*18 / 09 / 2023*

*Day – 15*

**Running Time & Big O Notation**

* **Running Time**
* Secara umum, kita selalu dihadapkan dengan pada kebutuhan untuk memilih algoritma yang paling efisien, dimana kita selalu berusaha untuk melakukan optimasi baik dalam hal waktu maupun ukuran.
* Pada bagi pertama kita sudah mempelajari tentang *binary search* dan *simple search* yang menjadi pertanyaan disini adalah berapa banyak waktu yang bisa optimalkan dengan menerapkan *binary search* ?
* Pada pendekatan pertama atau cara pertama, dimana kita menggunakan *simple search* maka kita butuh melakukan pengecekan satu persatu, dengan kata lain kalo kita memiliki 100 elemen maka akan membutuhkan 100 kali pengecekan. Dan bila terdapat 4,000,000,000 elemen, maka akan dibutuhkan sampai dengan 4,000,000,000 pengecekan.
* Jadi, jumlah pengecekan maksimal akan sama dengan jumlah elemen yang terdapat dalam list tersebut. Ini sering kali diistilahkan sebagai *linear time*.
* Kalo kita menggunakan *binary search*, maka ketika kita dihadapkan dengan 100 item / elemen, itu akan membutuhkan paling buruk sebesar 7 kali pengecekan. Dan kalau kita dihadapkan 4,000,000,000 item / elemen, maka paling banyak kita akan melakukan paling banyak 32 kali pengecekan saja.
* *Binary Search* ini akan menerapkan yang nama nya *logarithmic time* atau seringkali dikenal dengan istilah *log time*.
* Disini kita melakukan komparasi antara *binary search* dengan *simple search*.

|  |  |
| --- | --- |
| **Simple Search** | **Binary Search** |
| 100 Items => 100 Tebakan | 100 Items => 7 Tebakan |
| 4,000,000,000 Items => 4,000,000,000 Tebakan | 4,000,000,000 Items => 32 Tebakan |
| Linear Time O(n) | Logarithmic Time O(Log n) |

* **Big O Notation**
* *Big O Notation* merupakan *special notation* / notasi khusus yang bisa kita gunakan untuk mendeskripsikan seberapa cepat suatu algoritma.
* Ngapain sih kita harus memperhatikan *Big O notation* ? kenapa kita tidak langsung saja memakai saja algoritma nya ?
* Nah ternyata dalam kehidupan sehari-hari yang paling sering digunakan algoritma dikembangkan oleh orang lain. Oleh karena itu ada baik nya bagi kita untuk bisa mengidentifikasi atau membandingkan antara algoritma yang satu dengan algoritma yang lain nya.
* Dalam sesi ini, akan dibahas bagaimana sih implementasi *Big O notation* yang dapat membantu kita dalam membandingkan *running times* antar beberapa algoritma.
* **Penggunaan Algoritma Running Times Berdasarkan Tingkat Nilai Yang Berbeda**
* Nah disini kita dihadapkan pada suatu contoh kasus .
* Bob diminta untuk menuliskan algoritma pencarian yang akan digunakan oleh NASA.
* Algoritma ini akan dijalankan sewaktu melakukan pendaratan roket dibulan, dimana algoritma ini digunakan untuk membantu proses perhitungan terkait lokasi pendaratan.
* Dan untuk kasus ini, Bob hanya memiliki waktu 10 detik saja untuk melakukan proses perhitungan pendaratan.
* Disini Bob dihadapkan pada 2 algoritma yaitu *simple search* dan *binary search*.
* Algoritma yang akan digunakan ini harus cepat dan tepat.
* Disatu sisi yang kita tahu *binary search* jauh lebih cepat bila dibandingkan dengan *simple search*.
* Tetapi disisi yang lain, *simple search* itu jauh lebih mudah untuk di implementasikan, artinya sewaktu bob menuliskan algoritma *simple search* ini maka sangat kecil kemungkinan untuk membuat suatu kesalahan atau menghasilkan suatu *bug* dalam kode program nya. Dan tentu nya Bob tidak menginginkan ada nya *bugs* pada kode program yang ditulis untuk mendaratkan roket dibulan.
* Disini kita coba asumsi kan bahwa untuk melakukan pengecekan satu elemen itu dibutuhkan waktu 1ms (millisecond).
* Bob mulai melakukan kalkulasi terhadap 100 elemen
* Dengan *simple search* berarti bob harus melakukan terhadap 100 elemen, sehingga bob membutuhkan waktu sekitar 100ms.
* Sedangkan kalo bob menggunakan *binary search* maka bob hanya perlu melakukan pengecekan sekitar 7 elemen sehingga membutuhkan waktu sekitar 7ms saja.
* Secara realistis, jumlah elemen yang perlu dicari itu berkisar 1,000,000,000 elemen.
* Artinya, kalo bob menjalankan 1,000,000,000 elemen dengan menggunakan *binary search* maka bob akan membutuhkan waktu sekitar 30ms karena log2 1,000,000,000 itu berkisar 30.
* *Binary search* itukan 15 kali lebih cepat bila dibandingkan *simple search*, karena dari kasus sebelumnya kita lihat bahwa *simple search* membutuhkan waktu 100ms untuk 100 elemen sedangkan *binary search* membutuhkan waktu 7ms.
* Oleh karena itu, kalo *binary search* membutuhkan waktu 30 x 15 = 450ms, benar? Artinya, nilai ini masih dibawah threshold karena threshold nya adalah 10 detik.
* Dengan alasan inilah bob akhir nya bob memutuskan untuk menggunakan *simple search*.
* Apakah bob membuat keputusan yang tepat ?
* Ternyata bob ini memang salah besar.
* Disini ternyata kalo kita perhatikan kalo kita menggunakan *simple search* untuk melakukan pencarian pada 1,000,000,000 item maka akan dibutuhkan 1,000,000,000 ms atau dengan kata lain 11 hari.
* Yang jadi permasalahan ini adalah ternyata *run time* untuk *binary search* dan *simple search* itu kedua nya tidak berkembang dengan rate yang sama (kecepatan).

|  |  |  |
| --- | --- | --- |
| **Jumlah Elemen** | **Simple Search** | **Binary Search** |
| 100 Elemen | 100ms | 7ms |
| 10,000 Elemen | 10s | 14ms |
| 1,000,000,000 Elemen | 11 hari | 32ms |

* Seiring dengan meningkat nya jumlah item peningkatan waktu dari *binary search* itu sendiri ini bisa dibilang kecil bila dibandingkan dengan peningkatan waktu yang dibutuhkan oleh *simple search* yang bisa dibilang cukup besar.
* Artinya, ketika jumlah item yang harus diproses itu cukup besar atau bahkan sangat besar, *binary search* akan membutuhkan waktu yang jauh lebih sedikit bila dibandingkan dengan *simple search*.
* Atau dengan kata lain, *binary search* akan jauh lebih cepat bila dibandingkan dengan *simple search*.
* Oleh karena itu, tidaklah cukup untuk hanya mengetahui berapa lama suatu algoritma akan berjalan kita butuh mencari tahu bagaimana *running time* nya akan meningkat seiring dengan peningkatan ukuran nya.
* Oleh karena itu, *Big O notation* ini lebih relevan ketika kita membandingkan antar algoritma.
* **Penjelasan Big O Notation**
* *Big O notation* mengekspresikan bagaimana kecepatan suatu algoritma.
* Sebagai contoh, kita dihadapkan dengan suatu list dengan jumlah elemen n.
* *Simple Search* butuh untuk melakukan pengecekan pada setiap elemen nya, oleh karena nya ini akan membutuhkan sejumlah *n operations.* Sehingga penulisan *Big O notation* nya adalah O(n) atau *linear time*.
* *Big O notation* tidak mengekspresikan kecepatan suatu algoritma dalam satuan detik. Big O notation memungkinkan kita untuk membandingkan jumlah operasi dimana Big O notation akan mengekspresikan bagaimana kecepatan suatu algoritma berkembang atau *grow trade* suatu algoritma.
* *Binary Search* membutuhkan sejumlah *log n operations* untuk memeriksa list dengan ukuran n.
* Oleh karena itu, *running time* untuk *binary search* ketika diekspresikan dengan *Big O notation* adalah O(log n).
* Umumnya, Big O notation ini dituliskan dengan sebagai berikut.
* Big O => O(n)
* O = Big O, (n) = jumlah operasinya.
* Ekspresi semacam ini dikenal dengan istilah *Big O notation* karena kita menempatkan huruf O besar sebelum kita menuliskan jumlah operasi nya.
* **Visualisasi Beberapa Big O Run Times Studi Kasus**
* Disini ada suatu kasus sederhana yang bisa kita lakukan dengan memanfaatkan kertas dan pensil.
* Disini kita dihadapkan dengan suatu kasus untuk menggambarkan 16 kotak

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

* Algoritma semacam apa yang cukup baik untuk menggambar *grid* semacam ini?
* **Algoritma 1**
* Cara paling sederhana tentunya adalah menggambar 16 kotak satu persatu.
* Mengingat, *Big O notation* itu akan menghitung jumlah operasi.
* Untuk kasus disini, menggambarkan satu kotak berarti 1 operasi . Kalau kita harus menggambar 16 kotak berarti akan ada 16 operasi.
* Berapa banyak operasi pada satu kali gambar ?
* Berapa *running time* untuk algoritma ini ?
* Tentunya *running time* nya linier, karena jumlah operasi nya akan berbanding lurus dengan jumlah kotak yang perlu di gambar.
* **Algoritma 2**
* Disini kita akan menggambar kotak nya dengan cara melipat kertas nya.
* Pada pendekatan ini, melipat kertas satu kali itu akan dipandang sebagai satu operasi.
* Disini kita bisa lihat yah, untuk 1 operasi kita bisa menghasilkan 2 buah kotak.
* Selanjutnya, kita akan lipat kertas nya berulang kali.
* Dan setelah kita menyentuh lipatan keempat, kita akan buka kertasnya dan kita akan mendapatkan 16 *grid* yang dibutuhkan.
* Disini setiap kali kita melipat kertas, maka jumlah kotak yang terbentuk adalah 2 kali lipat dari kotak sebelumnya.
* Berapa *running time* untuk algoritma ini ?
* Jawaban nya adalah untuk algoritma pertama adalah *Big O* nya O(n) atau linier time, sedangkan untuk algoritma kedua *Big O* nya adalah O(logn).
* **Menetapkan Waktu Big O Dalam Kasus Terburuk**
* Disini kita dihadapkan pada kasus yang cukup unik.
* Semisal saja kita mau mengimplementasikan *simple search* untuk mencari seseorang pada buku telepon kita.
* Dari yang kita tahu yah, *simple search* itu membutuhkan *O(n) run time*, atau dengan kata lain pada kondisi terburuk kita perlu mengecek satu persatu setiap *entry* dari buku telepon kita.
* Dan semisal saja pada kasus ini kita butuh mencari Adit. Adit ini ternyata menempati urutan pertama pada buku telepon yang kita miliki. Artinya kita tidak perlu mengecek satu persatu *entry* mulai dari entry pertama sampai dengan *entry* terakhir karena kita sudah menemukan Adit pada *entry* yang pertama.
* Yang jadi pertanyaan disini adalah, apakah algoritma nya masih bisa dibilang sebagai *O(n) time?* Atau justru *O(1) time*? karena disini kita sudah menemukan Adit sebagai orang yang mau kita cari pada operasi pertama.
* Ternyata jawaban nya adalah simple search pada kasus pencarian nama Adit itu tetap harus dinotasikan *O(n) time.*
* Pada kasus pencarian adit, kita memang menemukan adit pada pencarian pertama (first time atau first trial). Tapi perlu diingat ini *best case* pada skenario nya.
* Yang perlu digaris bawahi disini adalah Big O notation itu berbicara mengenai skenario terburuk atau kondisi terburuk.
* Dengan kata lain, kondisi terburuk nya adalah kita harus memeriksa setiap *entry* dari buku telepon kita. Ini yang dmaksud dengan *O(n) time* atau *linier time*.
* Ini bisa digunakan sebagai suatu jaminan bahwa *simple search* itu tidak akan mungkin lebih lama atau lebih buruk dari *O(n) time*.
* **Mengenal Big O Notation Yang Umum Ditemui**
* Berikut adalah daftar dari *Big O* notation yang umum untuk ditemui dan disini diurutkan mulai dari *Big O* yang tercepat sampai dengan *Big O* yang terlambat.
* *Big O* yang tercepat dalam daftar kita adalah O(logn), atau seringkali dikenal dengan *log time* dan contoh algoritma yang memiliki *log time* adalah *binary search.*
* *Big O notation* berikutnya adalah O(n), ini seringkali dikenal sebagai *linier time* dan contoh algoritma nya adalah *simple search*.
* Lalu *Big O* ketiga disini adalah O(n \* log n) dan salah satu algoritma yang memiliki *run time* semacam ini adalah *quicksort* (dan kita akan mempelajari *quicksort* pada bab ke-4).
* *Big O* berikut nya adalah O(n2) atau seringkali dikenal dengan istilah kuadratik salah satu algoritma run time semacam ini adalah *seletion sort* (dan kita akan mempelajari *selection sort* pada bab ke-2).
* Berikut nya Big O terakhir yang disertakan pada list ini adalah O(n!) dan ini adalah run time yang sangat lambat sekali dan salah satu kasus yang dinotasikan dengan ekspresi O(n!) ini adalah *the traveling sales person problem.*
* Disini andai kata kita butuh menggambarkan 16 kotak lagi, dimana kita dihadapkan pada pilihan 5 jenis algoritma yang berbeda.
* Semisal saja disini, untuk mengerjakan 10 operasi kita membutuhkan waktu 1 detik.
* Jika kita memilih algoritma pertama, yang memiliki run time O(logn) berarti akan dibutuhkan sekitar 4 operasi untuk menggambar 16 kotak karena log 16 adalah 4.
* Perlu diingat juga yah disini, log basis disini menggunakan log basis 2.
* Karena nilai log 16 adalah 4, maka jika kita translasi kan kesini yah kita akan membutuhkan waktu sekitar 0,4 detik.
* Kenapa bisa 0,4 detik ? Karena akan terdapat 4 operasi dimana tiap detik nya kita bisa melakukan 10 operasi.
* Bagaimana jika diminta menggambarkan 1,024 kotak ? artinya disini jika kita bisa kalkulasikan adalah log 1,024 = 10 sehingga kita mendapatkan 10 operasi atau kita hanya membutuhkan waktu sekitar 1 detik saja karena 10 operasi itu bisa kita lakukan dalam 1 detik saja.
* Untuk algoritma kedua ini lebih lambat yah karena menggunakan O(n) time.
* Dengan kata lain, akan dibutuhkan 16 operasi sewaktu menggambar 16 kotak dan akan dibutuhkan 1,024 operasi untuk menggambar 1,024 kotak.
* Berapa waktu yang dibutuhkan dalam detik ?
* Dan disini kita dihadapkan pada suatu ilustrasi yang harapan nya bisa memberikan gamabaran ke kita terkait dengan Big O notation dimana sumbe x nya ini representasikan number of elements dan sumbu y nya merepresentasikan number of operation kalo kita liat disini mulai dari cepat dulu yah.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Jumlah Kotak | O(LOGn) | O(n) | O(nLOGn) | O(n2) | O(n!) |
| 16 | 0,4 detik | 1,6 detik | 6,4 detik | 25,6 detik | 66,301 tahun |
| 256 | 0,8 detik | 25,6 detik | 3,4 menit | 1,8 jam | 8,6x10 tahun |
| 1,024 | 1,0 detik | 1,7 menit | 17 menit | 1,2 hari | 5,4x10 tahun |

* Sebenarnya itu Big O notation itu tidak terbatas pada 5 notasi ini terdapat notasi yang lain hanya saja kelima notasi yang paling sering ditemui.
* Penggambaran yang ada disini ini juga merupakan bentuk penyederhanaan artinya secara realitas ploting nya tidak akan se-rapi ini.
* Ini mencoba untuk memberikan gambaran yang lebih baik kepada kalian terkait perbedaan antar kelima Big O notation ini.