Algorithm analysis ≈ complexity analyiss

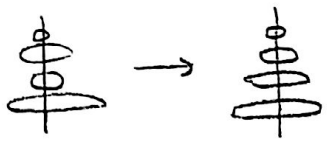
1) Time complexity

2) Space complexity

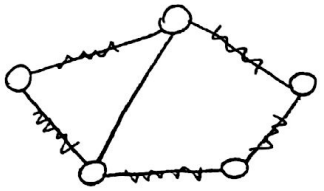
Lower Bound Analysis

Problemin karmaşıklığı ile algoritmanın karmaşıklığı aynı şey değildir. Lower bound problemlerle alakalı. Bir problemin çözümü mesela $O(n^2)$den küçük olması imkansızsa o problemin lower boundu $O(n^2)$ dir.

→ Towers of Hanoi



→ Hamilton Circuit Problem



öyle bir path olsun ki tüm vertexleri içersin.

NP-complete → yes veya no ile cevap verilebiler problemler.

→ Traveling Salesman Problem

En az maliyetle dolaşmak.

NP-hard problem → max, min filan soranlar.

Basic Operation

Usually the most important operation. Daha uzun zaman alan ve/veya daha fazla kullanılan istem.

An operation that the algorithm performs

OR

the operation contributing the most to the total execution time.

- searching a list
- sorting a list
- finding max in a list

} Basic operation is comparision

- traversing a tree/graph ... /visiting a node

T_n = the set of all inputs of size n

$T(I)$ = the # of basic operations performed by the algorithm for input I.

Best case complexity of an algorithm :

$$B(n) = \min \{ T(I) : I \in T_n \}$$

Worst case complexity : $W(n) = \max \{ T(I) : I \in T_n \}$

Average " " : $A(n) = \sum_{I \in I_n} T(I) \cdot P(I)$

$P(I)$ = probability of occurrence of input I

For a data size n, we can analyze 3 types of complexity

1) Best case complexity
2) Worst " "
3) Average " "

For instance, if A(n) is good, but W(n) is very bad, then I might not want to use this algorithm.

Calculating A(n):

→ Determine suitable values for the probabilities.
(i.e., a prob. distr. for the input)

→ We need to know & apply elements of probability theory.

→ Formula for A(n) (usually) involves summation or recurrence relation.

A(n)

⇓

We should convert the formula into "closed form"

* General Framework for the Analysis of an Algorithm

→ Decide parameter(s) indicating input size.

→ Identify the basic operation of the algorithm.

→ Count how many times the basic operation is performed for an input of size n.

* For some algorithms, complexity does not depend on the nature of the input. If this is the case, then $A(n) = W(n) = B(n)$

Usually, iterative algorithms result in summation and recursive algorithms result in recurrences.

Suppose that we have arrived at a closed form formula

$$A_1(n) = 3n^2 + \frac{100 \log n}{\sqrt{n}}$$

$$A_2(n) = \frac{10 + 2(n! - 5n)}{n^{3/2}}$$

Growth rate of functions

Language ⇒ $\Theta, O, \Omega, o, \omega$

Asymptotic Analysis ⇒ Analyzing the behavior of an algorithm as the input size tends to infinity.

$$f_1(n) = \frac{n^2}{2} - \frac{n}{2}$$

$$f_2(n) = n^2$$

n	n/2	$n^2/2$	$n^2/2 - n/2$	n^2
10	5	50	45	100
1000	500	. .	. .	
10000	5000	.	–	– .
100000		.	— .	– .

bunlar artık aynı gibi

$$\lim_{n \to \infty} \frac{f_1(n)}{f_2(n)}$$

3. edition

→ Anony Levitin
Introduction to Algorithm Design & Analysis of Algorith.

→ Kleinberg & Tardos
Algorithm Design

⇒ f_1 ve f_2 aynı hızda büyür.

Eliminate low-order terms (n^2 & n)

Eliminate constant factors

iff (if and only if)

∴ (therefore)

The complexity $f(n)$ of this algorithm is of order $g(n)$:>>

sth like a class of all functions related to $g(n)$.

Let F be the set of all functions $f(n)$ $f(n): \mathbb{N} \to \mathbb{R}$ that are eventually positive

$\exists a$ ⇓ there exist $\#$ ⇓ number n_0 s.t. ⇓ such that $f(n) > 0$ $\forall n > n_0$ ⇓ for all

Let $f(n)$ and $g(n) \in F$

$f(n) \in O(g(n))$ iff there exist positive constants c_1, c_2 and n_0 such that $\boxed{c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \text{whenever} \quad n \geq n_0}$

$\Theta(g(n))$ represents lots of functions. Always a set of functions.

$f(n) \not\in \Theta(g(n))$ $f(n) \in \Theta(g(n))$

$\Theta(n^2) = \{ n^2, 5n^2, n^2-3, n^2+n, n^2+\sqrt{n}, \ldots \}$

✗

Ex: $n^2 \in \Theta(5n^2+n)$

$c_1 \cdot 5n^2+n \leq n^2 \leq c_2 \cdot (5n^2+n)$ for $n \geq n_0$ $c_1 = ?$ 1/10
$c_2 = ?$ 1
$n_0 = ?$ 1

Ex: $n^2 \notin \Theta(n^3)$

→ Çelişki ile ispat

Solution. $\exists\ c_1, c_2$ & n_0 . s.t.

$c_1 \cdot n^3 \leq n^2 \leq c_2 \cdot n^3$ for all $n \geq n_0$

$c_1 \cdot n \leq 1$ $c_1 \leq \frac{1}{n}$

$1 \leq c_2 \cdot n$

As $n \to \infty$, $c_1 = 0$ → sadece 0 bunu sağlar.

$c_2 \leq \frac{1}{n}$

Yani sonuçta $n^2 \notin \Theta(n^3)$ olur.

02.10.14

Equivalence Relition (Denklik Bağıntısı)

→ Reflexivity $f(n) \in \Theta(f(n))$

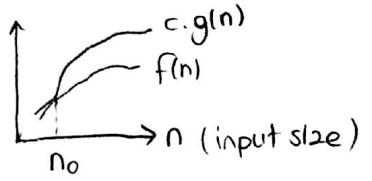
→ Symmetricity $f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n))$

→ Transitivity $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$

Big-Oh (O) Upper Bound

A function $f(n) \in O(g(n))$ iff there exists positive constants $\leq$ and n_0 such that $f(n) \leq c \cdot g(n)$ whenever $n \geq n_0$

s.t.



* $f(n)$ grows at the same rate or slower than $g(n)$
* O(Big-Oh) notation gives on upper bound for complexity

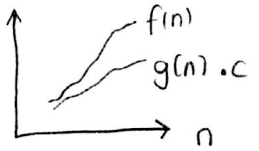
Ex: $5n^2 \in O(n^2)$ → genelde bu yazılır.

$5n^2 \in O(n^3)$

Omega (Ω) Lower Bound

A function $f(n) \in \Omega(g(n))$ iff there exists positive constants $\leq$ and n_0 s.t.

$c \cdot g(n) \leq f(n)$ whenever $n \geq n_0$



* $f(n)$ grows at the same rate or faster than $g(n)$
* Ω notation gives a lower bound for complexity.

$\Rightarrow \Omega$ genelde algoritmanın değil problemin karmaşıklığını anlatmak için kullanılır.

$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

↓ intersection (kesişim)

Little-Oh

A function $f(n) \in o(g(n))$ iff for every positive constant $\leq$, there is a positive integer n_0 such that $f(n) < c \cdot g(n)$ whenever $n \geq n_0$

* $f(n)$ grows slower than $g(n)$ → Big Oh 'daki eşitlik yok.

Bişey little-oh ise aynı zamanda big-oh'dur.

Little-Omega

A function $f(n) \in \omega(g(n))$ iff for every positive constant $\leq$, there is a positive n_0 such that $c\,g(n) < f(n)$ whenever $n \geq n_0$

* $f(n)$ grows faster than $g(n)$

Asymtotic Notation - Relational Operations

$\Theta \rightarrow =$

$O \rightarrow \leqslant$

$\Omega \rightarrow \geqslant$

$o \rightarrow <$

$\omega \rightarrow >$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 \text{ (constant)} ; \text{ if } f(n) \in o\,g(n) \\ c>0 ; \text{ if } f(n) \in \Theta(g(n)) \\ \infty ; \text{ if } f(n) \in \omega(g(n)) \end{cases}$$

L'hospital's Rule

Let $f(x)$ and $g(x)$ be functions that are table differentiable for sufficiently large real numbers x. If $\lim_{n\to\infty} f(x) = \infty$ and $\lim_{n\to\infty} g(x) = \infty$

OR if $\lim_{n\to\infty} f(x) = 0$ and $\lim_{n\to\infty} g(x) = 0$; then $\lim_{n\to\infty} \frac{f(x)}{g(x)} = \lim_{n\to\infty} \frac{f'(x)}{g'(x)}$

Properties

Let $f(n), g(n) \in F$

* $f(n) \in O(g(n)) \Leftrightarrow g(n)\ \Omega\ f(n)$
* $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$ (Denklik bağıntısındaki simetriklik)
* $f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
* $f(n) \in o(g(n)) \Leftrightarrow g(n) \in \omega(f(n))$
* $f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$
* $f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$

> terslerı her zaman doğru değil.

* Given $c>0$, $\Theta(f(n)) \equiv \Theta(c.f(n))$ → c'nin bir önemi yok.
* $f(n) \in O(g(n)) \Leftrightarrow O(f(n)) \underset{\text{subset}}{\subseteq} O(g(n))$

$n \in O(n^2)$ $\quad O(n) \subseteq O(n^2)$

$\Downarrow$

$\{n, n+\log n, 2n\}$

* $f(n) \in o(g(n)) \Leftrightarrow O(f(n)) \subset O(g(n))$
* $O(f(n)) = O(g(n)) \Leftrightarrow \Theta(f(n)) = \Theta(g(n)) \Leftrightarrow \Omega(f(n)) = \Omega(g(n))$

02.10.14

* Given two functions $f(n), g(n) \in F$,

$f(n)$ and $g(n)$ have the same order if $f(n) \in \Theta(g(n))$

* $f(n)$ has smaller order than $g(n)$ if $O(f(n)) \subset O(g(n))$

** $f(n)$ and $g(n)$ are <u>not comparable</u> if $f(n) \notin O(g(n))$ and $g(n) \notin O(f(n))$

Ex: Functions involving modulo operations

Papularly used classes:

→ $O(1)$ constant order. Usually in best case complexity.

→ $O(\log n)$ logarithmic order.

→ $O(n \cdot \log n)$

→ $O(\log \log n)$ ⇒ çok iyi bir durum.

Base of the logarithm is irrelevant in algorithm analysis.

→ $O(n^a)$, where $0 < a < 1$

A famous member of this classes is $O(\sqrt{n})$ $(a = \frac{1}{2})$

Ex: n^a ? $\log n$

$$\lim_{n\to\infty} \frac{n^{0.01}}{\log n} = \lim_{n\to\infty} \frac{0.01\, n^{-0.99}}{\frac{1}{n}} = \lim_{n\to\infty} 0.01 \cdot n^{0.01} = \infty$$

consider $a = 0.01$

n^a grows faster than $\log n$.

⇒ $\log n$ daha iyi

→ $O(n)$ Linear order
(" complexity)

→ $O(n \cdot \log n)$: Typical divide & conquer algorithms. (Quick sort, Merge sort)

→ $O(n^2)$: Quadratic order (for for) (Burası $O(1)$ ise!)

→ $O(n^3)$: Cubic order

* → $O(a^n)$ for $a > 1$: Exponential order ⇒ pratikte büyük değilse kabul edilebilir.

Towers of Hanoi

a^{an} $a^{\log \log n}$: exponential order.

→ $O(n \cdot a^n)$ for $a > 1$

→ $O(n!)$ Factorial order . Recursive bazı şeyler.

$$O(1) \subseteq O(\log\log n) \subset O(\log n) \subset O(n^a) \underset{a<1}{} \subset O(n) \subset O(n \cdot \log n) \subset O(n^a) \underset{a>1}{} \subset O(a^n) \subset$$

polynmial ⟸ ⟹ exponential

devamı

$$O(a^n) \subset O(n \cdot a^n) \subset O(n!) \subset O(a^{n^b})$$

An algorithm whose complexity is at most $O(f(n))$, where $f(n)$ is a polynomial of n is called a polynomial-time algorithm.

An algorithm whose complexity is at least $\Omega(a^n)$; $a > 1$; is called an exponential time algorithm.

Polynomial-time algorithms are tractable, whereas exp.-time algo's are not.

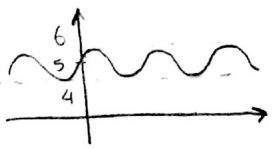
Poly-logarithmic order = $O((\log n)^a)$

$\log \log n$

$\log \log \log n$

$o(n)$ → sublinear

Ex: $f(n) = 5 + \sin(n)$ What is the complexity of $f(n) = ?$



→ It is always bounded by 6 ↳ constant $f(n) \in O(1)$

It is always bounded below by 4 ↳ constant $f(n) \in \Omega(1)$

$f(n) \in \Theta(1)$

Ex: Prove that $\sum_{k=1}^{p} k^p$ is $\Theta(n^{p+1})$ for $p > 1$ (a fixed constant).

for $k = 1 = p$

$[\ k^p\ ]$

bunun exponential olup olmaması p'nin constant olup olmamasına bağlı.

Ex: Prove that $\sum_{k=1}^{n} k^p$ is $\Theta(n^{p+1})$ for $p>1$ a fixed constant

Proof: $\Theta \Rightarrow O$ ve Ω old. göstermeliyiz.

Find an upper bound (O):

$$\sum_{k=1}^{p} k^p = \underbrace{1^p + 2^p + 3^p + \ldots + n^p}_{n \text{ terms}}$$

$1^p \leq n^p$

$2^p \leq n^p$

tüm terimler $\leq n^p$

$$\leq n \cdot n^p$$

$$\leq n^{p+1}$$

$$\sum_{k=1}^{p} k^p \in O(n^{p+1})$$

Find an lower bound (Ω):

$$\sum_{k=1}^{n} k^p = \sum_{k=1}^{n/2} k^p + \sum_{k=\lfloor \frac{n}{2} \rfloor + 1}^{n} k^p \geq \sum_{k=\lfloor \frac{n}{2} \rfloor + 1}^{n} k^p$$

bunu aynen yazdık.

$$= \underbrace{\left(\lfloor \tfrac{n}{2} \rfloor + 1\right)^p + \left(\lfloor \tfrac{n}{2} \rfloor + 2\right)^p + \ldots + n^p}_{\lceil \frac{n}{2} \rceil \text{ terms} \geq \frac{n}{2} \text{ terms}}$$

Each term $\geq \left(\frac{n}{2}\right)^p$

$$\geq \frac{n}{2} \cdot \left(\frac{n}{2}\right)^p = \frac{n^{p+1}}{2^{p+1}} \in \Omega(n^{p+1})$$

floor: $\lfloor x \rfloor$ (taban) an integer $\leq x$

ceiling: $\lceil x \rceil$ (tavan) an integer $\geq x$

Ex: $f(n) = \frac{n(n+1)}{2}$

$g(n) = n^2$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{\frac{n^2+n}{2}}{n^2} = \frac{n^2+n}{2n^2} = \frac{1}{2}$$

$f(n) \in \Theta(g(n))$ $\quad g(n) \in \Theta(f(n))$

Soru. Show that if $p(n)$ is a polynomial of degree k, where the coefficient of n^k is greater than 0, then $p(n) \in \Theta(n^k)$

$O(n^{1000\,000}) < 2^n$

Ex: Show that any polynomial function has smaller order than any exponential function.

i.e $O(n^k) < O(a^n)$ for $k>0$ $a>1$

$$\lim_{n\to\infty} \frac{n^k}{a^n} = \frac{k\cdot(n^{k-1})}{\ln a \cdot a^n} = \frac{k\cdot(k-1)\cdot n^{k-2}}{(\ln a)^2 \cdot a^n} = \lim_{n\to\infty} \frac{k!}{(\ln a)^k \cdot a^n} = \lim_{n\to\infty} \frac{1}{a^n} = 0 \;//$$

($k!$ sabit)

Ex: Computer orders of growth of $n!$ and 2^n

Solution: Stirling's Formula → Faktoriyel görünce ise yarayabilir.

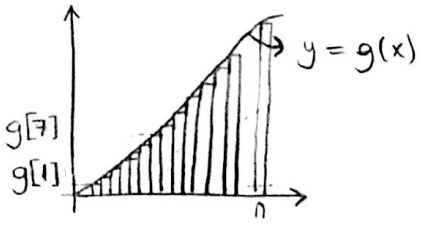
$$n! \cong \sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n \quad \text{for large } n$$

$$\lim_{n\to\infty} \frac{\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n}{2^n} = \lim_{n\to\infty} \left(\sqrt{2\pi n}\cdot\left(\frac{n}{2e}\right)^n\right) = \infty$$

($\frac{n}{2e}$ sabit)

Sometimes, this Σ is too complex. At that time, we squeeze the $\sum$ btwn 2 integrals.

Let $f(n) = \sum_{i=1}^{n} g(i)$, where g is a nondecreasing function.



integral değeri ≥ sumation değeri

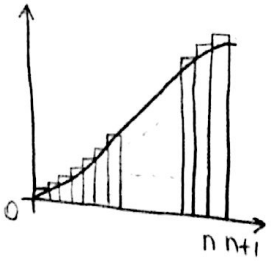
$g(1)\cdot 1 + g(2)\cdot 1 + g(3)\cdot 1 + \dots + g(n)\cdot 1$ → dikdörtgenlerin alanı

$$= \sum_{i=1}^{n} g(i) = f(n)$$

$\int_1^{n+1} g(x)\cdot dx$ = Entire area under the curve

$$\Longrightarrow f(n) \le \int_1^{n+1} g(x)\cdot d(x)$$

Big-Oh için de kullanılabilir.



→ Böyle büyük dikdörtgenleri alırsam, Omega (lower bound) bulmuş olurum.

$$g(1)\cdot 1 + g(2)\cdot 1 + \dots + g(n)\cdot 1$$

$$\int_0^1 g(x)\cdot dx \le f(n) \qquad f(n) \ge \int_0^n g(x)\cdot dx$$

this is valid iff g is a non-decreasing function.

Ex: $f(n) = \sum_{i=1}^{n} i^2$

↓

non-decreasing ✓

$$\int_0^n g(x)\,dx \leq f(n) \leq \int_1^{n+1} g(x)\,dx$$

$$\int_0^n x^2\,dx \leq f(n) \leq \int_1^{n+1} x^2\,dx$$

$$\frac{x^3}{3}\Big|_0^n \leq f(n) \leq \frac{x^3}{3}\Big|_1^{n+1}$$

$$n^3 \leq f(n) \leq \frac{(n+1)^3}{3} - \frac{1}{3} \Rightarrow f(n) \in \Theta(n^3)$$

★ logaritma non-decreasing

Ex: Sum of logarithms

$$L(n) = \sum_{i=1}^{n} \log i = \log(n!)$$

Solution 1:

$$\int_0^n \ln x\,dx \leq L(n) \leq \int_1^{n+1} \ln x\,dx$$

$$x\ln x - x\Big|_0^n \leq L(n) \leq x\ln x - x\Big|_1^{n+1}$$

$$n\ln n - n \leq L(n) \leq (n+1)\ln(n+1) - (n+1) + 1$$

$$\underbrace{(n+1)\ln(n+1) - n}$$

$O(n \ln(n))$

$\boxed{L(n) \in O(n \log n)}$

Solution 2:

$$L(n) = \underbrace{\log(1) + \log(2) + \dots + \log(n)}_{n \text{ terms}}$$

Each term $\leq \log n$

$\boxed{L(n) \in O(n \log n)}$

For lower bound:

$$\sum_{i=1}^{\lfloor n/2 \rfloor} \log(i) + \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} \log(i) = \log\left(\lfloor \tfrac{n}{2} \rfloor + 1\right) + \log\left(\lfloor \tfrac{n}{2} \rfloor + 2\right) \underbrace{\dots + \log(n)}$$

$n - \lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil$ # of terms $\geq \frac{n}{2}$ terms

Each term $\geq \log\left(\lfloor \frac{n}{2} \rfloor + 1\right) \geq \log\left(\frac{n}{2}\right)$

$\geq \frac{n}{2} \log\left(\frac{n}{2}\right)$ $\boxed{L(n) \in \Omega(n \log n)}$

Ex: Harmonic Series

$$H(n) = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

NOT non-decreasing, in fact it is non-increasing

$$\int_{1}^{n+1} g(x)\,dx \leq H(n) \leq \int_{0}^{n} g(x)\,dx \qquad \text{for a non-increasing function}$$

$$\int_{1}^{n+1} \frac{1}{x}\,dx \leq H(n) \leq \int_{0}^{n} \frac{1}{x}\,dx$$

$$\ln x \Big|_{1}^{n+1} \leq H(n) \leq \ln x \Big|_{0}^{n}$$

$$\ln(n+1) \leq H(n) \leq \ln(n) - \underbrace{\ln 0}_{-\infty}$$

$\infty + \infty$

$$H(n) = \Omega(\ln(n))$$ bu kısım işimize yaramadı.

* So, this method works for finding a < B, but not for an UB. (upper bound).

Linear (Sequential) Search

Pseudo code:

A list with n elements

```
function LinearSearch ( L[1:n], x )
                                  ↳ searched element
    for i = 1 to n do
        if ( L[i] == x ) then
            return i
        end if
    end for
```

→ Basic operation = comparison

* A comparison-based searching or sorting algorithm is an algorithm that is based on making comparisons involving list elements and then making decisions based on these comparison

Best case: If $x = L[1]$, then best case occurs $B(n) = 1 \in O(1)$

Worst case: If $x = L[n]$ or x does not occur in the list, the worst case occurs,

$W(n) = n \in O(n)$

Average case: Assume that the prob. of a succesful search is P

Assume that x can equally likely be found in any position. (uniform prob. distr.)

A classical assumption

...sume that the elements in the list are distinct. Let p_i = prob. that i operations will be done.

09.10.14 ⑤

Prob. that the searched element is at index i.

$$A(n) = \sum_{i=1}^{W(n)} i \cdot p_i \qquad p_i = \dots, \forall p$$ for a succesful search

$$= \begin{cases} \frac{p}{n} & \text{for } 1 \le i \le n-1 \\ & \end{cases}$$

Prob. that $(x = L[i]) = \frac{p}{n}$ for $1 \le i \le n-1$

Prob. that $(x = L[i])$ or x does not occur in the list $= \frac{p}{n} + \underbrace{1-p}_{\text{olmama olasılığı}}$ for $i = n$

$$A(n) = \sum_{i=1}^{n} i \cdot p_i = \left(\sum_{i=1}^{n-1} i \cdot \frac{p}{n} \right) + n \cdot \underbrace{\left[\frac{p}{n} + 1 - p \right]}_{\text{prob. that searched item is at the end or isn't in the list.}}$$

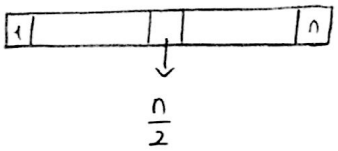
$$= \frac{p}{n} \cdot \frac{n(n+1)}{2} + p + n(1-p)$$

$$= \frac{p(n-1)}{2} \quad \dots \quad \in \Theta(n)$$

Average complexity is also linear.

Binary Search

Assume that the list is sorted. Firstly suppose that $n = 2^k - 1$



Worst case occurs if ; x is not in the list <u>OR</u> x is at $L[1]$ <u>OR</u> x is at $L[n]$

Compare x with $L[2^{k-1}], L[2^{k-2}], L[2^{k-3}], \dots, L[1]$

⇓

There are k comparison. $k = \log_2(n+1)$ $W(n) = \log_2(n+1)$ for $n = 2^k - 1$

genellersek ;$W(n)$ for all n ;

$W(n)$ is a non-decreasing function

$2^{i-1} \le n \le 2^i - 1$ $n = 2^k - 1$ old. göre

$3 \le 6 \le 7$ $(i = 3)$

I can find i such that this holds.

$W(2^{i-1}-1) \le W(n) \le W(2^i - 1)$

$W(n) = \log_2(n+1)$ for $n = 2^k - 1$

⇓

$\log_2(2^{i-1}-1+1) = \log_2(2^{i-1}) \le W(n) \le \log_2(2^i)$

$i - 1 \le W(n) \le i$ ⇒ now let's write i in terms of n

Insertion Sort

23.10.14
Persembe
(1)

1 comparison will occur if $x = L[1] > L[i-1]$

2 " " " if $L[i-2] < x < L[i-1]$

.
.
.

i " " " if $L[1] < x < L[2]$

i " " " $x < L[1]$

There are $i+1$ cases

" " $i+1$ intervals that x can fall into we assume that each of these intervals is equally likely.

$$P(T_i = J) = \begin{cases} \frac{1}{i+1} & ; \text{ if } 1 \le J \le i-1 \\ \frac{2}{i+1} & ; \text{ if } J = i \end{cases}$$

Prob. that there are J comparisons

$$\frac{i(i+3)}{2(i+1)} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

$$E[T] = \sum_{i=1}^{n-1} E[T_i] = \sum_{i=1}^{n-1} \left(\frac{i}{2} + 1 - \frac{1}{i+1} \right)$$

$$= \frac{n(n-1)}{4} + n - 1 - \underbrace{\sum_{i=1}^{n-1} \frac{1}{i+1}}_{\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}}$$

$$\underbrace{\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}}_{H_{n-1}}$$

$$\frac{n(n-1)}{4} + n - 1 - (H_n - 1) = \frac{n(n-1)}{4} + n - H_n \qquad H_n \in \Theta(\log n)$$

$$\in \Theta(n^2)$$ ⇓ Average case complexity

* Liste cok da karmasik degilse insertion, cok karisiksa quick sort kullanmak mantikli.

* A sorting algorithm is a stable sorting algorithm if given any 2 elements $L[i], L[j]$ where $i<j$ and $L[i]=L[j]$, the final positions i', j' of $L[i]$, $L[j]$, respectively satisfy $i'<j'$

* An sorting algorithm is an online sorting algorithm if the entire list is not input (Teker teker input geliyor biz yerleştiriyoruz.) to the algorithm in advance, but instead elements are added to the list over time. The algorithm maintains the dynamic list in sorted order.

  ↳ Insertion sort is online

  ↳ Quicksort is not online.

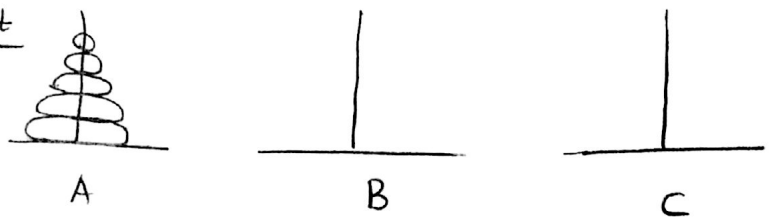
* Given integers n, k, where $k<=n$ suppose that $L[1..n]$ is a list such that every element in the list is no more than k positions from its stable final position in the sorted list L.

Then insertion sort performs at most $2k(n-1)$ comparisons when sorting the list.

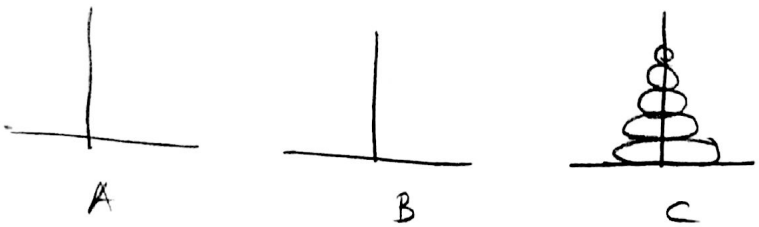
$k=1 \Rightarrow$ almost sorted if k is fixed, then insertion sort $\in \Theta(n)$

Towers of Hanoi

Input



Goal




```
procedure Tower of Hanoi (A,B,C, n)
    if n=1 then
        write ("Move from A to C")
    else
T(n-1) ← call Towers (A,B,C,n-1)
 T(1)  ← write ("Move from A to C)
T(n-1) ← call Towers (B,A,C,n-1)
    end if
end
```

$T(n) = 2T(n-1)+1$

$T(1) = 1$

$T(2) = 2(T(1))+1$

$T(3) = 2.(T(2)+1$

$= 2.(2.T(1)+1)+1$

$T(4) = 2^3.T(1) + 4+2+1$

$T(n) = 2^{n-1}.\underbrace{T(1)}_{1} + 2^{n-2} + 2^{n-3} + 2^{n-2} + .. + 1$

$\sum_{k=0}^{n-1} 2^k \Leftarrow$ 2^i

$\sum_{k=0}^{n-1} x^k = \frac{x^n - 1}{x-1}$; for $x \neq 1$ $n>0$ $x>1$

$\sum_{k=0}^{n-1} x^k = \frac{x^n-1}{x-1} \Rightarrow \frac{2^n-1}{2-1} = \underline{2^n-1}$ $\Rightarrow$ Towers of Hanoinin costu.

$T(n) \in \Theta(2^n)$

General Form Of A Recurrence Relation

$x(n) = c_1 . x(n-1) + c_2 x(n-2) + \dots + c_k . x(n-k) + f(n)$

$\underbrace{x(0), \dots, x(k-1)}$ are given

Initial conditions

OR

$x(n) = c . x\left(\frac{n}{b}\right) + f(n)$; $x(1)$ is given

* These relations can be non-linear, homogenous / In homogenous recurrence relations with constant coefficients.

Forward Substitution

$T(n) = 2\,T(n-1) + 1$

$T(1) = 1$
$T(2) = 3$
$T(3) = 7$
$T(4) = 15$

Try to observe a common pattern.

Guess:

$T(n) = 2^n - 1$ dedik. Simdi ispatlamamiz lazım.

Tümevarım olabilir.

Proof: $T(1) = 2^1 - 1$

$T(1) = 1$ Base Case

Inductive Step:

Assume true for $k-1$

$T(k-1) = 2^{k-1} - 1$

Prove for k.

$T(k) = 2T(k-1) + 1$

$= 2(2^{k-1} - 1) + 1$

$= 2^k - 2 + 1 = 2^k - 1$ //

Backward Substitution

$x(n+1) = b_{n+1} \cdot x(n) + c_{n+1}$

$\vdots$

$x(n+1) = b_{n+1} \cdots$

closed form formula

$x(n+1) = a \cdot x(n) + b \cdot x(n-1)$; $n \geq 1$ $x(0)$ & $x(1)$ are given.

$\Downarrow$

2nd order recurrence relation

Characteristic equation

$\alpha^2 = a\alpha + b$

$\alpha^2 - a\alpha - b = 0$

Roots: α^+ ve α^-

$T(n) = T(n-1) + T(n-2)$

$T(1) = 1$

$T(2) = 1$

$T(n) = T(n-1) + T(n-2) + T(n-3)$

Roots α^+ and α^-

(masela Fibonacci)

i) If both α^+ and α^- are real and distinct, then $x(n) = c_1 \alpha_+^n + c_2 \alpha_-^n$ for some c_1, c_2

we can find these constants from the initial conditions.

ii) If $\alpha^+ = \alpha^-$ (say α) and real, then

$x(n) = c_1 \alpha^n + c_2 \cdot n \cdot \alpha^n$

Recurrence Relation For Fibonacci Numbers

$f(n) = f(n-1) + f(n-2)$ fibonaccinin karakteristik denklemi $\Rightarrow \alpha^2 = \alpha + 1$

$f(0) = 0$ ⇓

$f(1) = 1$ roots : $\Delta = b^2 - 4ac$

$$x_{1,2} = \frac{-b \mp \sqrt{\Delta}}{2a}$$

$$f(1) = c_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right) + c_2 \left(\frac{1-\sqrt{5}}{2}\right)$$

$$\Rightarrow c_1 - c_2 = \frac{2}{\sqrt{5}}$$

$$c_1 + c_2 = 0$$

$$2c_1 = \frac{2}{\sqrt{5}}$$

$$c_1 = \frac{1}{\sqrt{5}} \quad c_2 = -\frac{1}{\sqrt{5}}$$

Example : $x(n+1) = 2 \cdot x(n) - x(n-1)$ $n \geq 1$ $x(0) = 1$ $x(1) = 5$

Solution :

$$\alpha^2 = 2\alpha - 1$$

$$\alpha^2 - 2\alpha + 1 = 0$$

$$(\alpha - 1)^2 = 0 \qquad \alpha_+ = 1, \quad \alpha_- = -1$$

$$x(n) = c_1 \alpha^n + c_2 \cdot n \cdot \alpha^n$$

$$x(0) = c_1 = 1$$

$$x(1) = c_1 \cdot \alpha + c_2 = 5 \quad c_2 = 4$$

$$x(n) = 1 + 4n \qquad \boxed{x(n) = 4n+1}$$

Another type of recurrence :

$$x(n) = a\, x(n/b) + f(n) \; ; \; x(1) = c$$

$a > 1 \Rightarrow$ Divide & conquer algo. (Quick sort + merge sort)

1) $x(n) = a^{n-1} + \sum_{i=2}^{n} a^{n-1} \cdot f(i)$ [By backward substitution]

* If $a=1$ and $f(n)=1 \Rightarrow x(n) = b + \sum_{i=2}^{n} 1 = b+n-1 \in \Theta(n)$

$x(n) = x\left(\frac{n}{b}\right) + 1$

* If $a=1$ and $f(n)=\log n \Rightarrow x(n) = b + \sum_{i=2}^{n} \log(i) = b + \log(n!)$ ($\log$ ka inside ka ain $\Rightarrow n$)

$\in \Theta(n \log n)$

Master Theorem:

Let $x(n)$ be an eventually non-decreasing function that satisfies the recurrence relation:

$x(n) = a \cdot x\left(\frac{n}{b}\right) + f(n)$; $n = b^k$ $(k = 1, 2, \ldots)$ $x(1) = c$ where $a \geq 1$, $b \geq 2$, $c > 0$

If $f(n) \in \Theta(n^d)$, where $d \geq 0$, then

$$x(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \cdot \log n) & \text{if } a = b^d \text{ for all } n \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

16.10.14

BINARY SEARCH

$\log_2(n+1) - 1 \leq W(n) \leq \log_2(n+1) + 1 \quad \therefore W(n) \in \Theta(\log n)$ for all n

This method is called interpolation

Average Case :

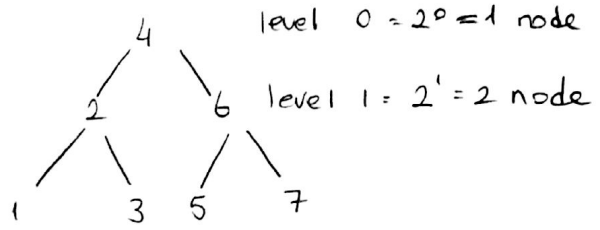
Assume that → The prob. of a succesful search. is $p \quad 0 \leq p \leq 1$

→ It is equally likely that x can be found in any position.

→ If x is not in the list / its equally likely that it can be found in any one of the $n+1$ intervals

$L[1], L[2], \ldots, L[n]$ ↳ $x > L[n]$

$x < L[1]$ $L[1] < x < L[2]$ $L[n-1] < x < L[n]$ $\quad \frac{1-p}{n+1}$



level $0 = 2^0 = 1$ node

level $1 = 2^1 = 2$ node

Internal node = leaf haricindekiler

$\underbrace{1 + 2^1 + 2^2 + \ldots + 2^{k-1}}_{\text{internal node}} + \underbrace{2^k}_{\text{son level leaves}}$

★ $2^k - 1$ tane internal node var.

Prob. that x can be found in any position $L[i] = \frac{p}{n}$ for $1 \leq i \leq n$

Average number of comparisons ⇒ If x equals $L[4]$ ⇒ 1 comparison

$x = L[2]$ ya da $x = L[6]$ → 2 comparison

$x = L[1]$ $x = L[3]$ $x = L[5]$ $x = L[7]$

$\underbrace{\frac{p}{n} \cdot 1}_{x = L[4]} + \underbrace{\frac{p}{n} \cdot 2}_{x = L[2]} + \underbrace{\frac{p}{n} \cdot 2}_{x = L[6]} + \underbrace{\frac{p}{n} \cdot 3 \ldots\ldots}_{\text{4 tane}}$ ⇒ $\frac{p}{n}(1+2+2+3+3+3+3+\ldots)$

1. levelde 2. levelde 3. levelde

Internal Path Lenght = Sum of the lenghts of the paths from the root to the internal notes.

Average # of comparisons = $\frac{p}{n}(IPL(T) + n)$

IPL ⇒ Internal Path Lenght.

1+1+2+2+2+2 . . . IPL

Comparison sayısı = IPL + n ⇒ listede varsa.

Listede yokken

Leaf Path Length = The sum of the lengnths of the paths from the root to the leaf nodes.

When x is not in the list, x falls into any one of the n+1 intervals = $\frac{1-p}{n+1}$

Average # of comparisons = $\frac{1-p}{n+1}$ ($\underbrace{3+3+3+\ldots+3}_{LPL}$)

$$A(n) = \sum_{I \in T(n)} T(I) \cdot P(I) = \frac{p}{n}(IPL + n) + \frac{1-p}{n+1}(LPL)$$

* A 2-tree is a binary tree such that every node that is not a leaf has exacty 2 children.
* For a 2-tree T, IPL(T) = LPL(T) − 2I I is the # (number) of internal nodes.
* For any binary tree T, $LPL(T) \geq L \cdot \lfloor \log_2 L \rfloor + 2(L - 2^{\lfloor \log_2 L \rfloor})$ where L is the number of leaf nodes.

* $A(n) = \frac{p}{n} \cdot \underbrace{(IPL+n)}_{IPL+n = LPL-n} + \frac{1-p}{n+1} \cdot (LPL) \Rightarrow$ n = # of internal nodes

$$A(n) = \frac{p}{n} \cdot (IPL(T)+n) + \left(\frac{1-p}{n+1}\right) LPL(T)$$

$$= \frac{p}{n} \cdot (LPL(T) - n) + \left(\frac{1-p}{n+1}\right) \cdot LPL(T)$$

$$= \left(\frac{p}{n} + \frac{1-p}{n+1}\right) \cdot LPL(T) - p$$

$$\geq \left(\frac{p}{n} + \frac{1-p}{n+1}\right) \left[\underbrace{(n+1)}_{L} \cdot \lfloor \log_2 n+1 \rfloor + 2\left((n+1) - 2^{\lfloor \log_2 (n+1) \rfloor}\right)\right] - p$$

$$\geq \underbrace{\left(\frac{p}{n} + \frac{1-p}{n+1}\right)}_{\Omega(\frac{1}{n})} \underbrace{(n+1) \cdot \lfloor \log_2 (n+1) \rfloor}_{\Omega(\log n)} - p \qquad A(n) \in \Omega(\log n)$$

ERTION SORT

```
procedure InsertionSort ( L[1:n] )
  for i = 2 to n do
      current = L[i]
      position = i-1
      while ( (position ≥ 1) and (current < L[position]) ) do
          L[position+1] = L[position]
          position = position - 1
      end while
      L[position+1] = current
```

Best Case: Basic op. is executed once and the while loop terminates (each time)

$$B(n) = \sum_{i=2}^{n} 1 = n-1 \quad \in \Theta(n)$$

Worst case: For each iteration of the loop, the while loop is executed max # of times.

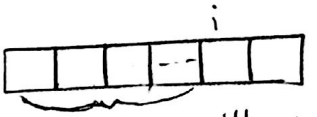
Küçükten büyüge sıralanmış listeyi büyükten küçüge sıralamak istersek.

$$W(n) = \sum_{i=2}^{n-1} (i-1) = \sum_{k=1}^{n-2} k = \frac{(n-1)(n-2)}{2} \in \Theta(n^2)$$

Average Case:

Step 1: Step 1 of the for loop

Place L[2] in its proper position



already sorted — i^{th} elemanı sorted kısma yerleştirmeye çalışıyoruz.

Let T_i = # of basic operations of step i

$$T = T_1 + T_2 + \dots + T_{n-1} = \sum_{i=1}^{n-1} T_i$$

↓ is a random variable.

$$E[T] = E\left[\sum_i T_i\right] = \sum_i E[T_i]$$

↓ expected value of T.

Calculate $E[T_i]$

$i \rightarrow$ step olan i

$$E[T_i] = \sum_{J=1} J \cdot P(T_i = J)$$

↳ Prob. that there are J comparisons in step i

30.10.14
Persembe

BRUTE FORCE & EXHAUSTIVE SEARCH

ES is an important special case of BF.

Exponentiation

$a^n = \underbrace{a \times a \times a \times \ldots \times a}_{n \text{ times}}$ ⇒ bu uzun.

$a^n = a \times a^{n-1}$ ⇒ recursive ✓

Selection Sort & Bubble Sort ⇒ 2 well-known example of BF strategy

Selection Sort.

→ Scan the entire list to find the smallest element.

→ Exchange it with the first element

→ Put the smallest element in its final position in the sorted list.

→ Then continue with the 2nd smallest etc.

Pseudocode for Selection Sort (A [0 .... n-1])

```
for i ← 0 to n-2 do
    min ← i
    for j ← i+1 to n-1 do
        if A[j] < A[min]
            min ← j
        end if
        swap A[i] and A[min]
    end for
end for
```

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-i)$$

$$= (n-1) + (n-2) + \ldots + (n-1-(n-2))$$

$$= \sum_{i=1}^{n-2} i = \frac{n.(n-1)}{2} \in \Theta(n^2)$$

Bubble Sort

→ Compare komsu elements of the list

→ Exchange them if they are out-of-order

By doing this repeatedly we end up <<bubbling up>> the largest element to the last position in the list.

If a pass through the list makes no exchanges, the list has been sorted and we swop the algorithm.

Pseudo code for Bubble sort $(A[0 \ldots n-1])$

```
for i ← 0 to n-2 do
    for J ← 0 to n-2-i do
        if A[J+1] > A[J]
            swop A[J] and A[J]
        end if
    end for
end for
```

$$\sum_{i=0}^{n-2} \sum_{J=0}^{n-2-1} 1 = \sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + \ldots + 1$$

$$= \sum_{i=1}^{n-1} i = \frac{n \cdot (n-1)}{2} \in \Theta(n^2)$$

Sequential Search & BF String Matching

NOBODY-NOTICED-HER "NOT" ★ DNA Sequencing

String Matching Problem : Given a string of $\underline{n}$ characters called the text and a string of m characters called the pattern, find a substring of the text that matches the pattern.

Pseudocode BFString Match ((T[0...n-1]) (P[0...m-1]))
Text Pattern

```
for i ← 0 to n-1 do
    J ← 0
    while (J < m and P[J] = T[i+J]) do
        J ← J+1
    end while
    if J = m
        return i
    end if
end for
return -1;
```

$$\sum_{i=0}^{n-m}\sum_{J=0}^{m-1} 1 = \sum_{i=0}^{n-m} m$$

$$= m\cdot(n-m+1)$$

$$= mn - m^2 + m \in O(mn)$$

worst case.

CLOSEST PAIR AND CONVEX HULL PROBLEMS

→ Birbirine en yakın iki nokta nedir? (Closest pair problem)

Distance btwn 2 points (P_i and P_J) $= \sqrt{(x_i-x_J)^2+(y_i-y_J)^2}$

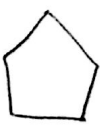
Pseudocode for BF-CP

```
d ← ∞
for i ← 1 to n-1 do
    for J ← i+1 to n do
        d ← min ( d , √((x_i - x_J)^2 + (y_i - y_J)^2) )
    end for
end for
return d ;
```

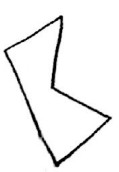
$$\sum_{i=1}^{n-1}\sum_{J=i+1}^{n} 1 = \sum_{i=1}^{n-1}(n-i) = (n-1)+(n-2)+\dots+1$$

$$= \frac{n\cdot(n-1)}{2} \in O(n^2)$$

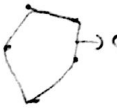
Convex Hull Problem



Convex



Concave



→ convex hull

Convex: A set of points in the plane is called convex if for any 2 points p and q in the set, the entire line segment with the endpoints at p and q

Convex Hull: The convex hull of a set $\leq$ of points is the smallest convex set containing S.

If S is convex, CH is the set S itself.

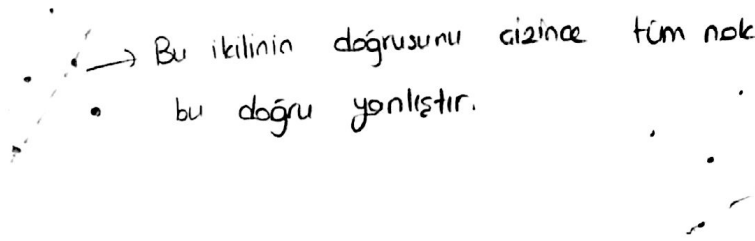
If S is a set of 2 points, CH is the line segment connecting them.

If S is a set of 3 points, CH is the triangle connecting them.

Theorem: The convex hull of any set S of $n \geq 2$ points not all on the some line is a convex polygon with the vertices at some of the points of S. (These points are called extreme points)

Extreme Point: of a convex set is a point of this set that is NOT a middle point of any line segment with end points in this set.

Hence, we need to know which pairs of points need to be connected to form the boundary of the convex hull.



Observation: A line segment connecting 2 points p_i and p_j of a set of n points is a part of the convex hull's boundary iff all the other points of the set lie on the same side of the straight line through these points.

* Repeating this test for every pair of points yields a list of line segments that make up the convex hulls boundary.

For each of $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ pairs of distinct points, we need to check whether the remaining points are on the same side.

Combinatorial problems deal with combinatorial objects such as permutations, combinations, and subsets of a given set.

Combinatorics is a branch of discrete math.

Traveling Salesman Problem (TSP)

n cities K_n → complete graph with n vertices.

Tüm vertexler birbirine bağlı.

Find the shortest (min-cost) (cycle) tour through a given set of cities visits each city exacty once.

An exhaustive search approach to TSP : Find all hamilton cycle and find the one with min-cost.

n notes, how many HCs are there?

$\frac{(n-1)!}{2}$ → cycle olduğu için.

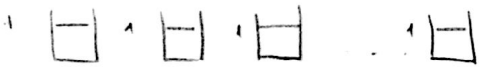
Knapsack Problem

n items, (each with index i) profits P_i weights N, total weight limit W whose total weight

Find the set of items does not exceed the weight limit & total profit is maximized.

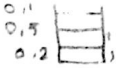
Bin Packing

Bins:



$0 < w_i < 1$

0.2, 0.5, 0.1 lik itemler koyulacak.



Assignment Problem

06.11.14

n people — C[i,J] = cost of assigning person i to job J

n jobs — 1 Job - 1 person — optimize cost s.t

$$\begin{bmatrix} 1 & (3) & 4 & 2 \\ (2) & 5 & 8 & 6 \\ 7 & 4 & (3) & 9 \\ 10 & 2 & 8 & (6) \end{bmatrix}$$

Hungarian Algorithm

$O(n^3)$

Depth - First - Search and Breadth - First - Search

DFS & BFS systematically process all vertices & edges of a given graph.

DFS

→ Start at an arbitary vertex.

↳ Iteration :

→ Proced to an adjacent vertex (Ties are resolved orbitarlly).

→ Continue until a dead end.

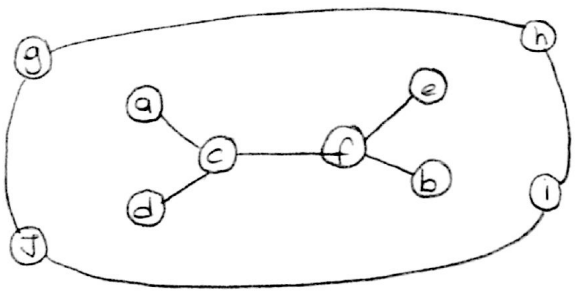
→ Geldiğin yere bak ordan başka yerlere bakmaya calls. (not visited yet)

→ The algorithm halts when you cannot continue any further.

Accompany a DFS traversal by constructing the DFS forest.

Algorithm DFS

```
count ← 0
for each vertex in v do
    if u is marked with 0 (if u is unvisited)
        dfs(u);
        count ← count+1, // Mark u with cound
for each vertex w in u adjacent to u do
    if w is marked with 0 do
        dfs(w)
```



Tree ⇒ acyclic ⇒ forest
connected

$a \to c \to d \to \underset{\text{Back-up}}{c} \to f \to e \to \underset{\text{Back-up}}{f} \to b \to f \to c \to a$

Adjacency Matrix Representation. → $|v| \times |v|$

$$\begin{bmatrix} 0 & 1 & 1 & \cdots \\ 1 & 0 & \cdots & \cdots \\ \cdots & \cdots & 0 & \\ \cdots & \cdots & & 0 \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}_{|v| \times |v|}$$

$\Downarrow$

$O(|v|^2)$

Adjacency List Representation →

$v_1 - v_2 - v_5 - v_6$

$v_2 - v_3 - v_7$

v_3

$\vdots$

$v_{|v|}$

$|v| + \sum \deg(v)$

Handshaking Lemma

$= 2|E|$

$v + 2|E| \Rightarrow O(|v| + |E|)$

Decrease and Conquer

Based on exploiting the relationship between a solution to a given instance of a problem and a solution to its smaller instance. (insertion sort)

1) Decrease by a constant : The size of an instance is reduced by the same constant on each iteration of the algorithm. genellikle bu constant 1 oluyor.

Ex: Exponentiation: Computing $a^n = a^{n-1} \cdot a$

2) Decrease by a constant factor : Suggest reducing a problem instance by the same constant factor an each iteration.
Genellikle bu sayı 2 'dir. (Quick-Sort)

xponentiation

n. sift

$a^n = a^{n/2} + a^{n/2}$

Binary Search

$$a^n = \begin{cases} a^{n/2} + a^{n/2} & \text{if } n \text{ is even.} \\ a^{\frac{n-1}{2}} + a^{\frac{n-1}{2}} * a & \text{if } n \text{ is odd.} \end{cases}$$

3) Variable Size Decrease : The size reduction pattern varies from 1 iteration of the algorithm to another

Ex: Euclids algorithm for computing the greatest common divisor.

GCD:

$gcd(m,n) = gcd(n, m \bmod n)$

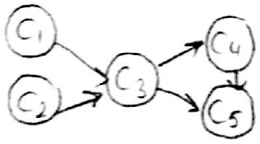
Ara tir.
SWE
Critical Path

92, 10 iain $\Rightarrow$ gcd (10, 2) = 2

Topological Sorting

prerequisite - restricted tasks

In CS, a topological sort or topological ordering of a digraph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.



C_1, C_2, C_3, C_4, C_5

C_2, C_1, C_3, C_4, C_5

En son C5 yapilabilir. Çünkü o hiabiseyi bağlamıyor

When does TS not have a feasible solution?

→ Directed cycle

TS has a feasible solution iff the graph is a DAG (Directed acycle Graph)

Algorithm 1: Perform a DFS traversed and note the order in which vertices become dead ends. (Because if it, is a dead end, no other vertex can follow it.)

Algorithm 2. Repeatedly identfy in the remaining graph a source (a vertex with no incoming edges), note it down and delete it along with all edges outgoing from it.