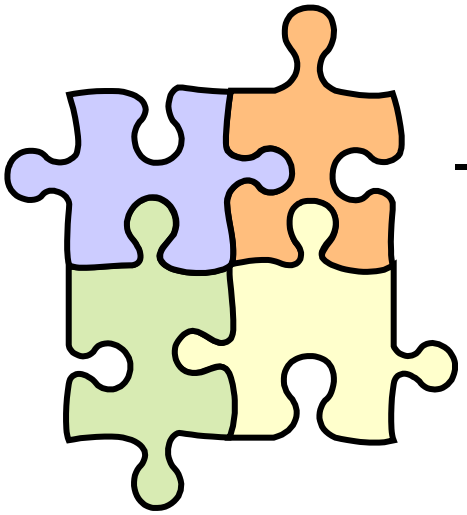
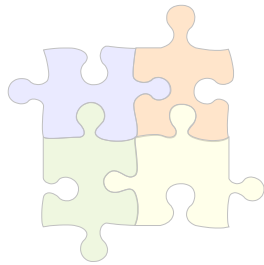
# Langkah Awal menuju Analisis Kompleksitas Algoritma



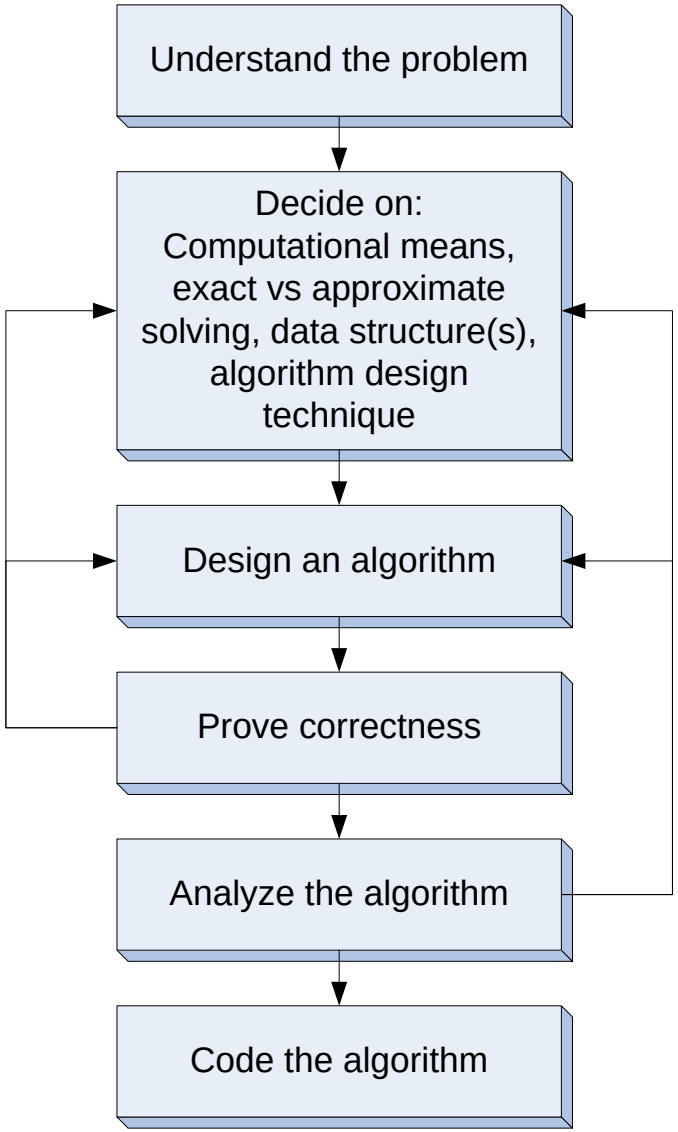
Analisis dan Strategi Algoritma

# Isi



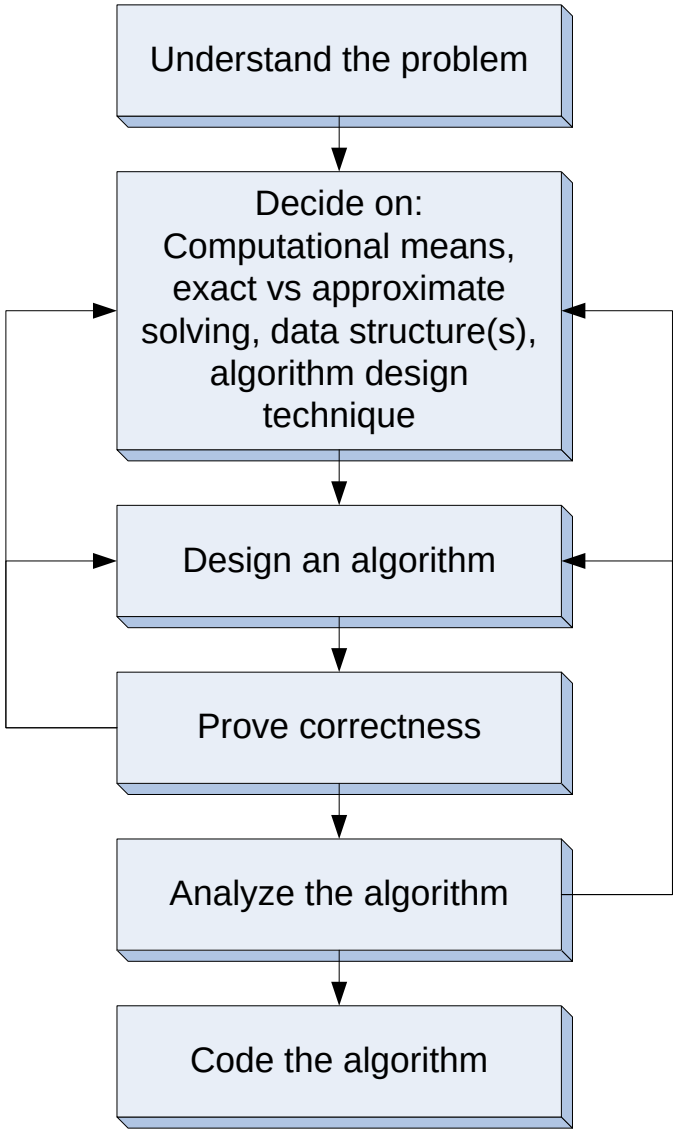
- Proses Desain dan Analisis Algoritma
- Tipe-tipe Problem yang penting
- Kebutuhan akan algoritma yang efisien
- Analisis framework

# Proses Desain dan Analisis Algoritma



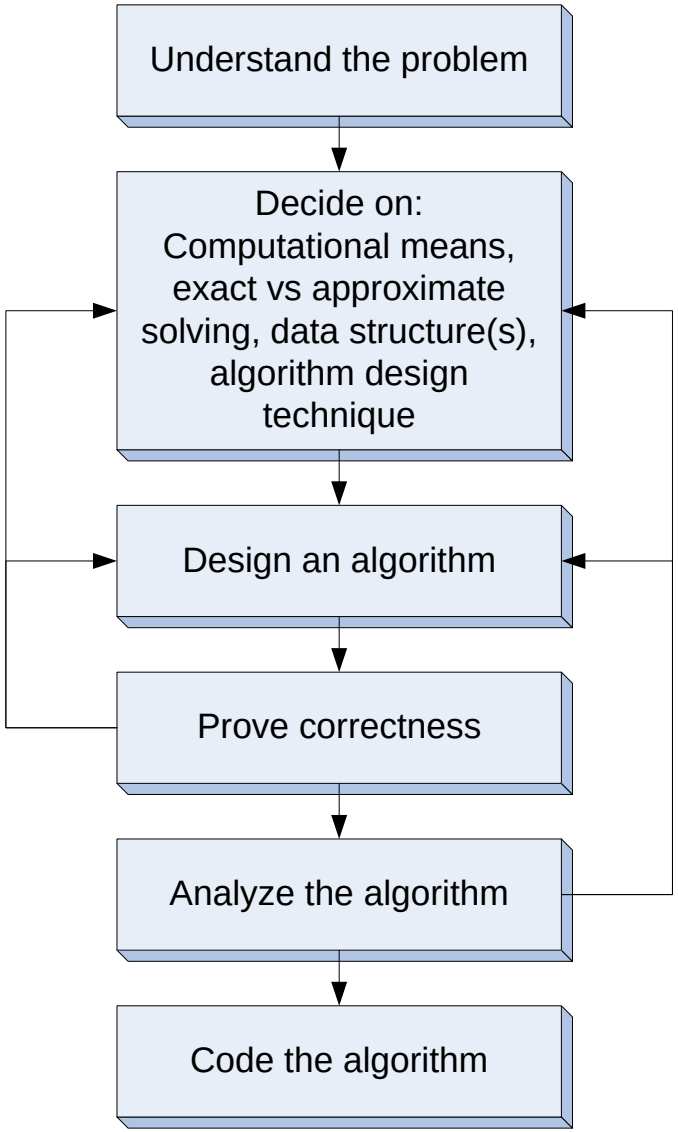

- Input → contoh problem; menentukan jangkauan problem
- Kemampuan peralatan komputasi
- Perkiraan:
  - Problem tidak dapat diselesaikan dengan tepat , contoh: akar pangkat dua
  - Allgoritma yang memiliki solusi secara tepat akan ditolak jika memerlukan waktu yang relatif lama.
  - Algoritma perkiraan adalah bagian dari algoritma eksak yang lebih memuaskan
- Algoritma+Struktur Data = Program
- Pendekatan umum untuk memecahkan masalah secara algoritmik

# Algorithm Design and Analysis Process



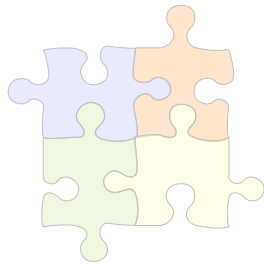

- Menentukan algoritma:
  - Menggunakan bahasa alami
  - Menggunakan flowchart
  - Menggunakan desain hardware
  - Menggunakan source code program
  - Menggunakan pseudocode
  - Bentuk lain yang lebih cocok?
- Correctness: membuktikan bahwa algoritma menghasilkan hasil yang diinginkan untuk setiap input yang sesuai dalam waktu tertentu
  - Biasanya menggunakan induksi matematis
  - Dapatkan kita gunakan tracing sederhana?
    - Incorrectness
  - Approx alg → the error < limit

# Algorithm Design and Analysis Process



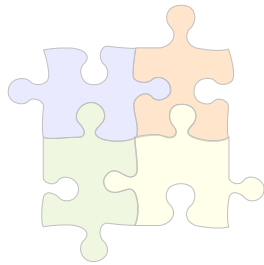

- Kualitas algoritma:
  - Correctness
  - Efficiency:
    - Time efficiency
    - Space efficiency
  - Simplicity
  - Generality: the problem, input range
- Pemrograman algoritma:
  - Resiko: transisi yang tidak benar atau tidak efisien
  - Pembuktian kebenaran program?
  - Practical: testing & debugging

# Beberapa tipe problem penting



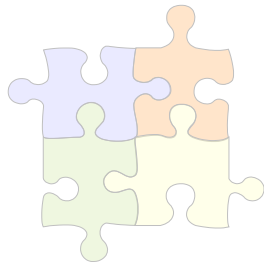
- Sorting
- Searching
- Pemrosesan String
- Problem Graph
- Problem Kombinatorik
- Problem Geometrik
- Problem Numerik

# Tipe Problem: Sorting



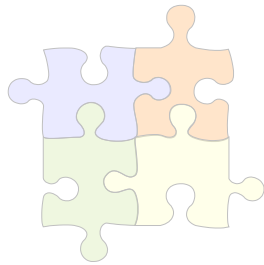
- Problem: menyusun ulang hal-hal yang terdapat pada daftar dengan urutan naik
- Jika ada records, kita perlu sebuah key
- Terdapat beberapa lusin algoritma sorting
- Dua properti algoritma sorting:
  - Stabil: Mempertahankan urutan relatif sembarang dua elemen input yang sama
  - Di tempat: tidak memerlukan memori ekstra, kecuali, mungkin, beberapa unit memori

# Tipe Problem: Searching



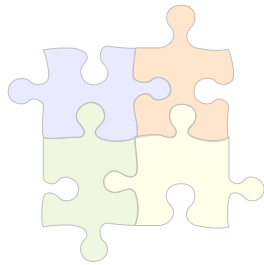
- The problem: menemukan suatu nilai dari sekumpulan nilai yang ada
- Jangkauan algoritma searching:
  - Pencarian sekuensial hingga pencarian binary (sangat efisien, namun terbatas) dan algoritma didasarkan pada representasi kumpulan nilai tersebut sehingga memungkinkan pencarian yang lebih baik
- Tantangan:
  - Kumpulan data yang sangat besar
  - Update: add, edit, delete

# Tipe Problem: Pemrosesan String



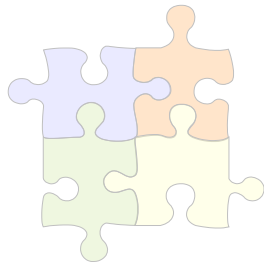
- String = urutan karakter alphabet
- Minat khusus: text strings, binary strings, gene sequences etc.
- Problem khusus: pencocokan string
  - Pencarian suatu kata dalam text

# Tipe Problem: Problem Graph



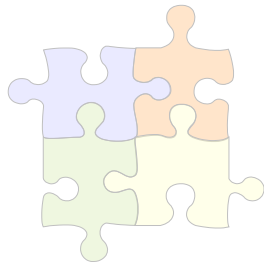
- Algoritma graph dasar: graph traversal, shortest-path, sorting topologik pada graph dengan ujung berarah
- Beberapa problem sangat sulit diselesaikan dengan cara komputasi – hanya beberapa contoh problem yang dapat diselesaikan dalam waktu yang dapat diterima–
  - Traveling Salesman Problem (TSP)
  - Graph-Coloring Problem (GCP)

# Tipe Problem : Permasalahan Kombinatorik



- Problem: menemukan suatu objek kombinatorik –seperti permutasi, kombinasi, atau subset – yang memenuhi batasan-batasan tertentu dan memiliki properti yang diinginkan
- Problem-problem paling sulit
  - Sejumlah objek kombinatorik tertentu tumbuh dengan cepat seiring peningkatan ukuran problem
  - Tidak diketahui algoritma eksak untuk menyelesaikan problem tersebut dalam waktu yang diinginkan
- Dari perspektif yang lebih abstrak, TSP & GCP merupakan salah satu contoh permasalahan kombinatorik

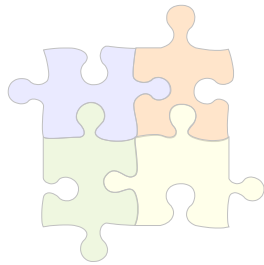
# Tipe Problem : Pemasalahan Geometrik



- Berkaitan dengan objek geometrik: titik, garis, poligon etc.
- Yunani kuno: membangun bentuk geometris sederhana –segitiga, lingkaran etc.– menggunakan penggaris dan kompas yang tidak ditandai
- Masa kini: aplikasi komputer grafik, robot, tomography*
- Problem klasik:
  - Problem closest-pair: diberikan n titik pada suatu bidang, temukan pasangan terdekat diantaranya
  - Problem Convex hull: temukan poligon cembung terkecil yang melibatkan semua titik yang telah ditentukan

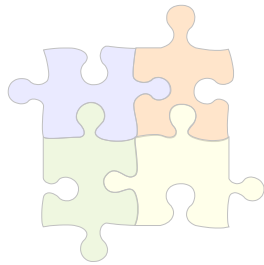
***) method of radiography displaying details in a selected plane within the body**

# Tipe Problem: Permasalahan Numerik



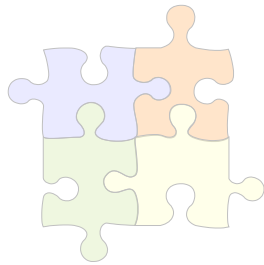
- Melibatkan objek matematis yang memiliki sifat kontinyu: memecahkan persamaan dan sistem persamaan, menghitung integral tak berhingga, mengevaluasi fungsi etc.
- Mayoritas problem-problem diatas hanya dapat dipecahkan dengan perkiraan
  - Komputer hanya dapat merepresentasikan angka real dengan kira-kira
  - Akumulasi kesalahan round-off
- Perubahan fokus komputasi industri: analisis numerik (pada industri & ilmu pengetahuan) menuju aplikasi bisnis(penyimpanan informasi, retrieval, transportasi melalui jaringan, dan presentasi kepada pengguna)

# Kebutuhan algoritma yang Efisien



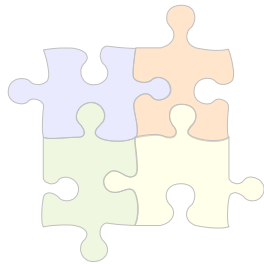
- Anggaplah anda memiliki komputer dengan kecepatan yang tidak terbatas dengan kapasitas memori gratis yang tidak terbatas
- Masih perlukah untuk mempelajari algoritma?

# Absolutely YES!



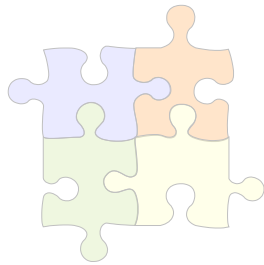
- Anda masih harus mendemonstrasikan bahwa metode pencarian solusi anda dapat berakhir dengan jawaban yang benar pula.

# Back to the real world



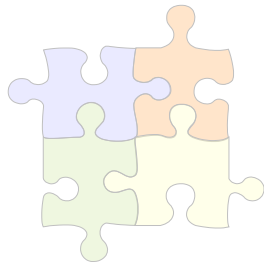
- Komputer mungkin saja cepat, tapi kecepatannya tidak mungkin tidak terbatas
- Memory mungkin saja murah , tapi tidak akan gratis.
- Resource yang membatasi:
  - Waktu komputasi
  - Ruang memori
- Resource tersebut harus digunakan dengan bijaksana, dan algoritma yang efisien akan membantu anda

# Efficiency: Sebuah Ilustrasi



- Ambil contoh dua algoritma sorting:
  - Insertion sort: membutuhkan waktu kira-kira sama dengan n hingga $c_1 n^2$ untuk sorting n item → $n^2$
  - Merge sort: membutuhkan waktu ($c_2$ n $\log_2$ n) untuk sorting n item → n $\log_2$ n
- $c_1 < c_2$ → tidak begitu berpengaruh dibanding ukuran input n
- Insertion sort biasanya lebih cepat dibanding merge sort jika jumlah input sedikit.
- Jika jumlah input n besar, keuntungan merge sort, $\log_2$ n vs n, akan lebih bisa mengimbangi perbedaan faktor konstan

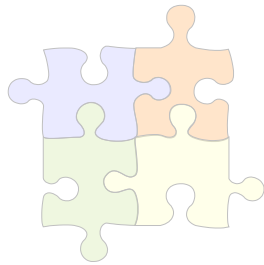
# Efficiency: Contoh (1)



- Array yang akan di-sort: $10^6$ angka
- Komputer A: $10^9$ inst/sec; running insertion sort; programer terbaik; codeIS → $2n^2$
- Computer B: $10^7$ inst/sec; running merge sort; programer biasa, HLL; the codeMS → $50\, n \log_2 n$

- Comp A: $(2(10^6)^2) / 10^9 = 2000$ sec
- Comp B: $(50.10^6 \log_2 10^6) / 10^7 \approx 100$ sec

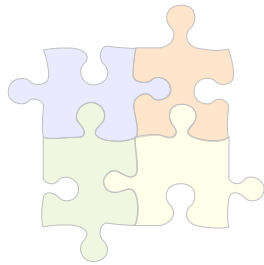
# Efficiency: Contoh Konkrit(2)



- Dengan menggunakan algoritma dengan pertambahan waktu eksekusi yang lebih lambat,penggunakan kompiler yang lebih buruk, comp B menyelesaikan 20x lebih cepat dibanding A.

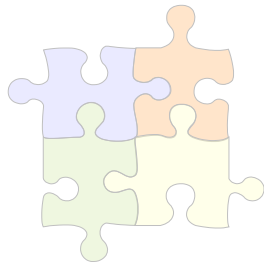
- Sorting $10^7$ angka…
  - Comp A: timeIS ≈ 2.3 days
  - Comp B: timeMS < 20 minutes

# Analisis Framework



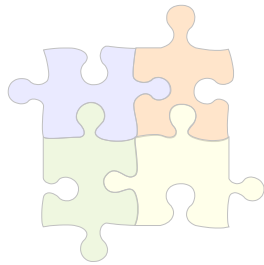
- Mengukur ukuran/jumlah input
- Unit untuk mengukur waktu eksekusi
- Tingkat pertumbuhan
- Efisiensi Worst-case, Best-case, and Average-case

# Mengukur jumlah/ukuran input



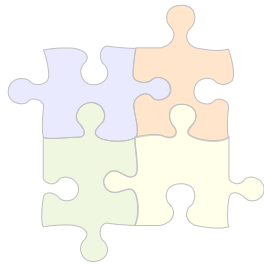
- Hampir seluruh algoritma membutuhkan waktu lebih lama untuk menyelesaikan input yang lebih banyak
- Penyelidikan efisiensi algoritma, logis dilakukan sebagai fungsi dari beberapa parameter seperti ukuran/jumlah input (n)
- Pada kebanyakan kasus, seleksi n input langsung dilakukan; contoh: ukuran daftar yang akan di-*sorting* ataupun *di-searching*.
- Pada problem evaluasi polinomial pangkat n, ukuran input adalah pangkat polinomial tersebut ataupun koefisiennya.

# Pemilihan Parameter yang mengindikasikan ukuran sangatlah penting



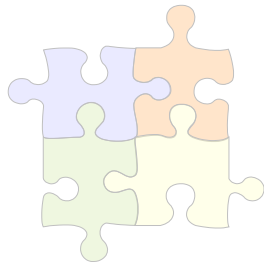
- Contoh: komputasi produk yang mempunyai matrik n x n
- Dua tolak ukur yang lazim:
  - Matrik ukuran n
  - Jumlah elemen N pada matrik yang dikalikan
    → dapat diterapkan pada matrik n x m

# Pilihan dapat dipengaruhi Operasi Algoritma



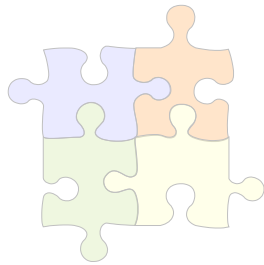
- Bagaimana kita mengukur ukuran input pada algoritma pemeriksaan ejaan?
  - Jika memeriksa karakter sebagai input → jumlah karakter
  - Jika memeriksa kata→ jumlah kata
- Pada algoritma yang melibatkan properti angka
  - Size = jumlah bit b pada representasi binary ke-n
  - $b = \lfloor \log_2 n \rfloor + 1$

# Unit Pengukuran Running Time



- Dapatkan kita menggunakan unit standar pengukuran waktu –detik, milidetik, dan sebagainya– ?
  - Kekurangan: tergantung pada kecepatan komputer, tergantung pada kualitas program, kesulitan pewaktuan (clocking) running time aktual dari program
- Salah satu pendekatan:mengukur jumlah waktu yang diperlukan setiap operasi algoritma dijalankan → sulit dan tidak perlu

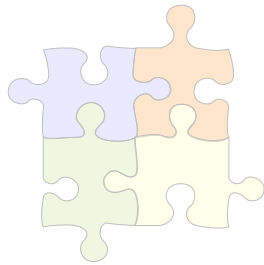
# Operasi Dasar



- Mengidentifikasi operasi paling penting dari suatu algoritma (*operasi dasar*)
  - Operasi tersebut memakai waktu yang paling banyak dari keseluruhan running time
- Menghitung jumlah waktu yang diperlukan untuk mengeksekusi operasi dasar

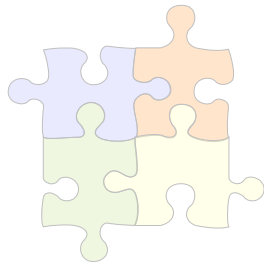
- Framework yang telah ada untuk analisis efisiensi waktu suatu algoritma: menghitung jumlah waktu eksekusi operasi dasar dalam algoritma dengan input ukuran n

# Tingkat Pertumbuhan



- Untuk n dengan jumlah besar, yang diperhitungkan adalah fungsi tingkat pertumbuhan
  - Abaikan pengali yang konstan
  - Exp: ½ $n^2$ → $n^2$, 50 n $\log_2$ n → n $\log_2$ n
- Fungsi-fungsi penting:
  - $\log_2$ n, n, n $\log_2$ n, $n^2$, $n^3$, $2^n$, n!

# Hanya jumlah input tidaklah cukup

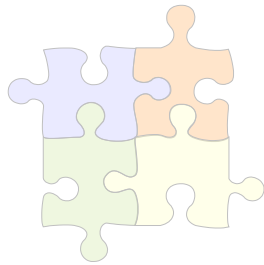


- Algoritma pencarian sekuensial

```
//Input: array A[0..n-1], search key K
//Output: index of first element of A that
//        matches K or -1 if there is no match
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i else return -1
```

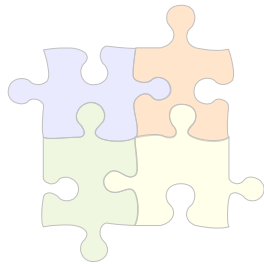
- Jalannya algoritma ini bisa sangat berbeda untuk ukuran data yang sama (n)
- The best, the worst, and average-case?

# Efisiensi Worst-case



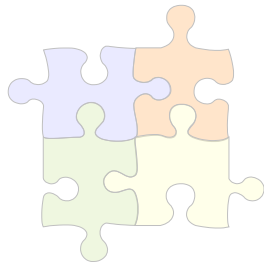
- $C_{worst}(n)$: Algoritma efisiensi  untuk input worst-case dengan ukuran n
  - Algoritma memerlukan waktu yang paling lama diantara semua input yang ada
- $C_{worst}(n)$ adalah penting: membatasi running time-nya
  - Akan menjamin running time tidak akan melebihi $C_{worst}(n)$ dari seluruh input yang mungkin

# The Best-case Efficiency



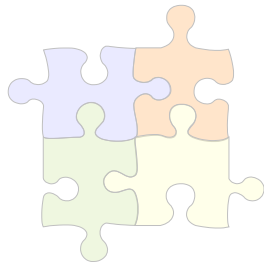
- $C_{best}(n)$: Algoritma efisiensi untuk input best-case dengan ukuran n
  - Algoritma yang menyelesaikan problem paling cepat diantara semua input (dengan ukuran n) yang mungkin
- Kegunaan:pada beberapa algoritma, kinerja best-case yang baik diberikan oleh beberapa tipe input yang mendekati tipe input yang terbaik
  - Exp: Input best-case untuk alg insertion sort adalah array yang telah disorting; kondisi best-case ini sedikit memburuk untuk array yang hampir selesai disorting → Insertion Sort mungkin metode pilihan untuk aplikasi yang berkaitan dengan array yang hampir selesai disorting

# Efisiensi Average-case



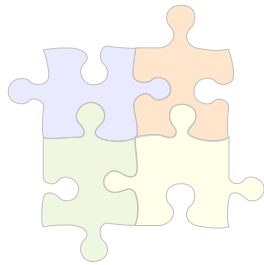
- Analisis worst-case maupun best-case tidak menghasilkan informasi yang diperlukan mengenai jalannya algoritma pada input ‘typical’ ataupun ‘random’.
- $C_{avg}(n)$
  - Kita harus membuat beberapa asumsi tentang input berukuran n yang mungkin

# $C_{avg}(n)$ untuk Pencarian Sekuensial (1)



- Asumsi standar:
  - Kemungkinan pencarian yang berhasil = p ($0 \leq p \leq 1$)
  - Kemungkin kecocokan pertama terjadi pada posisi ke-i adalah sama untuk semua i
- Kemungkinan kecocokan pertama pada posisi ke-i adalah p/n untuk setiap i; jumlah pembadingan yang dilakukan = i
- Pada kasus pencarian yang tidak berhasil, jumlah pembandingan adalah n, dengan kemungkinan (1-p)

# $C_{avg}(n)$ for Sequential Search (2)

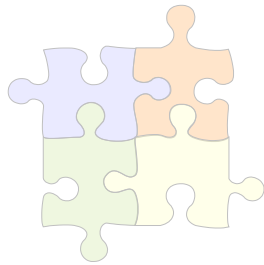


$$C_{avg}(n) = \left[1.\frac{p}{n} + 2.\frac{p}{n} + \ldots + i.\frac{p}{n} + \ldots + n.\frac{p}{n}\right] + n.(1-p)$$

$$= \frac{p}{n}\left[1 + 2 + \ldots + i + \ldots + n\right] + n.(1-p)$$

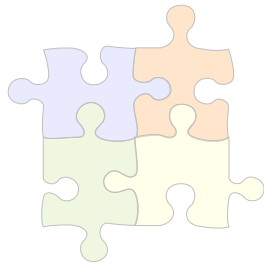
$$= \frac{p}{n}\frac{n(n+1)}{2} + n.(1-p) = \frac{p(n+1)}{2} + n.(1-p)$$

# Catatan pada $C_{avg}(n)$



- Apa kita benar-benar memerlukan $C_{avg}(n)$?
  - Jauh lebih baik dibanding $C_{worst}(n)$ yang terlalu pesimis
- $C_{avg}(n)$ tidak dapat diambil dengan cara mengambil rata-rata $C_{worst}(n)$ dan $C_{best}(n)$

# Tugas (1)



1. Buatlah sebuah program menggunakan bahasa pemrograman Pascal untuk mengimplementasikan algoritma binary search