pita, dan juga sel yang ditunjuk oleh mesin Turing tersebut. Perpindahan tersebut dilakukan sesuai dengan fungsi transisi (δ) yang mendefinisikan mesin Turing tersebut.

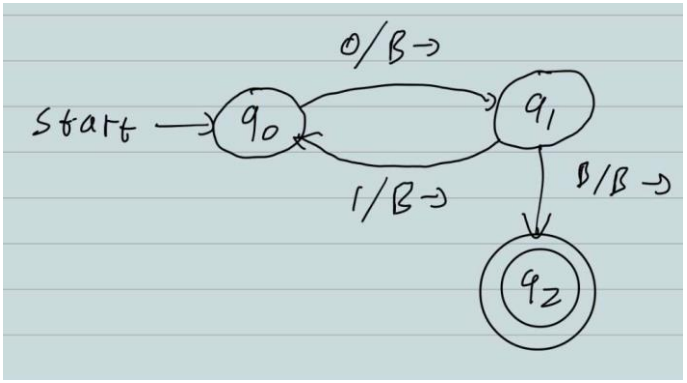
Pada makalah ini, mesin Turing akan direpresentasikan berdasarkan isi 7-tuple nya dan juga fungsi transisinya. Fungsi transisi itu sendiri akan direpresentasikan dengan tabel sebagai berikut

Tabel 2 : Contoh tabel fungsi transisi

States	Symbol		
	0	1	B
→q0	(q1, B, →)	-	-
q1	-	(q0, B, →)	(q2, B, →)
q2*	-	-	-

Dengan kolom states menyatakan keadaan mesin Turing dan symbol menyatakan symbol yang ditunjuk oleh mesin Turing tersebut. Setiap baris menunjukkan transisi pada mesin Turing dari sebuah keadaan tertentu dengan symbol tertentu menuju ke keadaan baru yang direpresentasikan dengan 3-tuple (S, A, D), dengan S adalah state yang baru, A adalah simbol baru yang menggantikan simbol sebelumnya dan D adalah arah pergerakkan sel yang ditunjuk oleh mesin Turing.

Selain itu, mesin Turing juga dapat divisualisasikan dengan diagram Transisi, berikut representasi mesin Turing pada tabel sebelumnya dalam bentuk diagram transisi.



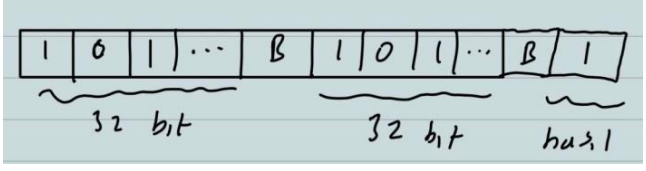
Gambar 2 : Contoh Diagram Transisi

III. ALGORITMA PADA MESIN TURING

Pada makalah ini, mesin Turing akan digunakan untuk memproses integer unsigned 32-bit yang direpresentasikan dalam binary. Kemudian, akan dibuat beberapa mesin Turing untuk melakukan algoritma yang umum digunakan dalam komputer.

A. isEqual

Permasalahan isEqual adalah mesin Turing menerima pita berisi unsigned integer 32-bit dengan representasi biner, yang diikuti sebuah blank karakter dan diikuti integer 32-bit lainnya, mesin Turing akan mencetak 0 jika kedua integer tidak sama, atau 1 jika kedua integer sama setelah integer kedua dengan dipisahkan sebuah blank sesuai dengan illustrasi



Gambar 3 : Illustrasi isEqual

Untuk menyelesaikan permasalahan tersebut didapat mesin Turing tanpa keadaan menerima :

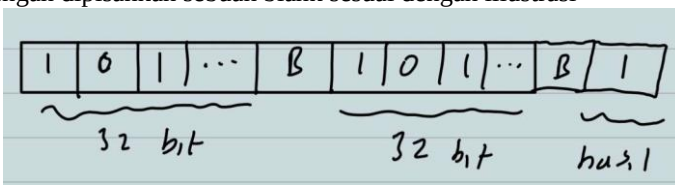
M = ({n1, n2=0, n2=1, R=0, R=1, L2, L1, Eq, Neq, REq1, REq2, RNeq, Done}, {0, 1}, {0, 1, X, B}, δ, n1, B)

Tabel 3 : Fungsi Transisi M_{Eq}

States				
	0	1	X	B
→n1	(R=0, X, →)	(R=1, X, →)	-	(REq1, X, →)
n2=0	(L2, X, ←)	(RNeq, X, →)	(n2=0, X, →)	-
n2=1	(RNeq, X, →)	(L2, X, ←)	(n2=1, X, →)	-
R=0	(R=0, 0, →)	(R=0, 1, →)	-	(n2=0, B, →)
R=1	(R=1, 0, →)	(R=1, 1, →)	-	(n2=1, B, →)
L2	-	-	(L2, X, ←)	(L1, B, ←)
L1	(L1, 0, ←)	(L1, 1, ←)	(n1, X, →)	-
Eq	-	-	-	(Done, 1, →)
Neq	-	-	-	(Done, 0, →)
REq1	-	-	(REq1, X, →)	(REq2, B, →)
REq2	-	-	(REq2, X, →)	(Eq, B, →)
RNeq	(RNeq, X, →)	(RNeq, X, →)	(RNeq, X, →)	(Neq, B, →)
Done	-	-	-	-

B. isGreaterThan

Permasalahan isGreaterThan adalah mesin Turing menerima pita berisi unsigned integer 32-bit dengan representasi biner, yang diikuti sebuah blank karakter dan diikuti integer 32-bit lainnya, mesin Turing akan mencetak 1 jika kedua integer pertama lebih besar dari pada integer kedua atau 0 jika tidak dengan dipisahkan sebuah blank sesuai dengan illustrasi



Gambar 4 : Illustrasi isGreaterThan

Untuk menyelesaikan permasalahan tersebut didapat mesin Turing tanpa keadaan menerima :

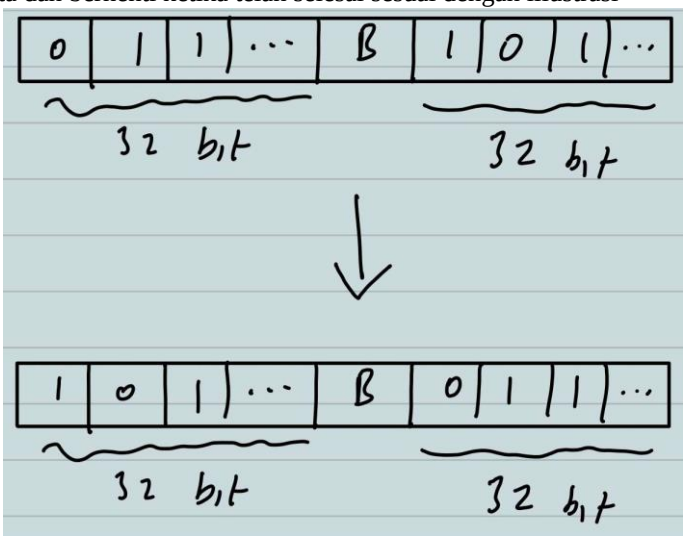
M = ({n1, n2=0, n2=1, R=0, R=1, L2, L1, G, Ng, RNg1, RNg2, Rg, Done}, {0, 1}, {0, 1, X, B}, δ, n1, B)

Tabel 4 : Fungsi Transisi M_{Ge}

States				
	0	1	X	B
→n1	(R=0, X, →)	(R=1, X, →)	-	(RNg1, X, →)
n2=0	(L2, X, ←)	(RNg2, X, →)	(n2=0, X, →)	-
n2=1	(Rg, X, →)	(L2, X, ←)	(n2=1, X, →)	-
R=0	(R=0, 0, →)	(R=0, 1, →)	-	(n2=0, B, →)
R=1	(R=1, 0, →)	(R=1, 1, →)	-	(n2=1, B, →)
L2	-	-	(L2, X, ←)	(L1, B, ←)
L1	(L1, 0, ←)	(L1, 1, ←)	(n1, X, →)	-
G	-	-	-	(Done, 1, →)
Ng	-	-	-	(Done, 0, →)
RNg1	-	-	(REq1, X, →)	(REq2, B, →)
RNg2	-	-	(REq2, X, →)	(Eq, B, →)
Rg	(Rg, X, →)	(Rg, X, →)	(Rg, X, →)	(G, B, →)
Done	-	-	-	-

C. Swap

Permasalahan Swap adalah mesin Turing menerima pita berisi unsigned integer 32-bit dengan representasi biner, yang diikuti sebuah blank karakter dan diikuti integer 32-bit lainnya, mesin Turing akan menukar posisi kedua integer tersebut pada pita dan berhenti ketika telah selesai sesuai dengan illustrasi



Gambar 5 : Illustrasi Swap

Untuk menyelesaikan permasalahan tersebut didapat mesin Turing tanpa keadaan menerima :

M = ({n1, n2=0, n2=1, R=0, R=1, L2=0, L1=0, L2=1, L1=1 L, Sweep1, Sweep2, Done}, {0, 1}, {0, 1, X, Y, B}, δ, n1, B)

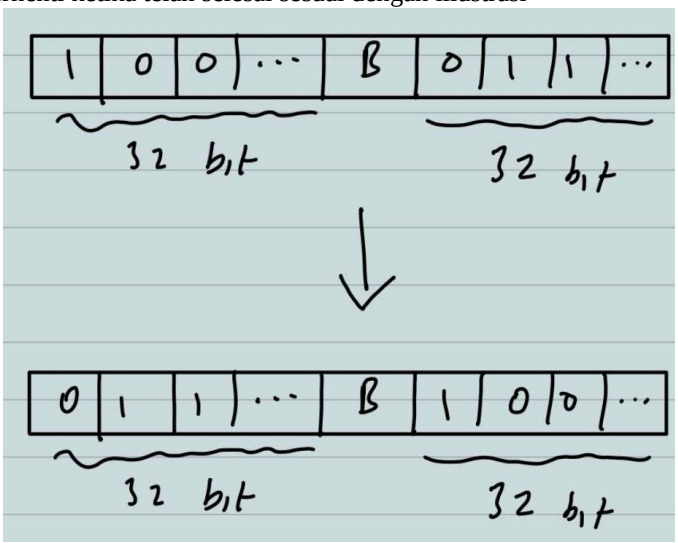
Tabel 5 : Fungsi Transisi M_{swap}

States	Symbol				
	0	1	X	Y	B
→n1	(R=0, X, →)	(R=1, Y, →)	-		(L, B, ←)

n2=0	(L2Eq, X, ←)	(L2=1, X, ←)	(n2=0, X, →)	(n2=0, Y, →)	-
n2=1	(L2=0, Y, ←)	(L2Eq, Y, ←)	(n2=1, X, →)	(n2=1, Y, →)	-
R=0	(R=0, 0, →)	(R=0, 1, →)	-	-	(n2=0, B, →)
R=1	(R=1, 0, →)	(R=1, 1, →)	-	-	(n2=1, B, →)
L2=0	-	-	(L2=0, X, ←)	(L2=0, Y, ←)	(L1=0, B, ←)
L1=0	(L1=0, 0, ←)	(L1=0, 1, ←)	(n1, X, →)	(n1, X, →)	-
L2=1	-	-	(L2=1, X, ←)	(L2=1, Y, ←)	(L1=1, B, ←)
L1=1	(L1=1, 0, ←)	(L1=1, 1, ←)	(n1, Y, →)	(n1, Y, →)	-
L	-	-	(L, X, ←)	(L, Y, ←)	(Sweep1, B, →)
Sweep 1	-	-	(Sweep1, 0, →)	(Sweep1, 1, →)	(Sweep2, B, →)
Sweep 2	-	-	(Sweep2, 0, →)	(Sweep2, 1, →)	(Done, B, →)
Done	-	-	-	-	-

D. Sort2

Permasalahan Sort2 adalah mesin Turing menerima pita berisi unsigned integer 32-bit dengan representasi biner, yang diikuti sebuah blank karakter dan diikuti integer 32-bit lainnya, mesin Turing akan menukar posisi kedua integer tersebut pada pita jika integer pertama lebih besar daripada integer kedua dan berhenti ketika telah selesai sesuai dengan illustrasi



Gambar 6 : Illustrasi Sort2

Untuk menyelesaikan permasalahan tersebut akan dilakukan penggunaan mesin Turing isGreaterThan dan mesin Turing Swap. Hasil dari penggunaan kedua mesin tersebut adalah sebagai berikut

M = ({Flag, L3, L2, L1, Done}, {0, 1}, {0, 1, X, B}, δ, n1, B)

Tabel 6 : Fungsi Transisi M_{sort2}

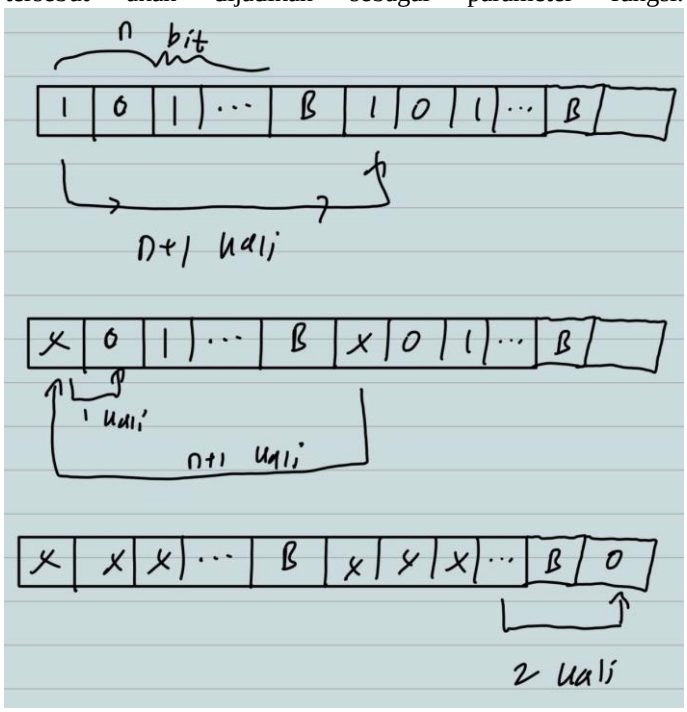
States				
	0	1	X	B
→M_{Great}	-	-	-	(Flag, B, ←)
Flag	(Done, 0, →)	(L3, 1, ←)	-	-
L3	-	-	-	(L2, B, ←)
L2	(L2, 0, ←)	(L2, 1, ←)	-	(L1, B, ←)
L1	(L1, 0, ←)	(L1, 1, ←)	-	(M_{swap}, B, →)
M_{swap}	-	-	-	(Done, B, →)
Done	-	-	-	-

Pada implementasi ini ada beberapa hal yang perlu diperhatikan. Mesin M_{Great} adalah mesin Turing isGreaterThan yang memiliki tambahan fitur pengembalian tape menjadi seperti semula dengan mekanisme yang serupa dengan mesin Swap. Kemudian, M_{swap} adalah mesin Swap dan untuk state yang berupa mesin, evaluasi fungsi transisi dilakukan dengan cara menjalankan mesin Turing tersebut dan pada state terakhir di saat mesin tersebut telah berhenti barulah fungsi transisi di evaluasi.

IV. Hasil Pembahasan

A. isEqual

Pada mesin ini, kasus terburuknya yang membutuhkan paling banyak langkah adalah kasus di saat kedua integer sama. Pada mesin Turing yang telah dibuat jumlah bit integer diambil kasus berjumlah 32, tetapi dalam analisis kompleksitas jumlah bit tersebut akan dijadikan sebagai parameter fungsi.



Gambar 7 : Illustrasi Jumlah Langkah isEqual

Dapat ditinjau bahwa pada pengecekkan setiap bit dibutuhkan n+1 langkah untuk berpindah ke posisi bit pada integer 1 ke bit dengan posisi yang sama pada integer kedua. Kemudian, dibutuhkan n+2 kali perpindahan untuk kembali ke integer 1 dan maju 1 posisi bit. Hal tersebut dilakukan sebanyak n kali, tetapi pada iterasi terakhir terdapat tambahan n + 2 pergerakan lagi sehingga diperlukan

$T(n) = (n+1 + n+2) \times n + (n + 2)$
$T(n) = 2n^2 + 4n + 2$ langkah
$T(n) = O(n^2)$

Sehingga didapat kompleksitas mesin Turing dalam melakukan fungsi isEqual adalah $O(n^2)$

B. isGreaterThan

Pada mesin Turing isGreaterThan, kasus terburuknya akan serupa dengan kasus isEqual, cara kerja kedua mesin juga sangat mirip sehingga jumlah langkah yang dibutuhkan adalah sama, yaitu $T(n) = 2n^2 + 4n + 2$ langkah dan kompleksitas waktunya juga $O(n^2)$

C. Swap

Pada mesin Turing Swap, jumlah langkah yang dibutuhkan akan konstan, karena mesin Turing tidak akan terminasi lebih cepat apapun kondisinya. Jumlah langkah yang diperlukan juga akan sama persis dengan isEqual, hingga saat dimana Swap telah menandai semua 0 dengan X dan semua 1 dengan Y. Kemudian, program akan kembali ke posisi awal yang membutuhkan n+2 langkah, kemudian mesin Turing akan bergerak sampai ujung dari integer kedua sehingga dibutuhkan 2*n bit + 1 blank = 2n + 1 langkah. Sehingga didapat nilai
$T(n) = ((n + 1) + (n + 2)) \times n + (n + 2) + (2n + 1)$
$T(n) = 2n^2 + 6n + 3$
$T(n) = O(n^2)$

Seingga didapat untuk Mesin Turing Swap, juga kompleksitasnya adalah $O(n^2)$

D. Sort2

Pada mesin Turing Sort2, dimanfaatkan mesin Turing isGreaterThan dan mesin Turing Swap, pada worst casenya dilakukan prosedur isGreaterThan dan kemudian dilakukan Swap. Setelah isGreaterThan selesai dilakukan, mesin Turing akan kembali ke posisi awal sehingga diperlukan 2n + 3 langkah tambahan. Sehingga, didapat jumlah langkah yang dibutuhkan adalah
$T(n) = T_{Mg}(n) + T_{swap}(n) + (2n + 3)$
$T(n) = 4n^2 + 10n + 5$
$T(n) = O(n^2)$

Sehingga didapat kompleksitas dari algoritma Sort2 juga adalah $O(n^2)$

V. KESIMPULAN

Dari hasil analisis kompleksitas yang dilakukan, dapat disimpulkan bahwa keterbatasan mesin Turing menyebabkannya memiliki kompleksitas waktu yang buruk. Keterbatasan yang dimaksud di sini adalah kemampuan mesin Turing yang hanya mampu membaca 1 bit di saat yang bersamaan dan keterbatasannya dalam membaca string yang harus melewati setiap bit satu per satu untuk bisa mengakses nilai bit di tempat lain.

UCAPAN TERIMA KASIH

Puji dan syukur kami ucapkan kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya yang memungkinkan penyelesaian makalah ini dengan sukses. Penulis juga ingin menyampaikan penghargaan setinggi-tingginya kepada orang tua yang senantiasa memberikan semangat dan dukungan selama perjalanan pendidikan penulis selama keberjalanan masa pendidikan di Institut Teknologi Bandung. Penulis juga ingin mengucapkan terima kasih kepada Ibu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T., selaku dosen pengampu mata kuliah matematika diskrit (IF2120) kelas K2, yang telah menjadi pembimbing, memberikan ilmu dan pengetahuan terkait materi, serta memberi kesempatan untuk mengembangkan makalah ini. Penghargaan juga kami sampaikan kepada semua rekan dan pihak lain yang turut mendukung kami, baik secara moral maupun material, walaupun tidak dapat disebutkan satu per satu.

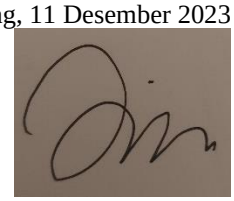
DAFTAR PUSTAKA

[1] Hopcroft, J. E., & Ullman, J. D. (1979). Introduction to automata theory, languages, and computation. Reading, Mass., Addison-Wesley.
[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/24-Kompleksitas-Algoritma-Bagian1-2023.pdf diakses pada 11 Desember 2023
[3] https://www.npg.org.uk/collections/search/person/mp18700/alan-mathison-turing
[4] https://www.donkcowan.com/blog/2013/5/11/big-o-notation

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, 11 Desember 2023



Abdul Rafi Radityo Hutomo - 13522089