ALGORITMA BRUTE FORCE

1. (*Sub-array terbesar*) Diberikan array bilangan bulat $a_1, a_2, \ldots, a_n$. Anda diminta untuk mencari sub-array yang memiliki jumah maksimal. Nilai maksimum sub-urutan adalah nol jika semua elemen array negatif. Sebagai contoh: array $[-2, 11, -4, 13, -5, 2, -1, 3]$ memiliki nilai maksimum sub-array 20, yaitu dari elemen ke-2 hingga elemen ke-4 (yaitu $[11, -4, 13]$, jumlah totalnya adalah $11 + (-4) + (13) = 20$).

   Solusi:

2. (*Masalah inversi*) Netflix menerapkan sistem rekomendasi untuk merekomendasikan film yang Anda sukai. Netflix mencoba mencocokkan film favorit Anda dengan film lain. Kemudian Netflix memeriksa basis data mereka untuk mencari orang dengan preferensi serupa. Tingkat kesamaan yang digunakan adalah jumlah *inversi* antara dua peringkat. Misalnya, peringkat dari Anda adalah $1, 2, 3, \ldots, n$, sedangkan peringkat dari orang terebut adalah $a_1, a_2, \ldots, a_n$. Film i dan j disebut inversi jika $i < j$ namun $a_i > a_j$. Misalnya, untuk film A, B, C, D dan E:

Movie	A	B	C	D	E
My rank	1	2	3	4	5
Rank of person X	1	3	4	2	5
Inversion 1		**3**		**2**	
Inversion 2			**4**	**2**	

Inversion: $(3, 2)$ and $(4, 2)$

Movie	A	B	C	D	E
My rank	1	2	3	4	5
Rank of person Y	1	2	4	3	5
Inversion 1			**4**	**3**	

Inversion: $(4, 3)$

   Karena jumlah inversi dengan Y lebih sedikit dibandingkan dengan X, maka preferensi saya lebih mirip dengan Y.

   Anda diminta untuk memecahkan masalah inversi ini dengan cara berikut: diberikan array A dengan n elemen. Hitung jumlah inversi dalam array. Pembalikan didefinisikan sebagai berikut: jika $i < j$ tetapi $A[i] > A[j]$, maka pasangan $(A[i], A[j])$ disebut inversi. Contoh: $A = [1, 9, 6, 4, 5]$ memiliki 5 inversi, yaitu pasangan: $(9, 6)$, $(9, 4)$, $(9, 5)$, $(6, 4)$, dan $(6, 5)$.

   (a) Hitung jumlah inversi dan daftar pasangan dalam array: $A = [1, 5, 4, 8, 10, 2, 6, 9, 3, 7]$.

   (b) Jika masalah inversi diselesaikan menggunakan algoritma Brute-Force, jelaskan langkah-langkahnya! Berapa banyak perbandingan elemen yang dibutuhkan dan bagaimana kompleksitas waktu dari algoritma (jika dinyatakan dalam notasi $\mathcal{O}(\cdot)$)?

   Solusi:

3. (*Banyak pemberhentian minimum*) Misalkan kita menempuh jarak sejauh 100 km, mulai dari titik 0 dan berakhir di kilometer 100. Terdapat pom bensin di sepanjang jalan dengan jarak 10, 25, 30, 40, 50, 75 dan 80 km dari titik awal. Tangki bensin awalnya hanya cukup untuk menempuh jarak 30 km. Bagaimana kita sampai ke tempat pemberhentian sehingga kita berhenti sesedikit mungkin?

Solusi:

4. (*Gadget testing*) Suatu perusahaan sedang melakukan uji coba untuk menentukan lantai tertinggi dari kantor pusatnya yang berlantai n, yang memungkinkan sebuah gadget dapat jatuh tanpa mengalami kerusakan. Perusahaan tersebut memiliki dua gadget identik untuk bereksperimen. Jika salah satunya rusak, maka gadget tersebut tidak dapat diperbaiki, dan eksperimen harus diselesaikan dengan gadget yang tersisa. (Catatan: kompleksitas waktu dihitung berdasarkan banyaknya percobaan yang dilakukan.)

   (a) Rancang sebuah algoritma dengan pendekatan brute-force untuk masalah ini. Jelaskan algoritmanya, kemudian tulis pseudocode untuk algoritma tersebut, dan tentukan kompleksitas waktunya!

   (b) Modifikasi algoritma tersebut untuk meningkatkan efisiensinya. (*Hint:* kita bisa membuat algoritma dengan kompleksitas waktu *worst-case* $\mathcal{O}(\sqrt{n})$.)

Solusi:

(a) Kita bisa mulai dari lantai pertama dan memeriksa setiap lantai yang berada tepat di atasnya secara terurut. Maka kita tidak perlu dua gadget. Kasus terbaik adalah ketika gadget rusak di lantai pertama. Kasus terburuk adalah ketika gadget pertama kali rusak di lantai ke-n. Tidak ada kasus terbaik metode lain yang lebih baik dari yang ini.

Tapi kita biasanya lebih peduli tentang kasus terburuk (kasus rata-rata lebih penting tetapi sulit untuk dianalisis). Jadi, kita akan mencoba meningkatkan efisiensi karena kita memiliki lebih dari satu gadget.

(b) Untuk meminimalkan jumlah gadget jatuh yang dibutuhkan, kita harus membuat "lompatan besar". Tetapi ketika gadget pertama rusak, kita kembali ke pencarian linier untuk sejumlah "lompatan" yang dilakukak pada kasus terburuk. Jadi, kita harus mencapai trade-off yang optimal antara jumlah lompatan dan panjang lompatan.

Katakanlah, panjang lompatan adalah j. Kemudian dalam kasus terburuk kita perlu membuat $\frac{n}{j}$ lompatan, dan setelah itu kita harus melakukan pencarian linier dengan panjang j. Titik optimal adalah ketika $\frac{n}{j} = j$ atau $j = \sqrt{n}$. Jadi, kita jatuhkan gadget pertama dari lantai $\sqrt{n}$ dan jika tidak rusak, pindah ke lantai $2 \cdot \lfloor\sqrt{n}\rfloor$, dan seterusnya. Saat gadget pertama rusak, kita harus menjatuhkan $\lfloor\sqrt{n}\rfloor$ dengan gadget kedua dalam kasus terburuk. Jadi, algoritmanya ada di $\mathcal{O}(\sqrt{n})$.

5. (*Titik-titik terdekat*) Misalkan $x_1 < x_2 < \cdots < x_n$ adalah bilangan real yang mewakili koordinat n desa yang terletak di sepanjang jalan lurus. Kantor pos perlu dibangun di salah satu desa ini.

   1. Rancang algoritma yang efisien untuk menemukan lokasi kantor pos yang meminimalkan jarak rata-rata antara desa dan kantor pos.
   2. Rancang algoritma yang efisien untuk menemukan lokasi kantor pos yang meminimalkan jarak maksimum dari desa ke kantor pos.

Solusi:

6. (*Permasalahan partisi*) **Permasalahan partisi** didefinisikan sebagai berikut.

   > *Diberikan n bilangan bulat positif, partisi menjadi dua himpunan bagian yang terpisah dengan jumlah elemen yang sama. (Perhatikan bahwa masalah ini tidak selalu memiliki solusi.) Rancang algoritma exhaustive search untuk masalah ini, dengan meminimalkan jumlah himpunan bagian yang perlu dihasilkan oleh algoritma.*

   Jelaskan algoritmanya, tuliskan pseudocode-nya, dan hitung kompleksitas waktu terburuknya!

7. (*Magic square*) Sebuah *magic square* dengan ordo n adalah susunan bilangan bulat dari 1 hingga n^2 dalam matriks $n \times n$, dengan setiap angka muncul tepat satu kali, sehingga setiap baris, setiap kolom, dan setiap utama diagonal memiliki jumlah yang sama.

    1. Buktikan bahwa jika *magic square* berorde n ada, maka jumlah yelemen-elemen pada setiap baris, kolom, maupun diagonalnya adalah $n(n^2 + 1)/2$.
    2. Rancanglah algoritma *exhaustive search* untuk menghasilkan semua *magic square* dengan orde n.
    3. Bacalah literatur dan selidiki algoritma yang lebih baik untuk menghasilkan kotak ajaib.
    4. Terapkan dua algoritma untuk *magic square*, yaitu: *exhaustive search* dan yang Anda temukan dalam bahasa pemrograman.
    5. Jalankan eksperimen untuk menentukan nilai terbesar dari n dimana setiap algoritma dapat menemukan *magic square* dengan orde n dalam waktu kurang dari 1 menit dengan program yang Anda buat.

Solusi: