Recap

Sorting Algorithms

Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)

Algoritme	Best time complexity	Average time complexity	Worst time complexity	Worst space complexity
Linear search	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Binary search	$O(1)$	$O(logn)$	$O(logn)$	$O(1)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge sort	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(n)$
Quick sort	$O(nlogn)$	$O(nlogn)$	$O(n^2)$	$O(logn)$
Heap sort	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(n)$
Radix sort	$O(d(n+k))$	$O(d(n+k))$	$O(d(n+k))$	$O(n+k)$
Counting sort	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(k)$

Insertion Sort (Page: 16):

Worst-case: This is if the array is sorted backwards. Then it has to check and sort every number. (n^2)

Best-case: if the array is already sorted.

Runtime if *n* is the same: O(n)

Merge Sort (Page: 29):

Runtime if *n* is the same: O($nlogn$)

Divide and conquer

- Divide the n element sequence to be sorted into two sub-sequences of n/2 elements each
- Conquer: Sort the two sub-sequences recursively using merge sort
- Combine: Merge the two sorted sub-sequences to produce the sort answer
- Divide: **O(1)**. Constant time
- Conquer: Recursively solve two subproblems, each of size n/2 which contributes 2T/(n/2) to the running time
- Combine: Merge procedure on an n-element sub array takes O(n) and so C(n)= **O(n)**

Quicksort (Page: 170):

Runtime if *n* is the same: O(n^2)

Heapsort (Page: 151):

Runtime if *n* is the same: O(n)

Counting Sort(Page:):

Radix Sort (Page:):

Asymptotic Analysis

Så vi ønsker et værktøj til at sammenligne funktioners essentielle voksehastighed på en måde så der ses bort fra multiplikative konstanter.

Mere præcist: vi ønsker for voksehastighed for funktioner sammenligninger svarende til de fem klassiske ordens-relationer:

$$\leq \quad \geq \quad = \quad < \quad >$$

De vil, af historiske årsager, blive kaldt for:

$$O \quad \Omega \quad \Theta \quad o \quad \omega$$

Hvilket udtales således:

"O", "Omega", "Theta", "lille o", "lille omega"

Følgende definitioner har vist sig at fungere godt:

Man kan nemt vise at disse definitioner opfører sig som forventet af ordens-relationer. F.eks.:

$$f(n) = o(g(n)) \Rightarrow f(n) = O(g(n)) \quad (\text{jvf. } x < y \Rightarrow x \leq y)$$

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n)) \quad (\text{jvf. } x = y \Rightarrow x \leq y\ )$$

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)) \quad (\text{jvf. } x \leq y \Leftrightarrow y \geq x\ )$$

$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n)) \quad (\text{jvf. } x < y \Leftrightarrow y > x\ )$$

$$f(n) = O(g(n)) \text{ og } f(n) = \Omega(g(n)) \Rightarrow f(n) = \Theta(g(n))$$
$$(\text{jvf. } x \leq y \text{ og } x \geq y \Rightarrow x = y\ )$$

Disse regler forklarer at følgende funktioner er sat i stigende voksehastighed (den ene er $o()$ af den næste):

$$1, \quad \log n, \quad \sqrt{n}, \quad n, \quad n \log n,$$
$$n\sqrt{n}, \quad n^2, \quad n^3, \quad n^{10}, \quad 2^n$$

De asymptotiske forhold mellem de fleste funktioner f og g kan afklares ved følgende sætninger:

$$\text{Hvis } \frac{f(n)}{g(n)} \to k > 0 \text{ for } n \to \infty \text{ så gælder } f(n) = \Theta(g(n))$$

$$\text{Hvis } \frac{f(n)}{g(n)} \to 0 \text{ for } n \to \infty \text{ så gælder } f(n) = o(g(n))$$

Eksempler:

$$\frac{20n^2 + 17n + 312}{n^2} = \frac{20 + 17/n + 312/n^2}{1} \to \frac{20 + 0 + 0}{1} = 20 \text{ for } n \to \infty$$

$$\frac{20n^2 + 17n + 312}{n^3} = \frac{20/n + 17/n^2 + 312/n^3}{1} \to \frac{0 + 0 + 0}{1} = 0 \text{ for } n \to \infty$$

Big O:

Express upper bound of algorithm. It measures the worst case time.

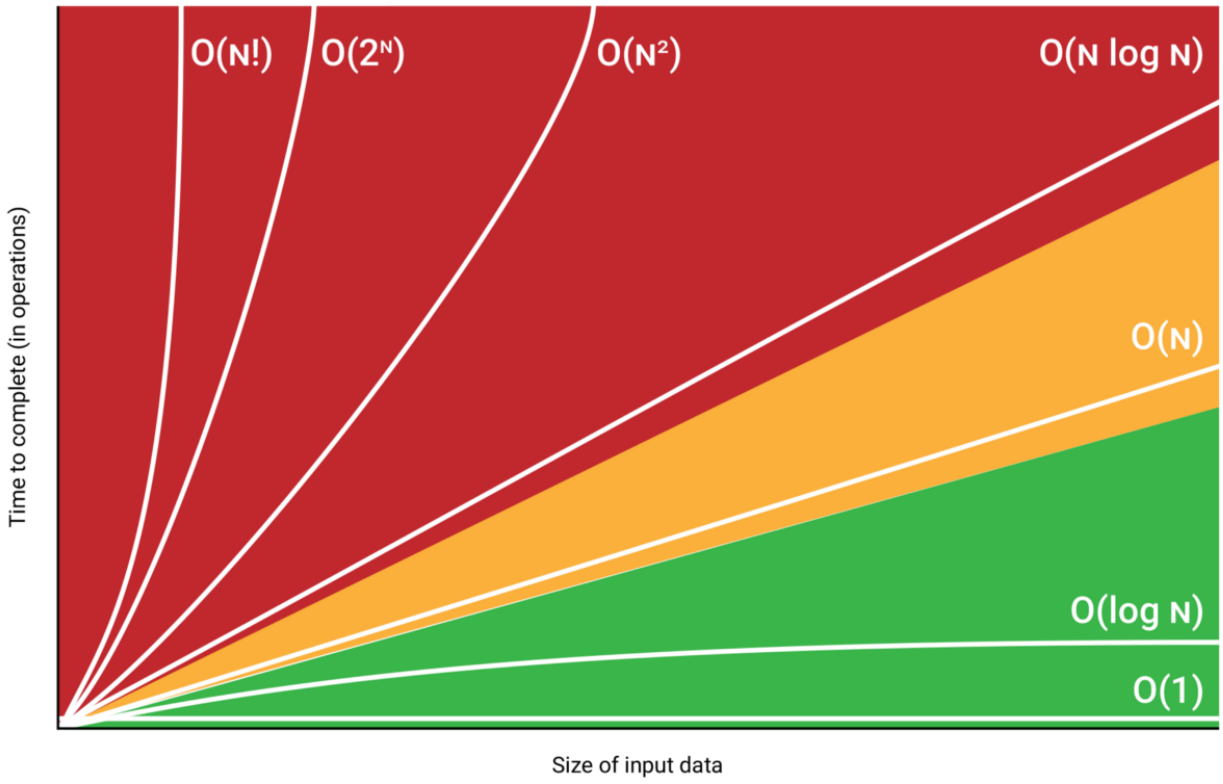
Omega Notation (Ω):

Express lower bound. Measure best case.

Theta Notation (θ):

Both lower bound and upperbound.n^2

$f = \Theta(g)$	f grows at the same rate as g	There exists an n_0 and constants $c_1, c_2 > 0$ such that for all $n > n_0$, $c_1 g(n) \leq \lvert f(n) \rvert \leq c_2 g(n)$.
$f = O(g)$	f grows no faster than g	There exists an n_0 and a constant $c > 0$ such that for all $n > n_0$, $\lvert f(n) \rvert \leq cg(n)$.
$f = \Omega(g)$	f grows at least as fast as g	There exists an n_0 and a constant $c > 0$ such that for all $n > n_0$, $cg(n) \leq \lvert f(n) \rvert$.
$f = o(g)$	f grows slower than g	For all $c > 0$, there exists an n_0 such that for all $n > n_0$, $\lvert f(n) \rvert \leq cg(n)$.
$f = \omega(g)$	f grows faster than g	For all $c > 0$, there exists an n_0 such that for all $n > n_0$, $cg(n) \leq \lvert f(n) \rvert$.
$f \sim g$	f/g approaches 1	$\lim_{n \to \infty} f(n)/g(n) = 1$



$4 = O(3) : true$

$n^4 = O(n^3) : false$

$3^n = O(n^4) : false$

$$n^{1/4} = O(n^{1/3})$$ ←**true**

$n(logn)^4 = O(n^4 logn) : true$

Hvilke af nedenstående udsagn er sande? [*Et eller flere svar.*]

Result: 5,00 of 5,00

Your answer:

- 5.a: n^3 er $O(n^2)$
- 5.b: $\log n$ er $O(n^{1/2})$
- 5.c: 1 er $O(n^{1/3})$
- 5.d: $n^{3/2}$ er $O(n \log n)$
- 5.e: 1.5^n er $O(n^{15})$
- 5.f: $n \log n$ er $O(n(\log n)^3 + n^{1/3})$

Hvilke af nedenstående udsagn er sande? [*Et eller flere svar.*]

Result: 3,33 of 5,00

Your answer:

6.a: $n^{1/2} + 2n^2 + (\log n)^2$ er $\Theta(n^2)$

Heap

Forskellige implementationer af prioritetskøer

	Heap	Usorteret liste	Sorteret liste
EXTRACT-MAX	$O(\log n)$	$O(n)$	$O(1)$
BUILD	$O(n)$	$O(1)$	$O(n \log n)$
INCREASE-KEY	$O(\log n)$	$O(1)$	$O(n)$
INSERT	$O(\log n)$	$O(1)$	$O(n)$

Heap: Ordered binary tree

Max heap: parent > child

Time: $O(nlogn)$

Trees

Always: $m = n - 1$

Binary Search Trees

height = log n

Operation	Balanced	Unbalanced
Search	O(log n)	O(n)
Minimum	O(log n)	O(n)
Maximum	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)
In-order Tree **Walk**	O(n)	O(n)

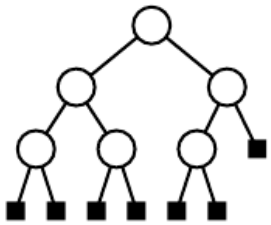
Red-Black trees balanced ops:

Search	O(log n)
Insert	O(log n)
Remove	O(log n)
In-order Tree **Walk**	O(n)

Red-Black Properties:

1. Every node is either red or black
2. The root is black
3. Every leaf (NIL) is black
4. If a node is red, then both its children are black
5. For each node, all simple paths from node to leaf contain same number of black nodes.

På hvor mange forskellige måder kan knuderne i nedenstående træ farves, så træet bliver til et lovligt rød-sort træ?



Q: 2 ways

Graphs

Not always this equation in a graph: $0 \leq n \leq m^2$

A DAG does not have any cycles

All weighted, oriented graphs does not necessarily have an MST

Prim algorithm a greedy algorithm

Prim's time complexity: $O(mlogn)$

Kruskal's time complexity: $O(mlogn)$

Bellman-Ford-Moore

does not need positive weighted edges (kanter)

Solves single shortest paths problem where edges are negative

Runtime complexity: $O(nm)$

Not greedy

At most iterations:|V| -1 edges

Breadth First Search

Data-structure: Queue (FIFO)

Black = visited

Grey = to be visited

Time-complexity: $O(n+m)$ or $O(n^2)$

Depth First Search

Data-structure: Stack

Black = visited

Grey = discovered

DFS does not guarantee to find the shortest path.

Time Complexity: $O(n+m)$

Dijkstra

Time Complexity: $O(mlogn)$

Shortest path to node

Must not contain negative edge weight

Greedy

Huffman

To find number of bits for a whole file:

Lav et Huffman-træ for en fil, hvis tegn har nedenstående hyppigheder:

Tegn	a	b	c	d	e
Hyppighed	5	25	10	30	15

Hvor mange bits fylder filen, hvis den kodes ved hjælp af dette træ?

1. Run program with right chars and observations

```
e -> 00
a -> 010
c -> 011
b -> 10
d -> 11
```

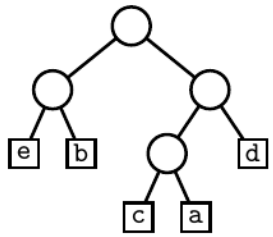
2. Multiply number of bits for each char with its right observation

   $3 * 5 + 2 * 25 + 3 * 10 + 2 * 30 + 2 * 15 = 185$

   Result is then 185 bits

To find code of specific string from tree

Med følgende Huffman-træ



hvad bliver kodningen af strengen `abbed`?

Move to right = 1

Move to left = 0

d = 11

a = 101

c = 100

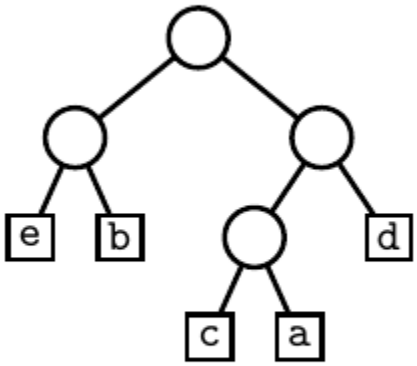
b = 01

e = 00

Result: 101 01 01 00 11

To decode a bit string from a tree

Med følgende Huffman-træ



hvad bliver dekodningen af strengen 111011001010100?

Move to right = 1

Move to left = 0

d = 11

a = 101

c = 100

b = 01

e = 00

Bit string: 11 101 100 101 01 00

Result = dacabe

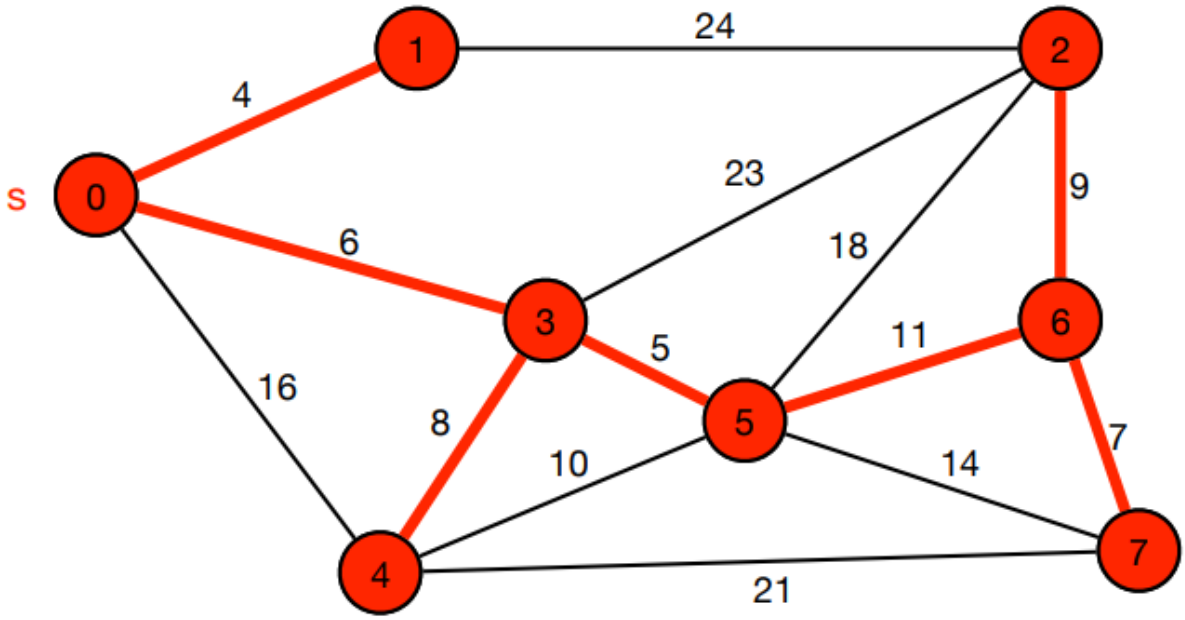
Prim

Time complexity = $O(mlogn)$

Greedy algorithm

MST

In each step, add `lightest` edge with one endpoint in Tree.



Cannot contain cycles

Results in MST

Kruskal

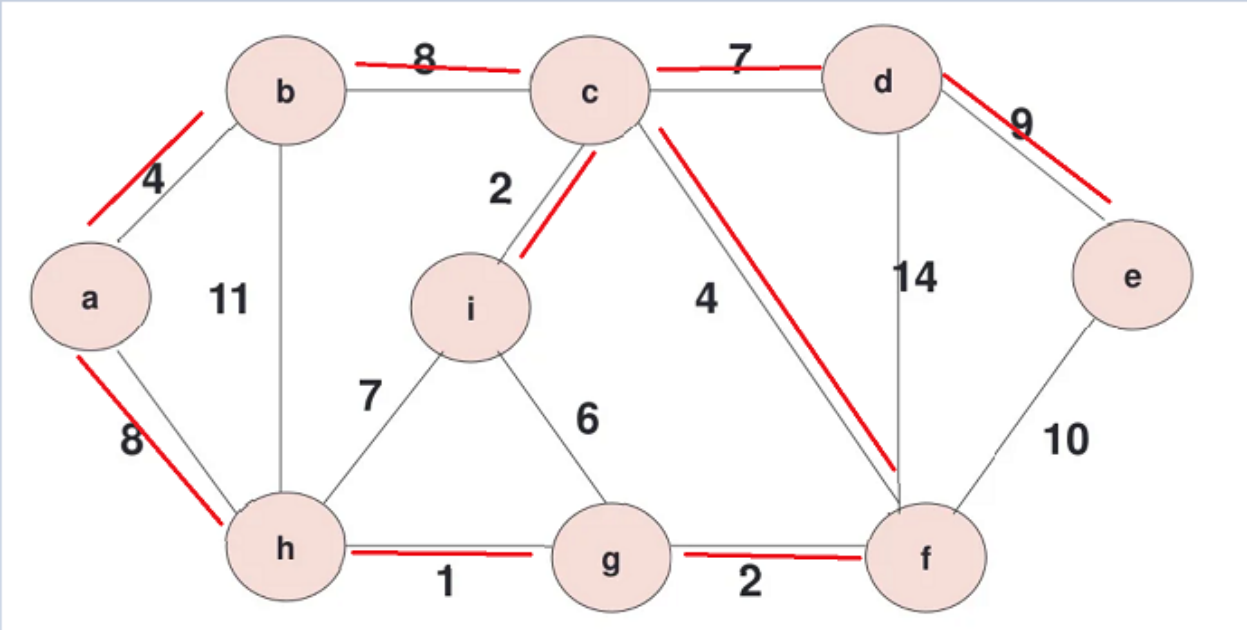
Time complexity = $O(mlogn)$

Greedy algorithm

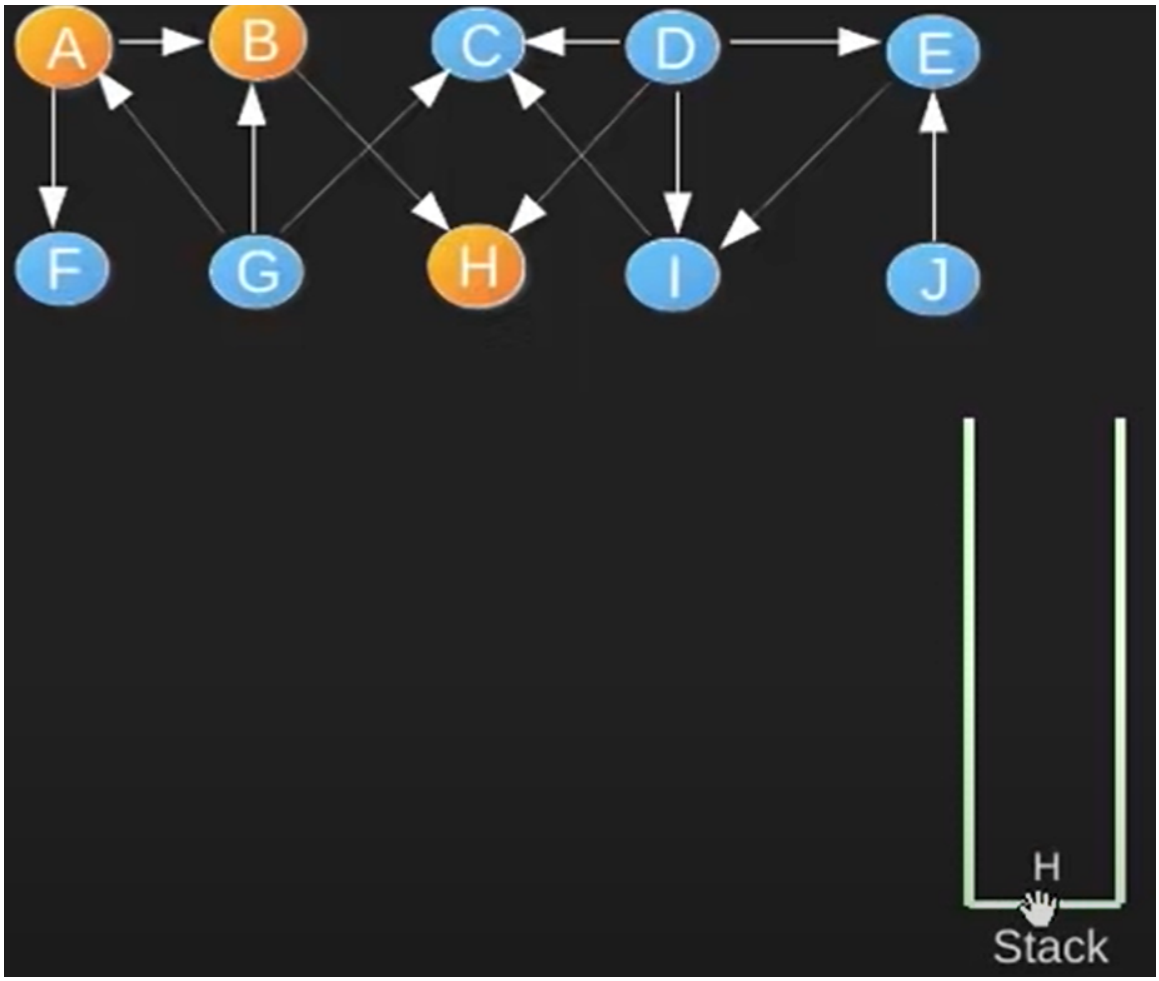
MST

Select the lightest edge/kant

Results in MST

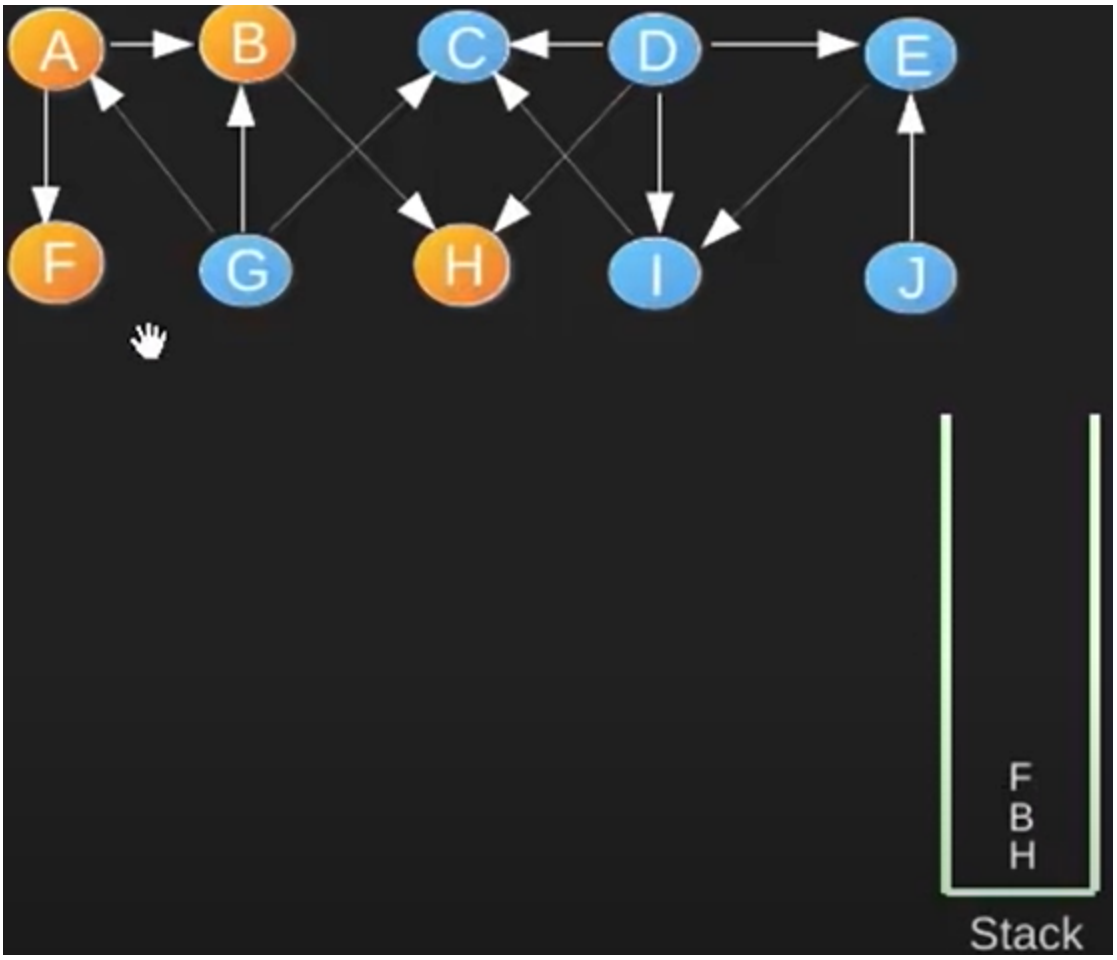


Topological Sorting



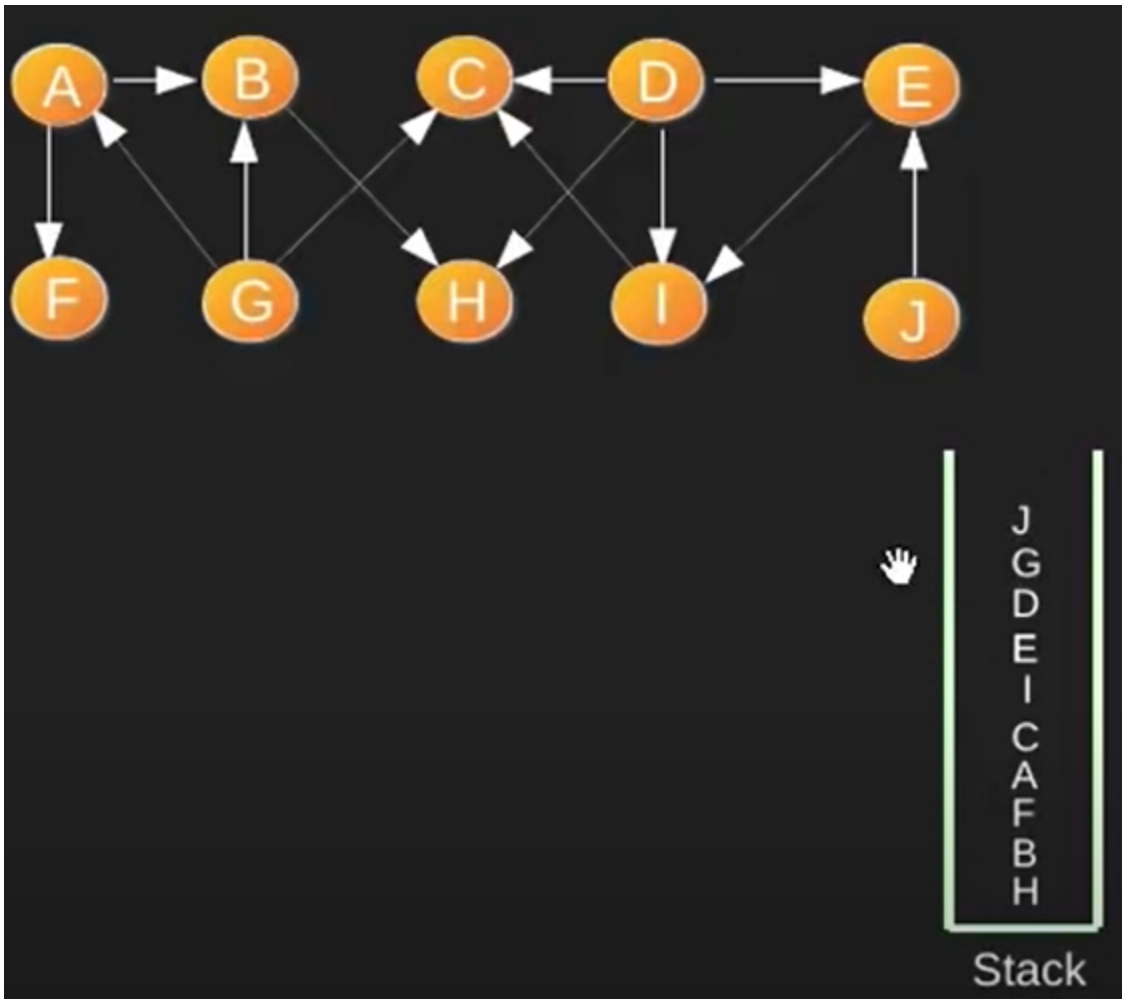
Start in A, go to neighbor B which goes to its arrow H. Not more arrows from H. H gets added to stack

B has not other neighbors and gets added to the stack while A searches F which has no arrows and gets added to the stack



A then gets added because no unsearched neighbors.

Then we start with node C and do the same thing again until final result



Hashing

Problem type 1

Vi ser på en hashtabel H, der bruger double hashing og en auxiliary hash-funktioner $h_1(x)$ og $h_2(x)$. Hashtabellen ser lige nu sådan ud:

	0	1	2	3	4	5	6
H:	33		27	32	55		47

Derefter indsættes 99, hvorefter hashtabellen ser sådan ud:

	0	1	2	3	4	5	6
H:	33	99	27	32	55		47

Hvis $h_1(99) = 2$, hvilke af følgende værdier af $h_2(99)$ er da mulige? [*Et eller flere svar.*]

To solve this kind of problems use the following formula

$$h(k,i) = (h_1(k) + i * h_2(k)) \bmod \mathrm{m}$$

$$\mathrm{m} = \text{length of array}$$

$$h_1(k) = 2$$

> We can see that 99 should be placed on index 1. Therefore we need to look at all the possible answers to compute the following formula:

$$(2 + i * option) \bmod 7$$

Where $i = iteration$

Options:

$$10.\text{a:} \quad h_2(99) = 1$$

$$10.\text{b:} \quad h_2(99) = 2$$

$$10.\text{c:} \quad h_2(99) = 3$$

$$10.\text{d:} \quad h_2(99) = 4$$

$$10.\text{e:} \quad h_2(99) = 5$$

$$10.\text{f:} \quad h_2(99) = 6$$

We know that there is an available spot on index 5. If any of our calculations hit 5 before 1 then the option is not the right answer.

Problem type 2

For en funktion $h'(x)$ gælder

$$\begin{aligned} h'(7) &= 2 \\ h'(9) &= 2 \\ h'(13) &= 3 \\ h'(17) &= 4 \end{aligned}$$

Vi ser på en hashtabel H, der bruger linear probing med ovennævnte funktion $h'(x)$ som auxiliary hashfunktion. Startende med en tom hashtabel er tallene $\{7, 9, 13, 17\}$ blevet indsat i en eller anden rækkefølge. Resultatet er blevet:

	0	1	2	3	4	5	6	7	8	9	10
H:			9	7	17	13					

Hvilke af nedenstående rækkefølger for indsættelse kan der være tale om? [*Et eller flere svar.*]

To solve this kind of problems we need to look at the order H is sorted in: 9 7 17 13.

We use the following formula to calculate the correct position for the numbers:

Linear hashing:
$h(k, i) = (h'(k) + i) \mod m$

We have to plot in the different values for the $h'(x)$ function at the top.

Calculation for $h'(7) = 2$ **:**

$$(2 + 0) \bmod 7 = 2$$

We can see that 7 should be plot into index 2. But we need to maintain order **9 7 17 13** Therefore, we cannot have any order where 7 is in the front. It needs to start with a 9

Next we can try to calculate $h'(7) = 2$ again but with an iteration of 1:

$$(2 + 1) \bmod 7 = 3$$

Therefore, we can plug 7 into index 3.

The same thing happens if we try to calculate $h'(13) = 3$ before $h'(17) = 4$

Then the order will be wrong.

So the possible orders are:

9 7 17 13

9 17 7 13

Problem type 3

Vi ser på en hashtabel H, der bruger linear probing og en auxiliary hash-funktion $h'(x)$. Hashtabellen ser lige nu sådan ud:

	0	1	2	3	4	5	6
H:	12		10		22		31

Derefter indsættes 97, hvorefter hashtabellen ser sådan ud:

	0	1	2	3	4	5	6
H:	12	97	10		22		31

Hvilke af følgende værdier af $h'(97)$ er mulige? [*Et eller flere svar.*]

To solve this kind of problem we need the same formula:

Linear hashing:
$h(k, i) = (h'(k) + i) \mod m$

We then look at the top array and try the different positions:

$(0 + 0) \bmod 7 = 0$

But index 0 is full. We try with i+1

$(0 + 1) \bmod 7 = 1$

Then $h'(97) = 0$ is a valid solution

We can again try with index 1

$(1 + 0) \bmod 7 = 1$

Since it fits in place 1 we can accept $h'(97) = 1$ as a solution as well.

The others won’t fit because the first free space they will find is index 3 and 5

But index 6 will fit into index 1 after 2 iterations:

$(6 + 2) \bmod 7 = 1$

So we accept $h'(97) = 6$ as a solution as well.

Linear (Page 272)

$$h(k, i) = (h'(k) + i) \bmod m$$

Quadratic (Page 272)

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m \ ,$$

Double (Page 272)

$$h(k, i) = (h_1(k) + i h_2(k)) \bmod m \ ,$$

Master Theorem (Page: 94)

Master Theorem

Rekursionsligningen

$$T(n) = aT(n/b) + f(n)$$

har følgende løsning, hvor $\alpha = \log_b a$:

1. Hvis $f(n) = O(n^{\alpha-\epsilon})$ for et $\epsilon > 0$ så gælder $T(n) = \Theta(n^{\alpha})$.
2. Hvis $f(n) = \Theta(n^{\alpha})$ så gælder $T(n) = \Theta(n^{\alpha} \log n)$.
3. Hvis $f(n) = \Omega(n^{\alpha+\epsilon})$ for et $\epsilon > 0$ så gælder $T(n) = \Theta(f(n))$.

Ekstra betingelse: I case 3 skal der også gælde, at der findes et $c < 1$ og et n_0 som opfylder at $a \cdot f(n/b) \leq c \cdot f(n)$ når $n \geq n_0$.

> Kort sagt: case afgøres af forholdet mellem voksehastighederne for $f(n)$ og for n^{α}.

Er de ens, har vi case 2. Er $f(n)$ mindre (med mindst en faktor n^{ϵ}), har vi case 1. Er $f(n)$ større (med mindst en faktor n^{ϵ}), har vi case 3 (hvis ekstrabetingelsen er opfyldt.).

Master Theorem brugt på de tre eksempler

Eksempel 1:

$$T(n) = 2T(n/2) + n$$

- $a = 2$
- $b = 2$
- $f(n) = n$
- $\alpha = \log_2 2 = 1$
- $f(n) = n = \Theta(n^1) = \Theta(n^{\alpha})$

Så vi er i case 2, så der gælder $T(n) = \Theta(n^{\alpha} \log n) = \Theta(n \log n)$.

Master Theorem brugt på de tre eksempler

Eksempel 2:

$$T(n) = 3T(n/2) + n$$

- $a = 3$
- $b = 2$
- $f(n) = n$
- $\alpha = \log_2 3 = \ln(3)/\ln(2) = 1.5849\ldots$
- $f(n) = n = O(n^{1.5849\cdots-\epsilon}) = O(n^{\alpha-\epsilon})$ for f.eks. $\epsilon = 0.1$.

Så vi er i case 1, så der gælder $T(n) = \Theta(n^{\alpha}) = \Theta(n^{1.5849\ldots})$.

Master Theorem brugt på de tre eksempler

Eksempel 3:

$$T(n) = 3T(n/4) + n^2$$

- $a = 3$
- $b = 4$
- $f(n) = n^2$
- $\alpha = \log_4 3 = \ln(3)/\ln(4) = 0.7924\ldots$
- $f(n) = n^2 = \Omega(n^{0.7924\cdots+\epsilon}) = \Omega(n^{\alpha+\epsilon})$ for f.eks. $\epsilon = 0.1$.

Så vi er i case 3, så der gælder $T(n) = \Theta(f(n)) = \Theta(n^2)$.

I case 3 skal vi dog også checke den ekstra betingelse: Med $c = 3/16 < 1$ og $n_0 = 1$ opfyldes at $a \cdot f(n/b) = 3(n/4)^2 = 3/16 \cdot n^2 \leq c \cdot f(n) = c \cdot n^2$ for alle $n \geq n_0$.

Induction

Mathematical induction works if you meet three conditions:

1. For the questioned property, is the set of elements infinite?
2. Can you prove the property to be true for the first element?
3. If the property is true for the first k elements, can you prove it true of $k + 1$?