SORTERINGSALGORITMER

1. Insertion Sort

   • Går fra venstre mot høyre i lista. • For hvert nytt element ser du på alle elementene til venstre for det, og "skyver det" inn på riktig plass i den sorterte delen. • Som å sette inn et nytt kort på riktig plass i en hånd med kort. • Effektiv for små lister eller nesten sorterte data. • Kjøretid: $O(n^2)$, stabil, in-place.

2. Selection Sort

   • Du finner det minste elementet i hele listen og flytter det til første plass. • Så finner du det nest minste og flytter det til andre plass. • Fortsetter til alt er sortert. • Som å velge den minste resten gang på gang. • Kjøretid: $O(n^2)$, ikke stabil, in-place.

3. Bubble Sort

   • Du går gjennom listen flere ganger og bytter naboer hvis de står i feil rekkefølge. • Store elementer "bobler" mot høyre for hver passering. • Stopper når ingen bytter skjer. • Kjøretid: $O(n^2)$, stabil, in-place, men ineffektiv.

4. Merge Sort

   • Deler listen i to halvparter helt til hver del har ett element. • Slår så sammen (merger) delene igjen mens du sorterer underveis. • Sammenligner alltid første element i hver del og flytter det minste over i en ny liste. • Krever ekstra plass til midlertidige lister. • Kjøretid: $O(n \log n)$, stabil, ikke in-place.

5. Quick Sort

   • Velger et tilfeldig "pivot"-element. • Deler resten av listen i to: de som er mindre enn pivot, og de som er større. • Sorterer hver del rekursivt og setter pivot mellom dem. • Som å bygge opp sorteringen rundt et midtpunkt. • Kjøretid: $O(n \log n)$ i gjennomsnitt, $O(n^2)$ i verste fall, ikke stabil, in-place.

6. Heap Sort

   • Bygger først en heap (prioritetskø) av alle elementene. • Tar så ut det minste (eller største) elementet og legger det bakerst i arrayet. • Gjentar til alt er sortert. • Som å stadig hente ut den største steinen fra en haug. • Kjøretid: $O(n \log n)$, ikke stabil, in-place.

7. Bucket Sort / Radix Sort

   • Bucket: Fordel elementene i bøtter etter intervall (som "10-spann"). Sorter hver bøtte separat (f.eks. med insertion sort). • Radix: Sorter tall på hver sifferposisjon (minste siffer først) med en stabil sorteringsalgoritme. • Begge er raske når dataene har begrenset område. • Kjøretid: $O(n + k)$, stabile, ikke in-place.

TREDATASTRUKTURER

9. Binært søketre (BST) – innsetting

   • Går nedover i treet fra roten. • Hvis elementet er mindre enn noden, gå venstre; hvis større, gå høyre. • Når du kommer til et tomt sted, sett inn der. • Kjøretid: O(h), der h er høyden (O(log n) i balansert tre).

10. AVL-tre (balansert BST)

    • Etter hver innsetting sjekker du balansefaktoren (høyde venstre – høyre). • Hvis forskjellen blir ±2, utfør rotasjon: ○ LL: høyre rotasjon ○ RR: venstre rotasjon ○ LR: venstre + høyre ○ RL: høyre + venstre • Holder treet høyde-balansert slik at søk og innsetting forblir O(log n).

11. Heap (Min-Heap / Max-Heap)

    • Representert som et array. • For hver node er verdien mindre (eller større) enn begge barna. • Ved innsetting: legg til bakerst og "heapify up" ved å bytte med foreldre. • Ved fjerning: bytt første og siste element, fjern siste, og "heapify down". • Kjøretid: O(log n) for innsetting og fjerning.

GRAFALGORITMER

12. DFS (Depth-First Search)

    • Start i en node, og gå så langt du kan ned i grafen før du backtracker. • Bruker stack (implisitt rekursjon). • Finner komponenter, brukes til topologisk sortering og SCC. • Kjøretid: O(|V| + |E|).

13. BFS (Breadth-First Search)

    • Utforsker grafen lagvis, først alle naboer, så naboers naboer. • Bruker kø. • Finner korteste vei i uvektet graf. • Kjøretid: O(|V| + |E|).

14. Dijkstra

    • Finner korteste vei i vektet graf uten negative kanter. • Starter fra en node, og bruker prioritetskø for å alltid besøke den nærmeste ubesøkte noden. • Oppdaterer avstander når det finnes en kortere vei. • Kjøretid: O((|V| + |E|) log |V|).

15. Bellman-Ford

    • Fungerer selv med negative vekter. • Går gjennom alle kanter |V|−1 ganger og oppdaterer avstander. • Oppdager negative sykler hvis det fremdeles finnes kortere vei etter |V|−1 runder. • Kjøretid: O(|V|·|E|).

16. Prim (minimalt spenntre)

    • Start med én node og bygg et tre ved å stadig legge til den billigste kanten som forbinder treet med en ny node. • Bruker prioritetskø. • Kjøretid: O(|E| log |V|).

17. Kruskal (minimalt spenntre)

    • Sorter alle kanter etter vekt. • Legg dem til i stigende rekkefølge, men bare hvis de ikke danner en sykel (bruk Union-Find). • Stopper når du har |V|−1 kanter. • Kjøretid: O(|E| log |V|).

18. Topologisk sortering

    • Bruk DFS og legg noder på stack når alle naboer er ferdig besøkt. • Pop stacken for å få en ordning. • Bare mulig for rettede asykliske grafer (DAGs). • Kjøretid: O(|V| + |E|).

19. Kosaraju (Sterkt sammenhengende komponenter)

    • Kjør DFS og legg noder på stack etter ferdig-tid. • Reverser alle kanter. • Kjør DFS på nytt i rekkefølgen fra stacken. • Hver DFS gir en SCC. • Kjøretid: O(|V| + |E|).

EKSTRA

20. Huffman-koding

    • Lag en prioritetskø av symboler etter frekvens. • Ta ut de to minste, slå dem sammen til ny node med summert frekvens. • Sett den nye noden tilbake i køen og gjenta til ett tre gjenstår. • Bruk venstre=0, høyre=1 for å få binære koder. • Kjøretid: O(n log n).

21. Hashing (med kollisjonshåndtering)

    • Separate chaining: hver bøtte i tabellen inneholder en liste. Kollisjoner legges i samme liste. • Linear probing: hvis plassen er opptatt, prøv neste indeks (rundt modulo n). • Kjøretid: O(1) forventet, O(n) i verste fall.

Sortering

Bubble / Insertion / Selection Sort

Brukes når:

```
• Små datasett.
• Du må forstå grunnidéen bak sortering (undervisningsalgoritmer).
• Du bryr deg ikke om effektivitet, men enkelhet.
```

Hvordan skille:

```
• Bubble: Sammenligner naboer og “bobler” opp største elementer → treg
(O(n²)).
• Insertion: Setter hvert nytt element på riktig plass i den sorterte delen
→ grei for nesten sortert input.
• Selection: Finner minste for hvert steg → få bytter, men mange
sammenligninger.
```

Merge Sort

Brukes når:

- Du må sortere store mengder data stabilt og effektivt (O(n log n)).
- Du har tilgang til deling av datastrukturen (rekursivt delt).

Nøkkelord: “Del og hersk”-algoritme. Deler listen i to → sorterer hver del → fletter sammen.

Quick Sort

Brukes når:

- Du vil ha rask sortering i praksis (ofte raskere enn Merge sort).
- Du kan akseptere ustabilitet og at det kan være dårlig i verste fall ($O(n^2)$).

Nøkkelord: “Pivot” → alt mindre til venstre, alt større til høyre. God for arrays som passer i minne.

Heap Sort

Brukes når:

- Du allerede bruker heap som datastruktur (f.eks. prioritetskø).
- Du vil ha O(n log n) uten ekstra minnebruk.

Nøkkelord: Bygger en heap → trekker ut største/mindre element én etter én.

Søking og trær

Binary Search

Brukes når:

- Dataen er sortert.
- Du må finne et element raskt i en tabell → O(log n).

Nøkkelord: “Del søket i to” – hopper midt i stedet for sekvensielt.

Binary Search Tree (BST)

Brukes når:

- Du vil lagre data slik at du kan søke, sette inn og slette raskt (O(h)).

Nøkkelord: Venstre barn mindre, høyre barn større.

AVL-tre

Brukes når:

- Du trenger garantert rask søk/innsetting/sletting (balansert høyde).

Nøkkelord: BST + høydebalansering. Automatisk rotasjon for å holde treet balansert.

Heap (Min/Max)

Brukes når:

- Du trenger rask tilgang til største/minste elementet.
- Typisk brukt i prioritetskøer (Dijkstra, scheduling, etc.).

Nøkkelord: "Forelder er alltid mindre/større enn barna." Ikke BST – heap handler om rekkefølge, ikke full sortering.

Hashing

Hash Map / Hash Table

Brukes når:

- Du vil ha ekstremt rask oppslagstid (O(1) i snitt).
- Du trenger kobling nøkkel → verdi (dictionary, symboltabell).

Nøkkelord:

- Bruker hashfunksjon og kollisjonsstrategi

→ Linear probing (flytt videre) → Separate chaining (liste per bøtte).

Grafalgoritmer

DFS (Depth-First Search)

Brukes når:

- Du vil finne alle noder som kan nås (f.eks. komponenter).
- Du trenger rekursiv utforskning – dypest mulig først.

- Du skal sjekke sykler eller topologisk sortering.

Nøkkelord: Bruker stack eller rekursjon.

BFS (Breadth-First Search)

Brukes når:

- Du vil finne korteste vei i uvektet graf.
- Du vil besøke noder lagvis fra en startnode.

Nøkkelord: Bruker kø, ikke stack.

Topologisk Sortering

Brukes når:

- Grafen er rettet og asyklisk (DAG).
- Du skal finne rekkefølge på oppgaver / avhengigheter.

Nøkkelord: "Sorter etter hva som må komme først." → F.eks. fag med forkunnskaper.

Dijkstra

Brukes når:

- Du skal finne korteste vei i graf med positive vekter.
- Negativ vekt = (bruk Bellman–Ford i stedet).

Nøkkelord: Grådig algoritme med prioritetskø (heap).

Bellman-Ford

Brukes når:

- Grafen kan ha negative vekter.
- Du også vil oppdage negative sykler.

Nøkkelord: Relakser alle kanter |V|−1 ganger.

Prim

Brukes når:

- Du vil bygge minimalt spenntre (MST) fra én startnode.
- Du bygger det gradvis, ett punkt om gangen.

Nøkkelord: “Dijkstra for MST.”

Kruskal

Brukes når:

- Du har kant-fokusert graf (liste av kanter).
- Du vil lage MST ved å legge til kanter én etter én uten å lage sykler.

Nøkkelord: Bruker sortering av kanter + union-find.

Borůvka

Brukes når:

- Du har svært store grafer.
- Du vil parallellisere MST-bygging (hver node starter som eget tre).

Nøkkelord: Alle trær velger samtidig sin billigste kant → repeter.

Kosaraju (SCC)

Brukes når:

- Du vil finne sterkt sammenhengende komponenter i en rettet graf.
- Hver SCC = “gruppe av noder i samme rundtur”.

Nøkkelord: 2 DFS + reversert graf.

Kompresjon og søk

Huffman-koding

Brukes når:

- Du vil komprimere data uten tap (lossless).
- Symboler har ulik frekvens (tekst, lyd, etc.).

Nøkkelord: Bruk prioritetskø for å bygge et binærtre basert på frekvens.

. .

IN2010 — Eksamensnotat

SORTERING Algoritme Når brukes Intuisjon Kjøretid Stabil? Insertion Sort Små/lite endrede datasett Setter hvert nytt element der det hører hjemme $O(n^2)$ Ja Selection Sort Enkleste form – bytte minst mulig Finner minste, flytter fremover $O(n^2)$ Nei Bubble Sort Undervisnings-eksempel Bobler største element oppover $O(n^2)$ Ja Merge Sort Store data, stabil sortering "Del og flett" rekursivt O(n log n) Ja Quick Sort Når minne er viktig, praktisk rask Velg pivot → del opp → rekursivt O(n log n) snitt / $O(n^2)$ verst Nei Heap Sort Når du har heap tilgjengelig Bygg heap, trekk ut maks/min gjentatt O(n log n) Nei Counting/Bucket Sort Tall i begrenset område Teller forekomster O(n + k) Ja

Huskeregler:

- Stabil sortering bevarer rekkefølge for like elementer.
- Quick = raskest i praksis, Merge = tryggest i teori.

TRÆR OG HEAPS Struktur Bruksområde Idé Viktig egenskap Operasjoner Binary Search Tree (BST) Dynamisk sortert datastruktur Venstre < rot < høyre Rekkefølge, men ikke balansert Søk/sett/slett: O(h) AVL-tre Raskere BST med balanse Holder høydeforskjell ≤ 1 Rotasjoner for balanse Alt: O(log n) Heap (min/max) Prioritetskø (Dijkstra, scheduling) Forelder < (eller >) barn Komplett binært tre Insert/delete: O(log n) Binomial heap Fleksibel heap for sammenslåing Flere heap-trær i "binær representasjon" Hurtig merge O(log n) Fibonacci heap Avansert heap for Dijkstra Amortisert raskere enn binomial Cut + mark Insert: O(1), delete-min: O(log n) 💡 Heaps brukes når du alltid vil hente ut minimum raskt.

HASHING Type Når brukes Idé Kollisjoner Kjøretid Hash Map / Table Oppslag (dictionary) h(k) → indeks Linear probing (flytt videre) / Chaining (liste) O(1) snitt Linear probing Færre cache misses Lete fremover ved kollisjon Kan bli "clusters" O(1) snitt Separate chaining Mange kollisjoner ok Liste per bøtte Flere noder per hash O(1) snitt

Hash = raskt oppslag, men husk lastfaktor $\alpha < 1$ for å holde O(1).

GRAFER OG TRAVERSERING Algoritme Bruksområde Intuisjon Kjøretid DFS Utforskning, sykler, komponenter Følg sti så dypt du kan før du backtracker O(V + E) BFS Korteste vei i uvektet graf Lagvis utforskning O(V + E) Topologisk sortering Rekkefølge i DAG Finn noder med in-degree 0 først O(V + E) Kosaraju / Tarjan (SCC) Sterkt sammenhengende komponenter 2 DFS (eller lavest nåbar) O(V + E)

DFS = dypest først, BFS = bredest først.

KORTESTE VEI-ALGORITMER Algoritme Når brukes Håndterer negative vekter? Idé Kjøretid BFS Uvektet graf – Naboer lagvis O(V + E) Dijkstra Positivt vektet graf Nei Grådig: velg nærmeste ubesøkte node (heap) O(E log V) Bellman-Ford Kan ha negative vekter Ja Relakser alle kanter O(V·E) DAG shortest path DAG Ja Topologisk rekkefølge O(V + E)

"Relaksere" = forbedre estimert avstand hvis ny vei er kortere.

MINIMALE SPENNTRÆR (MST) Algoritme Når brukes Idé Kjøretid Prim Bygg MST gradvis fra én node Legg til billigste kant som kobler ny node O(E log V) Kruskal Når du har kantliste Sorter kanter → legg til hvis ikke sykel (Union-Find) O(E log V) Borůvka Veldig store grafer, parallell mulig Hver node velger billigste kant til annet tre O(E log V)

Prim = nodefokusert, Kruskal = kantfokusert.

TILBAKESØK OG HEURISTIKKER Algoritme Når brukes Idé Kjøretid Backtracking Søke gjennom mulige løsninger (Sudoku, n-queens) DFS som backtracker når valget feiler Eksponentiell Branch and Bound Optimaliseringsproblemer DFS med grensekutt basert på heuristikk Eksponentiell, men raskere Greedy algoritmer Lokalt beste valg gir global løsning F.eks. Huffman, Dijkstra, Prim Ofte O(n log n)

Greedy = velg alltid billigste nå, håp det lønner seg.

TEKSTSØK OG KOMPRESJON Algoritme Bruksområde Idé Kjøretid Huffman Kompresjon Kombiner minst hyppige symboler O(n log n)

Huffman = prioritetskø over frekvenser.

KOMPLEKSITET OG TEORI Begrep Forklaring Eksempel P Løses i polynomiell tid Dijkstra, Merge Sort NP Kan verifiseres i polynomiell tid Sudoku, SAT NP-hard Minst like vanskelig som NP TSP NP-komplett I NP og NP-hard 3SAT, Hamiltonian Cycle Reduksjon Oversett problem A → B i polynomisk tid 3SAT → VC Beslutningsproblem Ja/nei-problem “Finnes det en vei kortere enn k?”

MINNE

```
• “Korteste vei?” → Dijkstra (positiv), Bellman–Ford (negativ)
• “MST?” → Prim (node), Kruskal (kant)
• “Rundturer / komponenter?” → Kosaraju
• “Rekkefølge?” → Topologisk sort
• “Uvektet korteste vei?” → BFS
• “Utforskning eller sykel?” → DFS
• “Kompresjon?” → Huffman
• “Prioritet?” → Heap
• “Oppslag?” → Hash
• “Balanse?” → AVL
```

Mentalt eksamenstips

```
• Husk at sensor vil se forståelse, ikke ren gjengivelse.
• Skriv: “Vi bruker Dijkstra fordi kantenes vekter er positive og vi ønsker
korteste vei fra én node til alle.”
• Alltid: idé + hvorfor + kjøretid = maks poeng.
• Ikke overtenk pseudokode. Du kan skrive i naturlig språk:
```

“For hver nabo av u, oppdater hvis vi finner en kortere vei.”