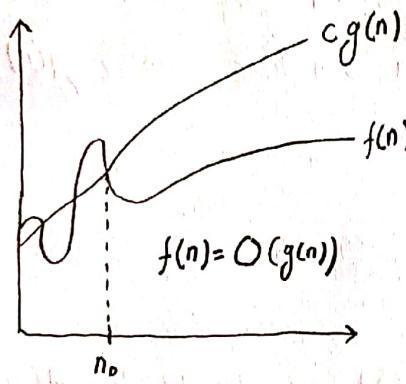


Algorithm Analysis

Asymptotic Notation & Functions & Running Times

We use asymptotic notation primarily to describe the running times of algorithms. Asymptotic notation actually applies to functions, however. The functions to which we apply asymptotic notation will usually characterize the running times of algorithms. Even when we use asymptotic notation to apply to the running time of an algorithm, we need to understand which running time we mean.

O-Notation



When we have only an asymptotic upper bound, we use O-notation. For a given function $g(n)$, we denote by $O(g(n))$ the set of functions.

$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0\}$

Since O-notation describes an upper bound, when we use it to bound the worst case running time of an algorithm, we have a bound on the running time of the algorithm on every input.

- if $f(n)$ is a polynomial of degree d then $f(n)$ is $O(n^d)$

- Use the smallest possible class of functions
say "2n is $O(n)$ " instead of "2n is $O(n^2)$ "

- if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$ then

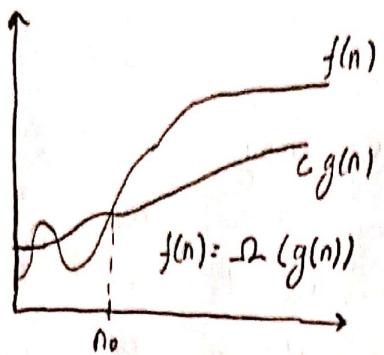
$$d(n) + e(n) \in O(f(n) + g(n))$$

$$d(n) \cdot e(n) \in O(f(n) \cdot g(n))$$

- if $d(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$ then $d(n)$ is $O(g(n))$

- if $p(n)$ is a polynomial in n then $\log p(n)$ is $O(\log n)$

Ω - Notation

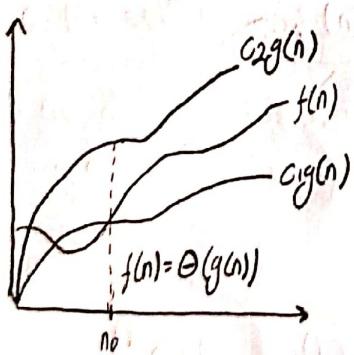


Ω - notation (omega) provides an asymptotic lower bound. For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions:

$$\Omega(g(n)) = \{f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$$

When we say that the running time of an algorithm is $\Omega(g(n))$, we mean that no matter what particular input of size n is chosen for each value of n , the running time on that input is at least a constant times $g(n)$, for sufficiently large n .

Θ -Notation



For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions.

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

We say that $g(n)$ is an asymptotically tight bound for $f(n)$.

Examples

• $2n+10$ is $O(n)$

$$\rightarrow 2n+10 \leq cn$$

$$(c-2)n \geq 10$$

$$n \geq 10/c-2$$

Pick $c=3$ and $n_0 = 10$

• n^2 is not $O(n)$

$$\rightarrow n^2 \leq cn$$

$$n \leq c$$

This inequality cannot be satisfied since c is a constant.

• $7n-2$ is $O(n)$

$$\rightarrow 7n-2 \leq cn$$

$$(7-c)n \leq 2$$

Pick $c=7$ and $n_0 = 1$

• $3n^3 + 20n^2 + 5$ is $O(n^3)$

$$\rightarrow 3n^3 + 20n^2 + 5 \leq cn^3$$

$$20n^2 + 5 \leq (c-3)n^3$$

Pick $c=4$, $n_0 = 21$

• $5n^2$ is $\Omega(n^2)$

$$\rightarrow cn^2 \leq 5n^2$$

$$c \leq 5$$

Pick $c=5$, $n_0 = 1$

• $5n^2$ is $\Omega(n)$

$$\rightarrow cn \leq 5n^2$$

$$c \leq 5n$$

Pick $c=1$, $n_0 = 1$

• $5n^2$ is $\Theta(n^2)$

$$\rightarrow c_1 n^2 \leq 5n^2 \leq c_2 n^2$$

$c_1, c_2 = 5$ and $n_0 = 1$

• $4n^2 + 2n$ is $O(n)$

$$\rightarrow 4n^2 + 2n \leq cn$$

$$4n + 2 \leq c$$

$$n \leq (c-2)/4$$

Since c is constant
inequality cannot be
achieved.

- $3n^2 - 100n + 6 = O(n^0) \rightarrow c = 3 \rightarrow 3n^2 > 3n^2 - 100n + 6$

- $3n^2 - 100n + 6 = O(n^3) \rightarrow c = 1 \rightarrow n^3 > 3n^2 - 100n + 6 \rightarrow n_0 = 4$

- $3n^2 - 100n + 6 \neq O(n) \rightarrow cn < 3n^2$ when $n > c$

- $3n^2 - 100n + 6 = \Omega(n^2) \rightarrow c = 2 \rightarrow 2n^2 < 3n^2 - 100n + 6 \rightarrow n_0 = 101$

- $3n^2 - 100n + 6 \neq \Omega(n^3) \rightarrow c = 1 \rightarrow n^3 > 3n^2 - 100n + 6 \rightarrow n > 3$

- $3n^2 - 100n + 6 = \Omega(n) \rightarrow cn < 3n^2 - 100n + 6 \rightarrow n > 100c$

- $3n^2 - 100n + 6 = \Theta(n^2)$, because both O and Ω apply.

- $3n^2 - 100n + 6 \neq \Theta(n^3)$, because only O applies.

- $3n^2 - 100n + 6 \neq \Theta(1)$, because only Ω applies.

$$\bullet f(n) = 2n+10 > f(n) \in O(g(n))$$

$$g(n) = n > g(n) \in O(f(n))$$

$$\bullet 2n+5 \text{ is } \Theta(n)$$

$$\rightarrow 2n+5 \geq c_1 n$$

$$5 \geq (c_1 - 2)n$$

$$c_1 = 2, n_0 = 1$$

$$\rightarrow 2n+10 \leq cn \rightarrow n \leq C \cdot (2n+10)$$

$$10 \leq (C-2)n \quad n \leq 2nc + 10c$$

$$10/(c-2) \leq n \quad (1-2c)n \leq 10c$$

$$\rightarrow 2n+5 \leq c_2 n$$

$$c=3, n_0 = 10 \quad C=1, n_0 = 0$$

$$5 \leq (c_2 - 2)n$$

$$c_2 = 7, n_0 = 1$$

$$\bullet 2^{n+1} \text{ is } \Theta(2^n)$$

$$\bullet (x+y)^2 \text{ is } O(x^2+y^2)$$

$$\rightarrow 2^{n+1} \leq c_1 2^n \rightarrow c_1 2^n \leq 2^{n+1} \rightarrow (x+y)^2 \leq n(x^2+y^2)$$

$$2 \cdot 2^n \leq c_1 2^n \quad c_1 2^n \leq 2 \cdot 2^n \quad x^2 + 2xy + y^2 \leq nx^2 + ny^2 \quad \text{if no } 2xy \text{ then inequality holds}$$

$$c_1 = 2, n_0 = 1 \quad c_1 = 2, n_0 = 1 \quad \text{if } x \leq y \text{ then } 2xy \leq 2y^2 \leq 2(x^2+y^2)$$

$$\text{if } y \leq x \text{ then } 2xy \leq 2x^2 \leq 2(x^2+y^2)$$

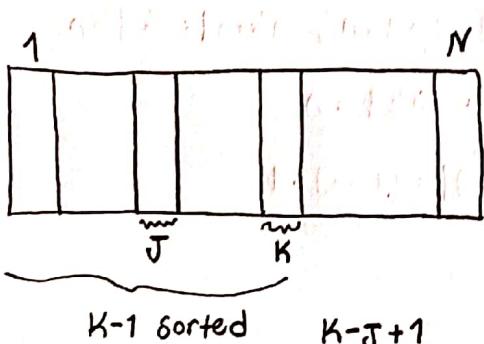
$$\text{Thus, } c = 2+1 = 3$$

$$(x+y)^2 \leq 3(x^2+y^2)$$

Insertion Sort Average Case

worst case $\longrightarrow O(n^2)$, $\Omega(n^2) \longrightarrow \Theta(n^2)$

best case $\longrightarrow O(n)$, $\Omega(n) \longrightarrow \Theta(n)$



$$\begin{aligned}
 & \frac{1}{K} \cdot \sum_{j=1}^{K-1} (K-j+1) \\
 &= \frac{1}{K} \cdot \left(K^2 - \frac{K \cdot (K+1)}{2} + K \right) \\
 &= \frac{2K^2 - K^2 - K + 2K}{2K} = \frac{K+1}{2}
 \end{aligned}$$

Number of operations
in inner while

$$\sum_{K=2}^N \frac{K+1}{2} = \frac{1}{2} \cdot \frac{(N+1) \cdot (N+2)}{2} \longrightarrow \text{Outer loop}$$

- Average Case = $\frac{1}{4} [N^2 + 3N + 2] \longrightarrow O(n^2)$

Using Limit For Comparing
Order Of Growths

$$\lim_{n \rightarrow \infty} \frac{t(n)}{g(n)} = \begin{cases} 0, & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n) \\ \text{CGR,} & \text{implies that } t(n) \text{ has same order of growth as } g(n) \\ \infty, & \text{implies that } t(n) \text{ has a larger order of growth than } g(n). \end{cases}$$

- $t(n) = \frac{1}{2}n(n-1)$, $g(n) = n^2$

$$\lim_{n \rightarrow \infty} \frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{n^2 - n}{n^2}$$

$$= \frac{1}{2} \lim_{n \rightarrow \infty} \left(1 - \frac{1}{n} \right) = \frac{1}{2}$$

Useful Formulas For The Analyses Of Algorithms

Logarithms

$$-\log_x y = y^{\log x}$$

$$-\log xy = \log x + \log y$$

$$-\log x/y = \log x - \log y$$

$$-\log_a x = \log_b a \cdot \log_b x$$

$$-a^{\log_b x} = x^{\log_b a}$$

Floor And Ceiling

$$-x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$$

$$-\lfloor x+n \rfloor = \lfloor x \rfloor + n, \quad \lceil x+n \rceil = \lceil x \rceil + n$$

$$-\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$$

$$-\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$$

Summations

$$*\sum_{i=L}^{U-1} 1 = U-L+1$$

$$*\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$$

$$*\sum_{i=1}^n i^k \approx \frac{1}{k+1} \cdot n^{k+1}$$

$$*\sum_{i=1}^n 1 = n$$

$$*\sum_{i=1}^n i^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$$

$$*\sum_{i=1}^n a^i = \frac{a^{n+1}-1}{a-1}$$

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

$$*\sum_{i=1}^n \frac{1}{i} \approx \ln n + X, \quad X \approx 0.5772$$

$$*\sum_{i=L}^U c a_i = c \cdot \sum_{i=L}^U a_i$$

$$*\sum_{i=L}^U a_i \pm b_i = \sum_{i=L}^U a_i \pm \sum_{i=L}^U b_i$$

Mathematical Analysis Of Non-Recursive Algorithms

```

Max Element (A[0..n-1])
max ← A[0]
for i ← 1 to n-1 do
    if A[i] > max
        max ← A[i]
return max

```

→ Comparisons $A[i] < \max$ ①

→ Assignments: $\max \leftarrow A[i]$ ②

$$C(n) = \sum_{i=1}^{n-1} 1 = n-1 \in O(n), \text{ where } n \text{ is number of elements.}$$

$C_{\text{worst}}(n)$ or $C_{\text{avg}}(n)$ or $C_{\text{best}}(n)$

```

Unique Elements (A[0..n-1])
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if A[i] = A[j]
            return false
    return true

```

$$\begin{aligned}
C_{\text{worst}}(n) &= \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=1}^{n-2} \left(\sum_{j=i+1}^{n-1} 1 \right) \\
&= \sum_{i=1}^{n-1} n-i-1 = \sum_{i=1}^{n-1} (n-1) - \sum_{i=1}^{n-1} i \\
&= (n-1) \cdot \sum_{i=1}^{n-1} 1 - \frac{(n-2)(n-1)}{2} = (n-1)(n-1) - \frac{(n-2)(n-1)}{2} \\
&= \frac{n \cdot (n-1)}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)
\end{aligned}$$

$$T(n) \approx C \cdot C(n) = C + n^2$$

cost of basic operations

Running time
of algorithm

Mathematical Analysis Of Recursive Algorithms

Time complexity
Space complexity

factorial(n)

if $n=0$

return 1

else

return $n * \text{factorial}(n-1)$

$$F(n) = F(n-1) \cdot n$$

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

↓ ↗ to multiply
to compute $F(n-1)$ by n
 $F(n-1)$

Recurrence Relation → Initial Condition: if $n=0$ return 1 → $m(0) = 0$

$$m(n) = n(n-1) + 1 \text{ for } n > 0$$

$$m(0) = 0$$

↑
The calls stop
when $n=0$
↑
No multiplication

Method Of Backward Substitution :

$$m(n) = n(n-1) + 1 , \text{ substitute } m(n-1) = m(n-2) + 1$$

$$= [m(n-2) + 1] + 1 = m(n-2) + 2 , \text{ substitute } m(n-2) = m(n-3) + 1$$

$$= [m(n-3) + 1] + 2 = m(n-3) + 3 ,$$

|

|

|

$$= m(n) = m(n-c) + c$$

→ $c \leftarrow n$

$$m(n-n) + n$$

$$= m(0) + n$$

$$= n //$$

binary Digit(n)

if $n=1$

return 1

else

return binary Digit($\lfloor n/2 \rfloor$)

→ Addition: $A(n) = A(\lfloor n/2 \rfloor) + 1$ for $n > 1$

→ Initial Condition: $A(1) = 0$

Smoothness rule $\longrightarrow n = 2^k$

$$A(2^k) = A(2^{k-1}) + 1 \text{ for } n > 1$$

$$A(2^0) = A(1) = 0$$

↳ Backward:

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

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

$$= A(2^{k-3}) + 3$$

⋮

$$= A(2^{k-c}) + c$$

↳ $c \leftarrow k$

$$A(2^0) + k = 0 + k = \log_2 n \in \Theta(\log n)$$

• $X(n) = X(n-1) + n, X(0) = 0$

$$= [X(n-2) + n-1] + n = X(n-2) + (n-1) + n$$

$$= [X(n-3) + n-2] + (n-1) + n = X(n-3) + (n-2) + (n-1) + n$$

⋮

$$= X(n-c) + (n-c+1) + (n-c+2) + \dots + n$$

$c \leftarrow n$

$$\underbrace{X(0) + 1 + 2 + \dots + n}_0 = \frac{n(n+1)}{2} \in \Theta(n^2)$$

• $X(n) = X(n/3) + 1$, $X(1) = 1$

$$n=3^k$$

$$X(3^k) = X(3^{k-1}) + 1$$

$$= X(3^{k-2}) + 2$$

$$\vdots \\ = X(3^{k-i}) + i$$

$$\xrightarrow{i \leftarrow k}$$

$$X(1) + k = k + 1 = \log_3 n + 1 \in \Theta(\log n)$$

• $\sum_{c=2}^{n-1} \log_2 c^2$, find the order of growth

$$= 2 \cdot \sum_{c=2}^{n-1} \log_2 c = 2 \cdot \sum_{c=1}^{n-1} \log_2 c - 2 \cdot \log_2 n$$

$$2\Theta(n \log n) - 2\Theta(\log n) = \Theta(n \log n)$$

Brute Force

Brute-force is a straight-forward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts.

$$- a^n \rightarrow \underbrace{a * a * \dots * a}_{n \text{ times}}$$

- Selection Sort and Bubble Sort.
- Linear (Sequential) search

- Brute Force String Matching
- Convex-Hull
- Closest Pair

Exhaustive Search

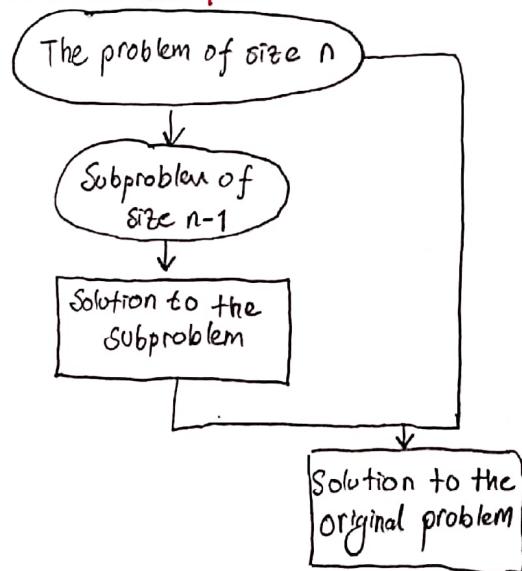
Exhaustive Search is simply a brute-force approach to Combinatorial problems.

- Traveling Salesman Problem
- Knapsack Problem

Decrease-and-Conquer

- Decrease by a constant.
- Decrease by a constant factor.
- Variable size decrease.

Decrease-by-one-and-Conquer



$$\begin{aligned} a^n &= a^{n-1} \cdot a \\ f(n) &= \begin{cases} f(n-1) * a & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases} \end{aligned}$$

Decrease-by-a-constant Factor

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even} \\ (a^{(n-1)/2})^2 * a & \text{if } n \text{ is odd} \\ 1 & \text{if } n=0 \end{cases}$$

Variable-size-decrease

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

Binary Search Algorithm

- Decrease-by-constant-factor

```
L ← 0
R ← n - 1
while L <= R do
    m ← ⌊(L+R)/2⌋
    if K = A[m]
        return m
    else if K < A[m]
        R ← m - 1
    else
        L ← m + 1
return -1
```

$$C_w(n) = C_w(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1, \quad C_w(1) = 1$$

$$C_w(2^k) = k + 1 = \log_2 n + 1$$

Comparison
every time

Divide and Conquer

In divide-and-conquer, we solve a problem recursively, applying three steps at each level of the recursion.

- Divide the problem into a number of subproblems that are smaller instances of the same problem
- Conquer the subproblems by solving them recursively. If the problem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Combine the solutions to the subproblems into the solution for the original problem.

The Master Theorem

Let $a \in \mathbb{N}^{>1}$ and $b \in \mathbb{N}^{>1}$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by the recurrence:

$$T(n) = aT(n/b) + f(n); \quad f(n) \in \Theta(n^d)$$

Then $T(n)$ has the asymptotic bounds:

- $T(n) \in \Theta(n^d)$, if $a < b^d$
- $T(n) \in \Theta(n^d \log n)$, if $a = b^d$
- $T(n) \in \Theta(n^{\log_b a})$, if $a > b^d$

$$T(n) = a \cdot T(n/b) + f(n)$$

→ Our division of the problem yields a subproblems, each of which is $1/b$ size of the original

For merge sort $D(n) \in \Theta(1)$
 $C(n) \in \Theta(n)$

→ $f(n) = D(n) + C(n)$

✓
running time of Divide and Conquer

$$A(n) = 2A(n/2) + 1$$

$$a=2$$

$$b=2$$

$$f(n) \in \Theta(n^d)$$

$$n^d=1, d=0$$

$$A(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$$

Merge Sort vs Quick Sort

Merge Sort Complexity: $\Theta(n \log n)$ → Best case
 Average case
 Worst case

Quick Sort Complexity: $\Theta(n \log n)$ → best case
 $\Theta(n^2)$ → Worst case
 $\Theta(n \log n)$ → Average case

Merge Sort

```

MergeSort(A[0..n-1], p, r)
if(p < r)
    q ← (p+r)/2
    mergeSort(A, p, q)
    mergeSort(A, q+1, r)
    merge(A, p, q, r)
  
```

```

① for c ← p to r do
    A[c] ← B[c-p]
  
```

```

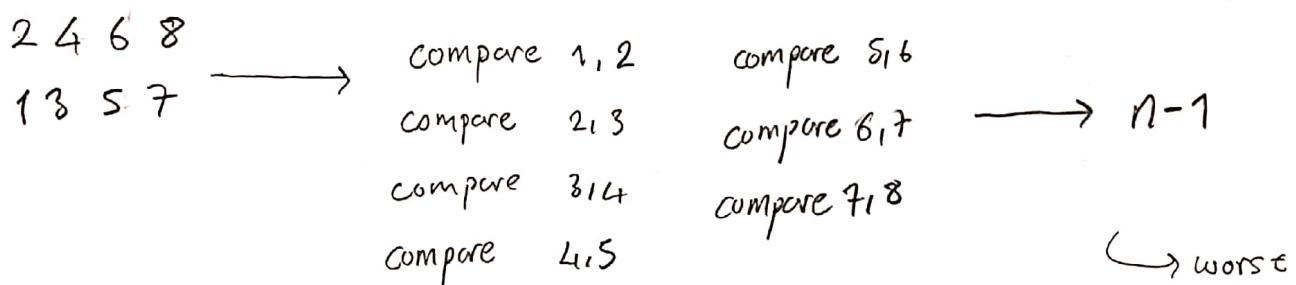
merge(A[0..n-1], p, q, r)
nβ ← r-p+1
let B be B[0..nβ]
c ← p
j ← q+1
k ← 0
while ((c ≤ q) and (j ≤ r)) do
    if A[c] < A[j]
        B[k] ← A[c]
        c ← c+1
    else
        B[k] ← A[j]
        j ← j+1
        k ← k+1
  
```

```

while c ≤ q
    B[k] ← A[c]
    k ← k+1
    c ← c+1
while j ≤ r
    B[k] ← A[j]
    k ← k+1
    j ← j+1
  
```

①

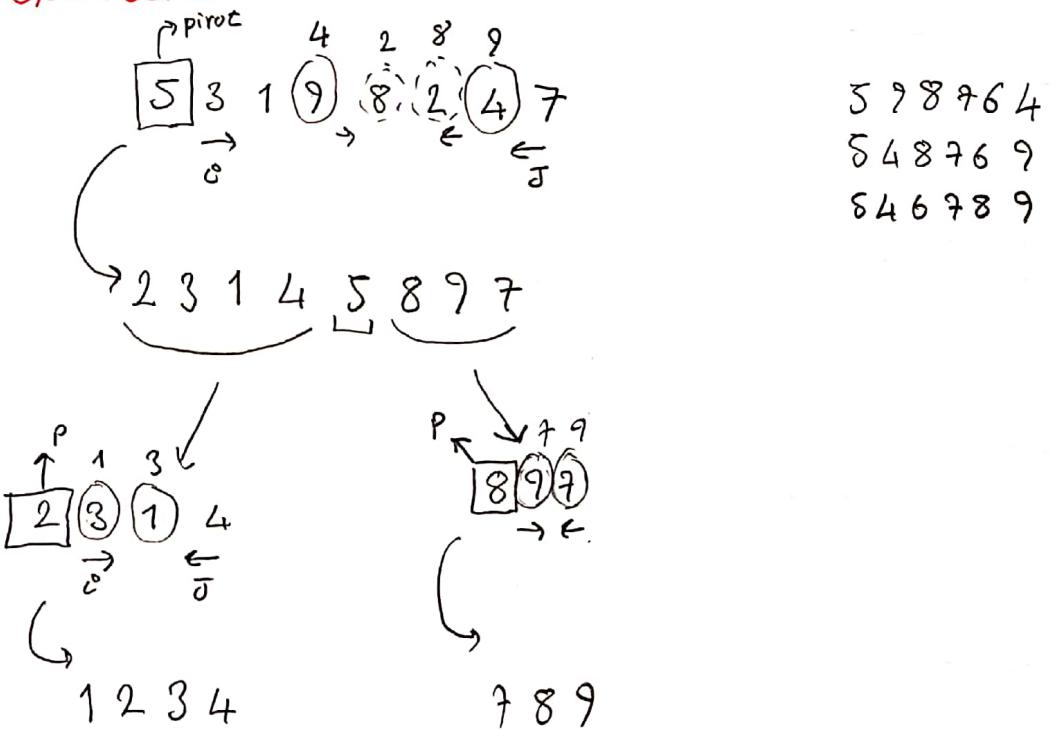
$$C(n) = 2C(n/2) + C_{\text{merge}}(n)$$



$$C_{\text{worst}}(n) = 2C_{\text{worst}}(n/2) + n - 1 \rightarrow \text{Worst case merge operation}$$

$$a=2, b=2, d=1 \rightarrow \Theta(n^d \log n) \rightarrow \Theta(n \log n)$$

Quick Sort



Quicksort ($A[l \dots r]$)

if $l < r$

$s \leftarrow \text{Partition}(A[l \dots r])$

Quicksort ($A[l \dots s-1]$)

Quicksort ($A[s+1 \dots r]$)

Average Case

- Randomly ordered array of size n

- Assuming that partition split can happen in each position s ($0 \leq s \leq n-1$) with the same probability $1/n$

$$C_{\text{avg}}(n) = \frac{1}{n} \cdot \sum_{s=0}^{n-1} [(n+1) + C_{\text{avg}}(s) + C_{\text{avg}}(n-1-s)] \quad \text{for } n > 1$$

$$C_{\text{avg}}(0) = 0$$

Hoare Partition ($A[l \dots r]$)

$p \leftarrow A[l]$

$i \leftarrow l$

$j \leftarrow r+1$

repeat

 repeat $i \leftarrow i+1$ until $A[i] \geq p$

 repeat $j \leftarrow j-1$ until $A[j] \leq p$

Until $i \geq j$

swap ($A[i], A[j]$)

swap ($A[l], A[j]$)

return j

$$C_{\text{avg}}(n) \approx 2n \ln n \approx 1.38 n \log_2 n$$

$$C_{\text{avg}}(1) = 0$$

↳ 1.38 more comparison

$$C_{\text{best}}(n) = 2C_{\text{best}}(n/2) + \hat{n}$$

$f(n)$

$\Theta(n \log n)$

(5) 9 8 7 6 4 3 2 1 pivot comparison of n times

If the pivot is the median of array

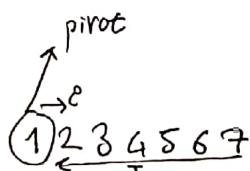
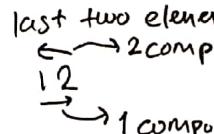
↳ means best case

$$C_{\text{worst}}(n) = (n+1) + n + (n-1) + \dots + 3$$

$$f(n) = \frac{(n+1) \cdot (n+2)}{2} - 3$$

$\in \Theta(n^2)$

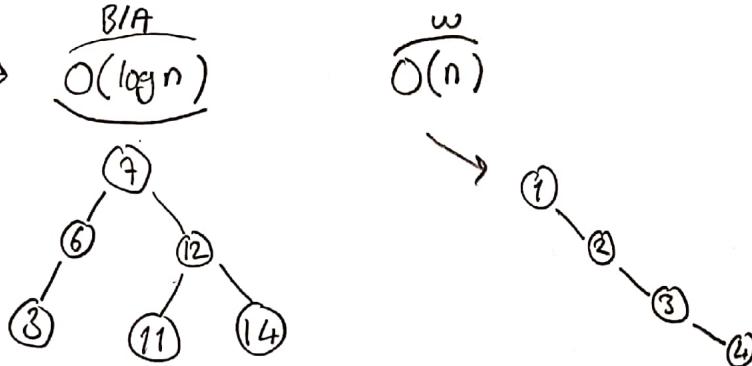
last two element



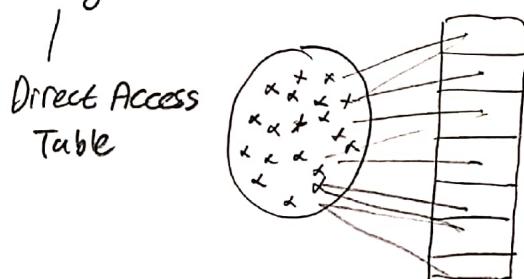
Left-to-right scan stops at $A[1]$
Right-to-left scan will go all the way to reach $A[0]$

Search Algorithms

- Linear (Sequential) Search $\rightarrow \overbrace{O(n)}^{W/A/B}$
- Binary Search (Sorted Dataset) $\rightarrow \overbrace{O(\log_2 n)}^{W/A} \quad \overbrace{O(1)}^B$
- Binary Search Tree $\rightarrow \overbrace{O(\log n)}^{B/A} \quad \overbrace{O(n)}^W$



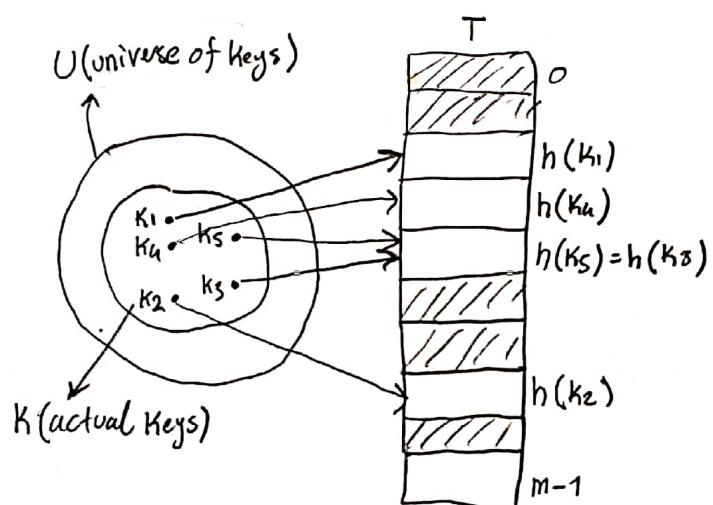
- Hashing



Hashing

Many applications require a dynamic set that supports only the dictionary operations; Insert, Search, Delete. A hash table is an effective data structure for implementing dictionaries.

Although searching for an element in a hash table can take as long as searching for an element in a linked list - $\Theta(n)$ time in worst case - in practice, hashing performs extremely well. Under reasonable assumptions, the average time to search for an element in a hash table is $O(1)$.



$$h: U \rightarrow \{0, 1, \dots, m-1\}$$

$$|T| = |K| = m \rightarrow \Theta(|K|)$$

↳ in direct addressing

$|T| = |U| \rightarrow$ much more space

A good hash function satisfies (approximately) the assumption of simple uniform hashing: each key is equally likely to hash to any of the m slots, independently of where any other key hashed too.

The Division Method

We map a key K into one of m slots by taking the remainder of K divided by m .

$$h(K) = K \bmod m \quad , \text{where } m \text{ is the size of hash table}$$

When using the division method, we usually avoid certain values of m .

For example, m should not be a power of 2, since $m=2^p$, then $h(K)$ is just the p lowest-order bits of K .

The Multiplication Method

Multiply key K by a constant A in the range of $0 < A < 1$ and extract fractional part of KA . Then multiply this value by m and take the floor of the result:

$$h(K) = \lfloor m \cdot [(K \cdot A) \bmod 1] \rfloor$$

$(K \cdot A) \bmod 1$ means the fractional part of $K \cdot A$:

$$(K \cdot A) \bmod 1 = K \cdot A - \lfloor K \cdot A \rfloor$$

In multiplication method value of m is not critical.

The optimal value of A depends on the characteristics of the data being hashed.

$$A \approx (\sqrt{5}-1)/2 = 0.6180339887\dots$$

is likely to work reasonably well.

Birthday Paradox And Hashing

$$365 \rightarrow \frac{364}{365} * \frac{363}{365} * \dots * \frac{365-(k-1)}{365}$$

↳ probability of k numbers not overlapping

Hash Table

Table size: m

$$\frac{m-1}{m} * \frac{m-2}{m} * \dots * \frac{m-(k-1)}{m} \approx \exp\left(\frac{-k(k-1)}{m}\right) \quad \left. \begin{array}{l} \text{probability of} \\ k \text{ numbers not overlapping} \end{array} \right\}$$

$$\text{Hash collision Probability} = 1 - \exp\left(\frac{-k(k-1)}{m}\right)$$

Key

- Integer
- Floating Number → Convert to integer
- String

"ELMA", "MALE", "ALME" should not generate the same address.

a) $h \leftarrow 0$ ↗ size of string
for $c \leftarrow 0 \dots S-1$ do ↗ table length
 $h \leftarrow \underbrace{(h + C + \text{ord}(c))}_{\substack{\text{↗ ASCII of char } c \\ \text{Constant}}} \text{ mod } m$

b) Horner's Method ↗ length of string
 $K = R^{L-1} * \text{str}[0] + \dots + R^0 * \text{str}[L-1]$
 $\downarrow \quad \downarrow$
31 $\text{str}[0] - 'A' + 1$

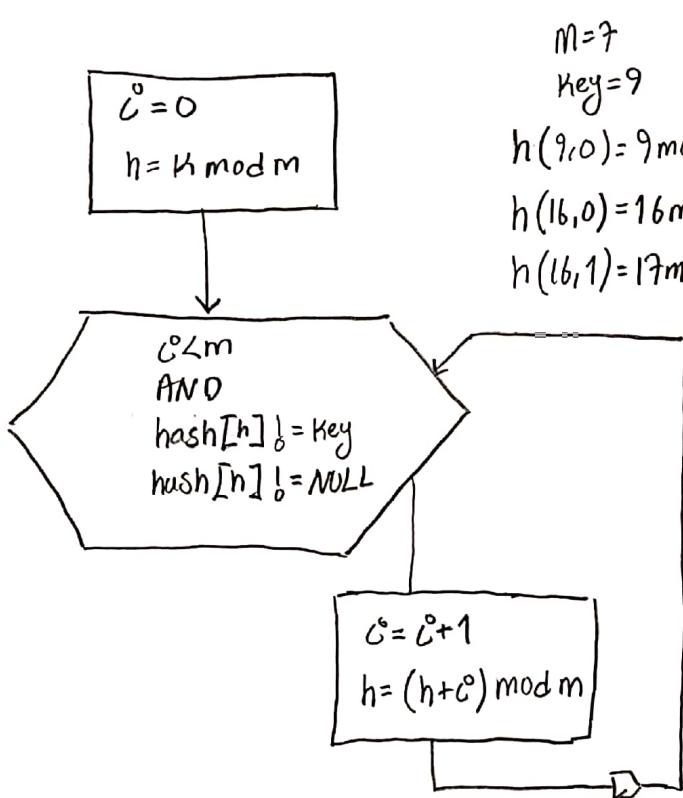
- Compound Keys

$$\text{Birthdate: 31.08.1997} \rightarrow R * 31 + R^2 * 8 + R^3 * 1997$$

Open Addressing

Linear Probing

$$h(k, c) = [h'(k) + c] \bmod m$$



$M=7$
 $\text{Key}=9$
 $h(9, 0) = 9 \bmod 7 = 2$
 $h(16, 0) = 16 \bmod 7 = 2$
 $h(16, 1) = 17 \bmod 7 = 3$

	0
	1
/ / /	9
/ / /	16
/ / /	1 / /
	4
	5
	6

- If the table is almost full, increase the table size (Rehashing).
- Delete operation $\rightarrow (-1)$: Delete flag.
- Linear probing suffers from a problem known as primary clustering. Long runs of occupied slots build up, increasing the average search time. Clusters arise because an empty slot preceded by c full slots gets filled next with probability $(c+1)/m$. Long runs of occupied slots tend to get longer, and the average search time increases.

Quadratic Probing

$$- h(k, c) = [h'(k) + c^2] \bmod m$$

$$h(9, 0) = 9 \bmod 7 = 2$$

$$h(9, 1) = 10 \bmod 7 = 3$$

$$h(9, 2) = 13 \bmod 7 = 6$$

$$- h(k, c) = [h'(k) + c_1 c + c_2 c^2] \bmod m$$

$m=7$

0
1
2
3
4
5
6

|-----|
9

This method works much better than linear probing, but to make full use of hash table, the values of c_1, c_2 and m are constrained.

Also, if two keys have the same initial probe position, then their probe sequences are the same, since $h(k_1, 0) = h(k_2, 0)$, implies $h(k_1, c) = h(k_2, c)$. This property leads to a milder form of clustering, called secondary clustering.

Double Hashing

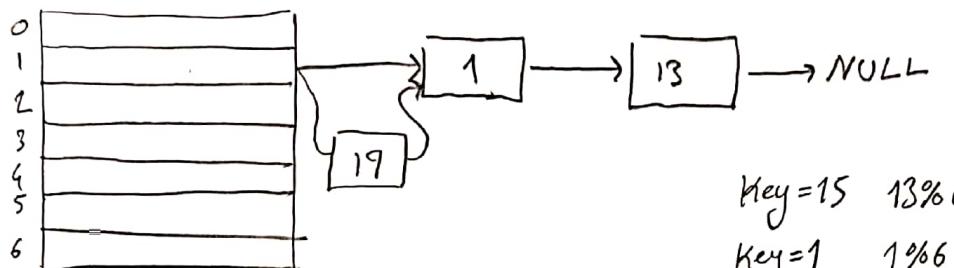
$$h(k, c) = [h_1(k) + c * h_2(k)] \bmod m$$

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod m'), \quad m' < m$$

Double hashing offers one of the best methods available for open addressing because the permutations produced have many of the characteristics of randomly chosen permutations.

Separate Chaining



$$\text{Key} = 15 \quad 15 \% 6 = 1$$

$$\text{Key} = 1 \quad 1 \% 6 = 1$$

$$\text{Key} = 19 \quad 19 \% 6 = 1$$

Largest Common Subsequence

$X: ATCTGAT$

$Y: TGCGATA$

$$LCS(X, Y) = TCTA$$

$$LCS(X, Y) = TGAT$$

KOZALAK

KALV

KOLV

KOLAV

KALKV

	B	D	C	B
B	0	0	0	0
A	0	1	1	1
C	0	1	1	2 → 2
D	0	1	2 → 2	2 → 2
B	0	1	2	2 → 3

$$LCS(X, Y) = BCB$$

$$LCS(X, Y) = BD\bar{B}$$

if $i=0 \rightarrow C[i, j]=0$

if $j=0 \rightarrow C[i, 0]=0$

$$C[i, j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ C[i-1, j-1] + 1 & i, j > 0, X_i = Y_j \\ \max\{C[i-1, j], C[i, j-1]\} & X_i \neq Y_j \end{cases}$$

while ($i \neq 0 \text{ and } j \neq 0$) {

switch ($B[i, j]$) {

case ADD:

add $X[i]$ to front of LCS

$i--;$

$j--;$

break;

case SKIPX:

$i--;$

break;

case SKIPPY:

$j--;$

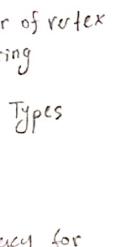
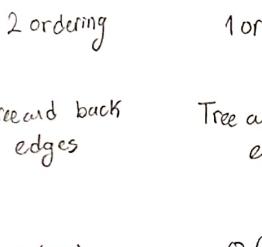
break;

}

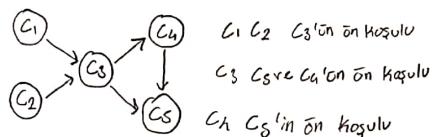
\rightarrow

$x_i, y_j \rightarrow C[i, j]$

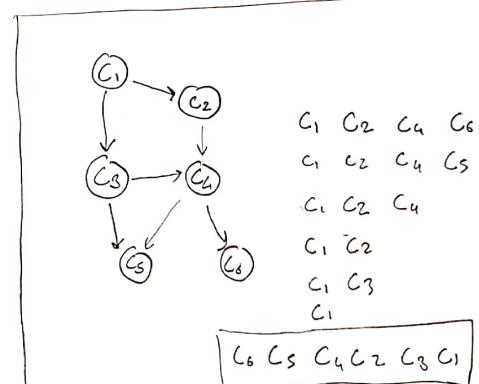
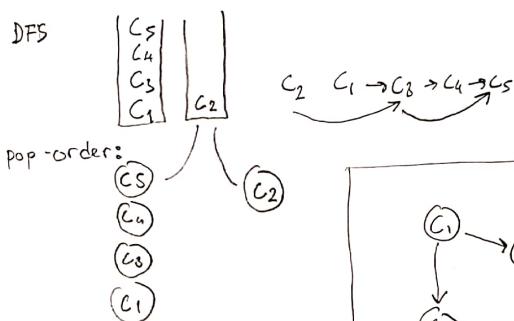
x_n, y_m

	<u>DFS</u>	<u>BFS</u>
Data Structure	Stack	Queue
Number of vertex ordering	2 ordering	1 ordering
Edge Types	Tree and back edges	Tree and cross edges
Efficiency for adjacency matrix	$\Theta(V ^2)$	$\Theta(V ^2)$
Efficiency for adj linked list	$\Theta(V + E)$	$\Theta(V + E)$
	→	<p style="text-align: right;">Decrease and comp topological sort</p> 
$C_1 \ C_2 \ C_3 \ C_4 \ C_5 \ C_6$	$(C_5)(C_6) \leftarrow$	$(C_4) \downarrow \uparrow \rightarrow (C_6)$

Topological Sorting



DFS
or
Decrease and conquer



Backtracking

- DFS with pruning

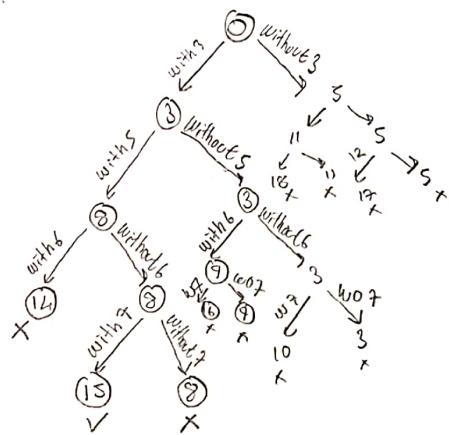
- Exhaustive search

- State-space-tree ↵
 - ↳ promising
 - ↳ non-promising

Subset sum problem

$$S = \{3, 5, 6, 7\}, d = 15$$

$$s_1 \leq s_2 \leq s_3 \leq \dots \leq s_n$$



N-queen (8-queen)

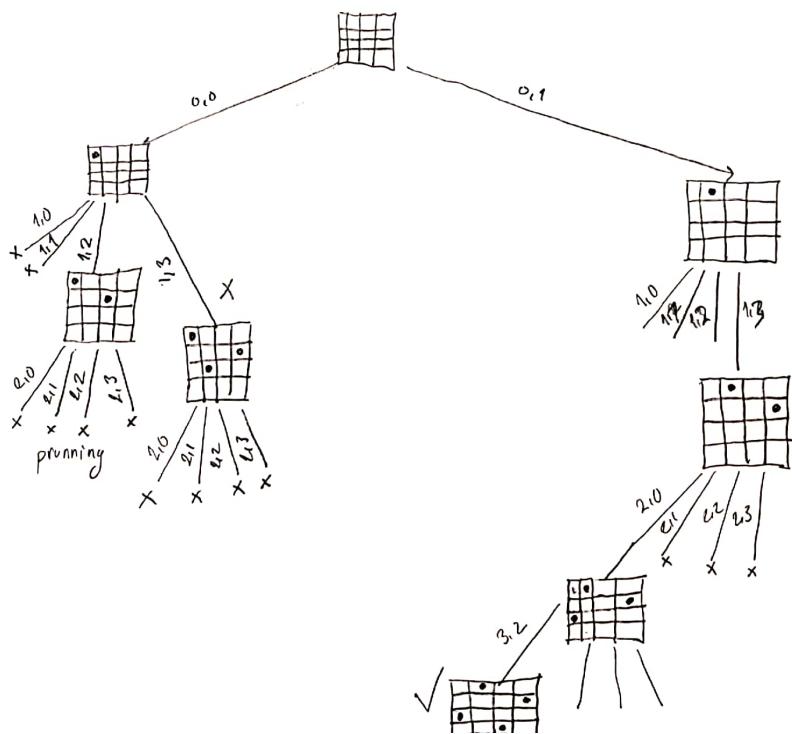
$$\text{Brute force} \rightarrow \frac{64!}{81,861} = 4 \text{ milyar}$$

$$\text{One per row} \rightarrow 8^8 = 16\text{ m}$$

One per row and col $\rightarrow 8! = 40,320$

Backtracking → 2057

4-queen problem (minimum)



Branch and Bound

-optimization

- ↳ maximize
 - ↳ minimize
 - ↳ objective function
-] under certain constraints

- Feasible solution

- Optimal solution

Matrix yerine array

a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
2	3	0	0	

↳ teknik

Complexity

$$1 + C_1 + C_1 \cdot C_2 + C_1 C_2 C_3 + \dots + C_1 C_2 \cdots C_n$$

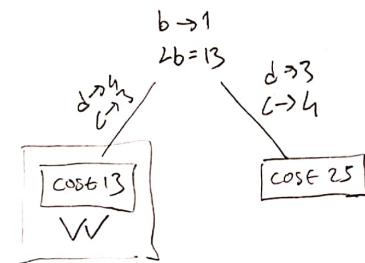
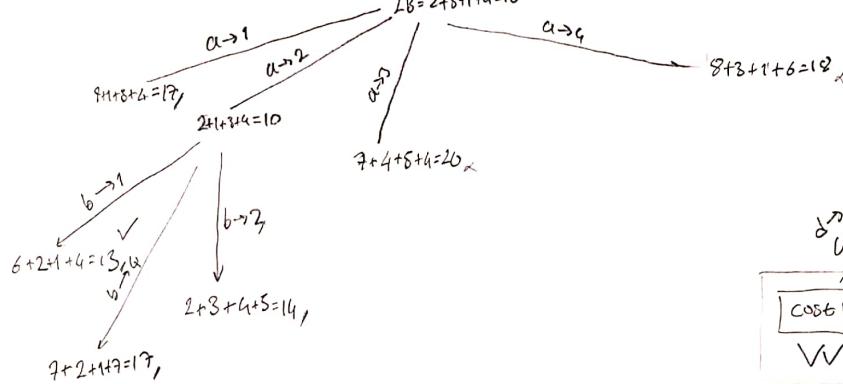
Assignment Problem

	Job 1	Job 2	Job 3	Job 4	
person a	9	2	7	8	
person b	6	4	3	7	
person c	5	8	1	8	
person d	7	6	9	4	

Lower bound

Start

$$LB = 2 + 3 + 1 + 4 = 10$$



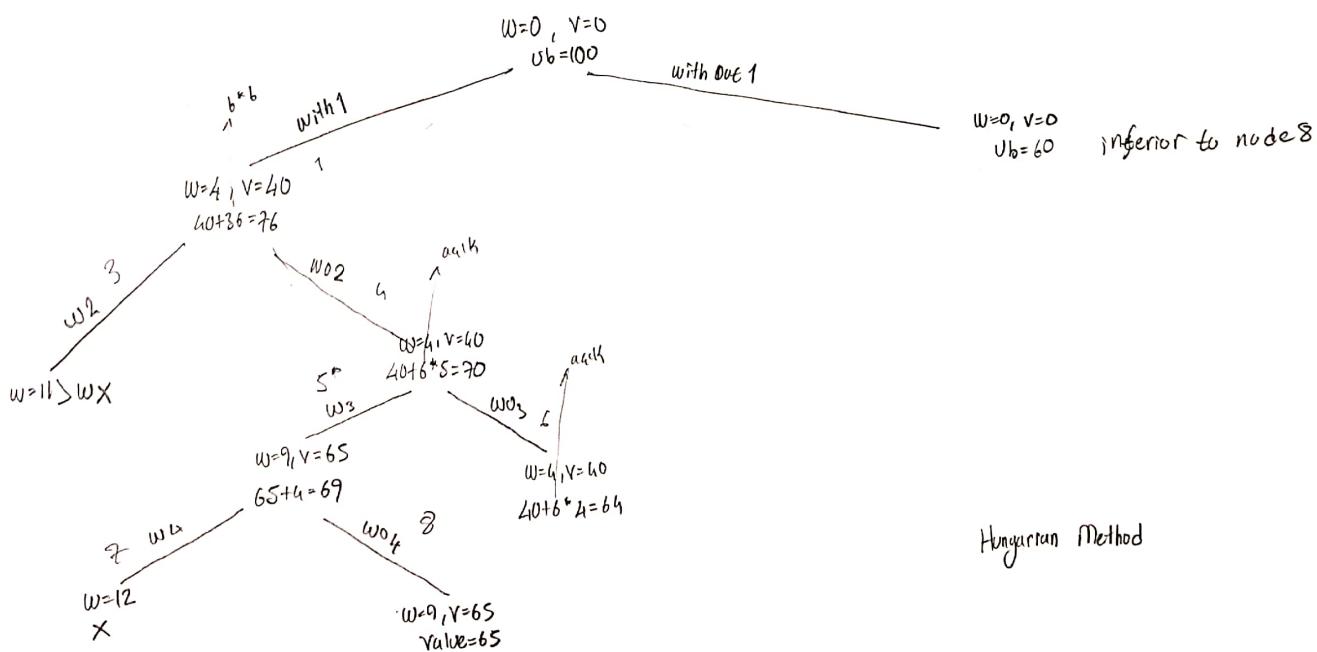
Knapsack

item	weight	value	Value/weight	
1	4	60	10	
2	7	62	6	
3	5	25	5	
4	3	12	4	

$W=10$

$V_1/W_1 \geq V_2/W_2 \geq \dots \geq V_n/W_n$

$10 \text{ kg} \rightarrow 100$



P, NP, NP-Complete, NP-Hard

polynomial \rightarrow tractable

not solved
in polynomial \rightarrow intractable

undecidable \longrightarrow Halting Problem (undecidable) \longrightarrow A: algorithm, P: program, I: input

decidable

$$A(P, I) = \begin{cases} 1 & \text{if program } P \text{ halts on input } I \\ 0 & \text{" " " " not halts on input } I \end{cases}$$

$$\Theta(P) = \begin{cases} \text{halts} & \text{if } A(P, P) = 0 \\ \text{does not halt} & \text{if } A(P, P) = 1 \end{cases}$$

$$\Theta(\alpha) = \begin{cases} \text{halts} & A(\alpha, \alpha) = 0 \\ \text{does not halt} & A(\alpha, \alpha) = 1 \end{cases} \longrightarrow \text{NP-hard}$$