**Belajar Algoritma Dari Nol Part 2**

1. **RUNNING TIME**

* Secara umum kita dihadapkan pada kebutuhan untuk algoritma yang paling efisien, dimana kita selalu berusaha untuk melakukan optimasi untuk waktu atau ukuran.
* Kembali ke binary search. Berapa banyak waktu yang bisa dioptimum-kan?
* Pada cara yang pertama dimana kita menggunakan simple search untuk mengecek setiap angka, 1 demi 1. Jika terdapat 100 list angka, itu akan menebak angka hingga 100 kali tebakan. Jika terdapat 4 miliar list angka, itu akan menebak 4 miliar kali tebakan.
* Jadi jumlah maximum pengecekan angka akan sama besar dengan list angka tersebut. Ini dapat kita sebut **linear time.**
* Berbeda dengan binary search. Jika list terdapat 100 elemen, maka akan membutuhkan paling buruk 7 tebakan. Jika list terdapat 4 miliar elemen, maka akan membutuhkan paling buruk 32 tebakan.
* Binary search menerapkan **logarithmic time.**

|  |  |
| --- | --- |
| **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) |
| O(n)  Linear time | O(log n)  Logarithmic time |

1. **BIG O NOTATION**

* Big O notation ini merupakan notasi khusus atau spesial yang dapat kita gunakan untuk mendeskripsikan seberapa cepat suatu algoritma.
* Mengapa kita harus memerhatikan Big O notation? Mengapa tidak langsung menggunakan algoritmanya saja? Karena ternyata dalam kehidupan sehari-hari kita menggunakan algoritma yang di develop orang lain, oleh karena itu sebaiknya agar kita dapat mengidentifikasi antara algoritma satu dengan algoritma lainnya.

1. **GROW (Algoritma berjalan pada tingkat waktu yang berbeda)**

**Studi Kasus:**

**[1/6]**

* Bob menulis algoritma yang akan digunakan NASA.
* Algoritma akan digunakan sewaktu roket telah mendarat di bulan, dan juga menghitung dimana roket akan melakukan pendaratan.
* Bob diberi hanya 10 detik untuk menentukan dimana akan mendarat.

**[2/6]**

* Bob dihadapkan pada 2 pilihan simple search dan binary search.
* Algoritma yang dibutuhkan harus cepat dan tepat.
* Disatu sisi, binary search lebih cepat.
* Disisi lain, simple search sangat mudah ditulis, dan sangat kecil kemungkinan bugs atau error ditemukan. Dan Bob tidak ingin ada bugs atau error di kode program untuk mendaratkan roket.

**[3/6]**

* Mari asumsikan untuk mengecek 1 elemen dibutuhkan 1 milisecond.
* Dengan hati-hati, Bob mulai melakukan proses kalkulasi terhadap 100 elemen.
* Dengan simple search, Bob harus mengecek 100 elemen, jadi akan membutuhkan waktu 100 ms untuk running.
* Namun dengan binary search, dia hanya membutuhkan 7 elemen untuk melakukan pengecekan, sehingga hanya dibutuhkan waktu 7 ms untuk running.

**[4/6]**

* Namun pada kenyataannya, list mempunyai kurang lebih 1 miliar elemen
* Jika Bob menggunakan binary search dengan 1 miliar elemen, itu akan membutuhkan 30 ms (log2 1,000,000,000 sekitar 30)
* Binary search 15 kali lebih cepat dibandingkan simple search, karena simple search akan membutuhkan 100 ms dengan 100 elemen, dan binary search membutuhkan 7 ms.
* Jadi simple search akan mengambil 30 x 15 = 450 ms, bukan? Nilai ini masih dibawah batas ambang atau *threshold* 10 detik.
* Karena alasan inilah Bob memilih simple search
* Apakah ini merupakan pilihan yang tepat?

**[5/6]**

* Tidak! Ternyata Bob salah besar.
* Running time untuk simple search dengan 1 miliar elemen akan membutuhkan 1 miliar ms, atau dengan kata lain membutuhkan 11 hari!
* Permasalahannya adalah, running times untuk binary search dan simple search tidak berkembang dengan kecepatan yang sama.

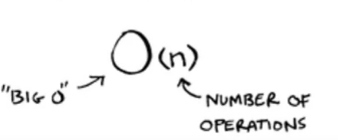
|  |  |  |
| --- | --- | --- |
|  | **Simple Search** | **Binary Search** |
| 100 elemen | 100 ms | 7 ms |
| 10,000 elemen | 10 second | 14 ms |
| 1,000,000,000 elemen | 11 days | 32 ms |

**[6/6}**

* Seiring dengan meningkatnya jumlah items, binary search membutuhkan peningkatan waktu yang dibutuhkan untuk running lebih kecil. Berbeda dengan simple search peningkatan waktu yang dibutuhkan untuk running lebih besar.
* Jika list dari angka menjadi besar, binary search akan jauh lebih cepat dibandingkan simple search.
* Oleh karena itu tidak cukup untuk mengetahui seberapa lama algoritma berjalan, tetapi kita juga butuh mengetahui waktu peningkatan running-nya seiring peningkatan ukurannya.
* Disinilah peran Big O notation dibutuhkan

1. **Penerapan Big O Notation**

* Big O notation mengajarkan kita bagaimana menentukan algoritma secepat mungkin.
* Contoh, jika kita memiliki sebuah elemen dengan ukuran *n.*
* Simple search akan mengecek setiap elemennya, jadi akan menggunakan operasi *n.* Run time dalam Big O notation nya adalah O(*n*).
* Big O notation tidak mengekspresikan kecepatan setiap algoritma. Big O notation mengajarkan kita untuk membandingkan jumlah operasi. Dimana Big O akan mengekspresikan bagaimana suatu algoritma dapat bekerja dengan cepat.
* Binary search membutuhkan log *n* untuk mengecek operasi list dari *n.*
* Jadi apa running time dalam Big O notation? Adalah O(log n).
* Secara umum, Big O notation dalam di tulis sebagai.

****

* Ini mengatakan pada kita operasi angka dari algoritma akan terbuat.
* Di juluki Big O notation karena kita meletakkan “O besar” di depan sebelum kita menghitung jumlah operasinya ( seperti lelucon, tetapi itulah faktanya!)

1. **Visualisasi perbedaan Big O run times (distudi kasus yang berbeda)**

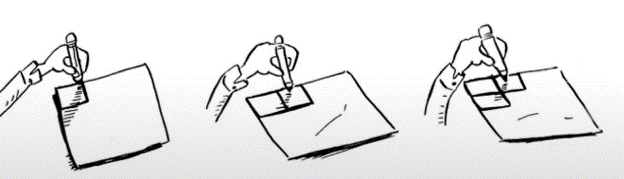
* Kita diminta untuk menggambarkan 16 box

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

* Apa algoritma yang cocok untuk menggambar grid semacam ini?

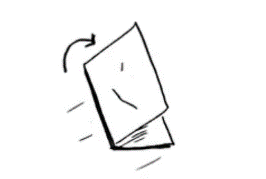
Algoritma 1

* Salah satu untuk melakukannya adalah menggambar 16 box, diwaktu yang sama.
* Ingat, Big O notation akan menghitung jumlah operasi.
* Seperti contoh, menggambar 1 box berarti membutuhkan 1 operasi. Kita harus menggambarkan 16 box yang berarti membutuhkan 16 operasi.
* Berapa banyak operasi yang dibutuhkan, untuk menggambar satu box dalam 1 waktu?



* Ini akan membutuhkan 16 step untuk menggambar 16 box. Apa running time untuk algoritma ini?

Algoritma 2

* Mari coba algoritma ini. Melipat kertas
* Dalam kasus ini, melipat kertas sekali adalah sebuah operasi
* Kita baru saja membuat 2 box dengan operasi ini!
* Lipat kertas lagi, lagi, dan lagi!



* Setelah lipatan ke-empat, buka kertas kembali dan kita akan mendapatkan 16 box yang indah!
* Dapat kita lihat, setiap lipatan akan membentuk kelipatan 2 dari box. Kita telah membuat 16 box dengan 4 operasi
* Kita juga dapat menggambar 2 box dari setiap lipatan, jadi kita dapat menggambar 16 box dalam 4 step.
* Apa running time untuk algoritma ini?
* Jawaban: Algoritma 1 membutuhkan O(n) time, dan algoritma 2 membutuhkan O(log*n*) time.