

Algoritmer og datastrukturer

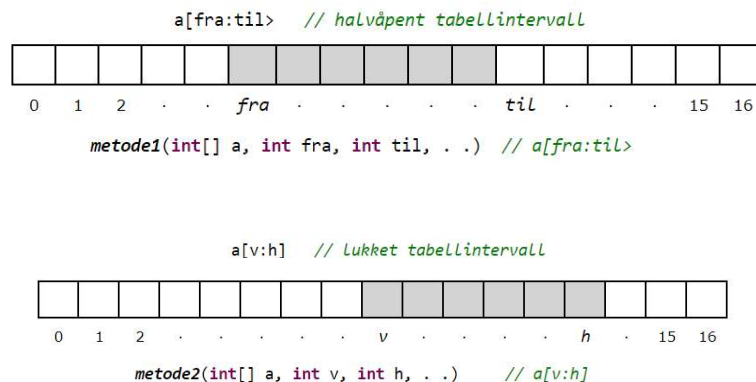
Hjelpeslides

Husk at disse ikke nødvendigvis dekker 100% av pensum!

Grunnleggende tema

- Kapittel 1: Grunnleggende begreper og teknikker
 - 1.1 Algoritmer og effektivitet: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
 - 1.2 Nest største tall: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13
 - 1.3 Ordnete tabeller: 1, 2, 4, 5, 6, 8, 9, 11
 - 1.4 Generiske algoritmer: 1, 2, 3, 4, 5, 6, 7, 8, 9
 - 1.5 Rekursjon: 1, 2, 3, 4, 5, 6, 7, 8, 9
 - 1.6 Multidimensjonelle tabeller: 1, 2, 3
 - 1.8: Algoritmeanalyse: 1, 2, 3, 4

Intervaller



Algoritmeanalyse

```
public static int maks(int[] a) // versjon 2 av maks-metoden
{
    int m = 0; // indeks til største verdi
    int maksverdi = a[0]; // største verdi

    for (int i = 1; i < a.length; i++) if (a[i] > maksverdi)
    {
        maksverdi = a[i]; // største verdi oppdateres
        m = i; // indeks til største verdi oppdateres
    }
    return m; // returnerer indeks/posisjonen til største verdi
} // maks
```

- Konstant tid – utføres uavhengig av datastørrelse
- Tre operasjoner (to assignment, en indeksring)

Algoritmeanalyse

```
public static int maks(int[] a) // versjon 2 av maks-metoden
{
    int m = 0; // indeks til største verdi
    int maksverdi = a[0]; // største verdi

    for (int i = 1; i < a.length; i++) if (a[i] > maksverdi)
    {
        maksverdi = a[i]; // største verdi oppdateres
        m = i; // indeks til største verdi oppdateres
    }
    return m; // returnerer indeks/posisjonen til største verdi
} // maks
```

- Lineær tid – utføres n ganger for n elementer
- $2*n-1$ operasjoner
(en assignment, n-1 sammenlikninger og n-1 inkrementer)

Algoritmeanalyse

```
public static int maks(int[] a) // versjon 2 av maks-metoden
{
    int m = 0; // indeks til største verdi
    int maksverdi = a[0]; // største verdi

    for (int i = 1; i < a.length; i++) if (a[i] > maksverdi)
    {
        maksverdi = a[i]; // største verdi oppdateres
        m = i; // indeks til største verdi oppdateres
    }
    return m; // returnerer indeks/posisjonen til største verdi
} // maks
```

- Lineær tid – utføres n ganger for n elementer
- $2*n$ operasjoner
(n indekseringer og n sammenlikninger)

Algoritmeanalyse

```
public static int maks(int[] a) // versjon 2 av maks-metoden
{
    int m = 0; // indeks til største verdi
    int maksverdi = a[0]; // største verdi

    for (int i = 1; i < a.length; i++) if (a[i] > maksverdi)
    {
        maksverdi = a[i]; // største verdi oppdateres
        m = i; // indeks til største verdi oppdateres
    }
    return m; // returnerer indeks/posisjonen til største verdi
} // maks
```

- Hvor mange ganger utføres denne? Logaritmisk tid!
- I beste tilfellet (største verdi først), kun én gang
- I verste tilfellet (sortert synkende), hver eneste gang
- I gjennomsnitt (tilfeldig permutasjon), $H_n \approx \log(n) + 0.577$ ganger
- Totalt $3*(\log(n) + 0.577)$

Algoritmeanalyse

```
public static int maks(int[] a) // versjon 2 av maks-metoden
{
    int m = 0; // indeks til største verdi
    int maksverdi = a[0]; // største verdi

    for (int i = 1; i < a.length; i++) if (a[i] > maksverdi)
    {
        maksverdi = a[i]; // største verdi oppdateres
        m = i; // indeks til største verdi oppdateres
    }
    return m; // returnerer indeks/posisjonen til største verdi
} // maks
```

- Konstant tid – utføres en gang
- En operasjon

Algorithmeanalyse

```
public static int maks(int[] a) // versjon 2 av maks-metoden
{
    int m = 0; // indeks til største verdi
    int maksverdi = a[0]; // største verdi

    for (int i = 1; i < a.length; i++) if (a[i] > maksverdi)
    {
        maksverdi = a[i]; // største verdi oppdateres
        m = i; // indeks til største verdi oppdateres
    }

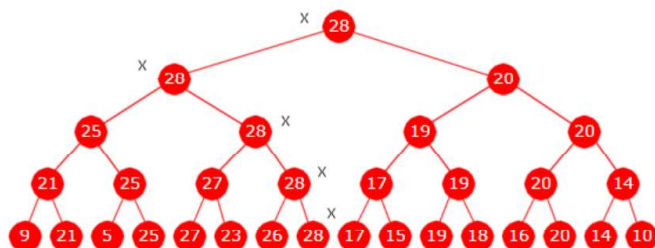
    return m; // returnerer indeks/posisjonen til største verdi
} // maks
```

- Total algoritme:
- $3 + 2 \cdot n - 1 + 2 \cdot n + 3 \cdot (\log(n) + 0.577) + 1 = 4 \cdot n + 3 \cdot \log(n) + 3.557$
- I "stor O" notasjon kaller vi det en $O(n)$ algoritme

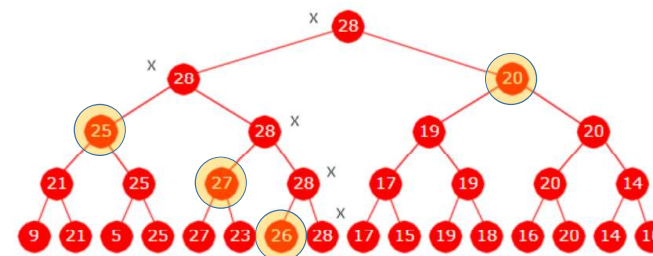
Repetisjon av algoritmeanalyse

Funksjonstype	n = 1	n = 10	n = 100	n = 1000	Beskrivelse
$f_1(n) = 1$	1	1	1	1	konstant
$f_2(n) = \log_2 n$	0	3,3	6,6	9,97	logaritmisk
$f_3(n) = \sqrt{n}$	1	3,2	10	31,6	kvadratrott
$f_4(n) = n$	1	10	100	1000	lineær
$f_5(n) = n \log_2 n$	0	33,2	664,4	9.965,8	lineæritmisk
$f_6(n) = n^2$	1	100	10.000	1.000.000	kvadratisk
$f_7(n) = n^3$	1	1.000	1.000.000	10 siffer	kubisk
$f_8(n) = 2^n$	2	1.024	31 siffer	302 siffer	eksponensiell
$f_9(n) = n!$	1	3.628.800	158 siffer	2568 siffer	faktoriell

Binære træer – Turneringer!



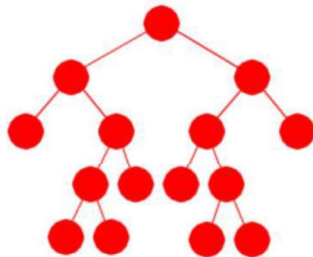
Nest største tall



- Ett av tallene 28 har vunnet over

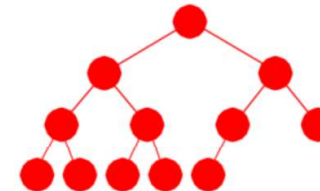
Fullt tre

- Hver node har to eller ingen barn



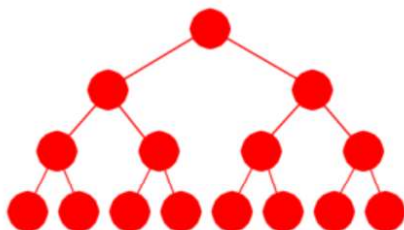
Komplett tre

- Alt unntatt siste nivå er fullt
- Siste nivå er fylt inn fra venstre



Perfekt binært tre

- Alt til og med siste nivå er fullt



Bubble sort

- Gå gjennom hver posisjon i tabellen og «boble» oppover
- For hvert tallpar, bytt så største kommer til høyre
- $n*(n-1)/4$ operasjoner (antall inversjoner!)

```
public static void boblesortering(int[] a)    // hører til klassen Tabell
{
    for (int n = a.length; n > 1; n--)        // n reduseres med 1 hver gang
    {
        for (int i = 1; i < n; i++)            // går fra 1 til n
        {
            if (a[i - 1] > a[i]) bytt(a, i - 1, i); // sammenligner/bytter
        }
    }
}
```

Programkode 1.3.3 e)

4: [4, 2], 3, 1]
4: [2, 4, 3], 1]
4: [2, 3, 4, 1]
4: [2, 3, 1, 4]

3: [2, 3], 1, 4]
3: [2, 3, 1], 4]
3: [2, 1, 3, 4]

2: [2, 1], 3, 4]
2: [1, 2, 3, 4]

Utvalgssortering – selection sort

- Finn minste tall og bytt med posisjon 1
- Repeter med tabell størrelse $n-1$
- $n*(n-1)/2$ operasjoner

[6, 7, **1** 4, 8, 9, 2, 5, 3, 10]
 [1, 7, 6, 4, 8, 9, **2** 5, 3, 10]
 [1, 2, 6, 4, 8, 9, 7, 5, **3** 10]
 [1, 2, 3, **4** 8, 9, 7, 5, 6, 10]
 [1, 2, 3, 4, 8, 9, 7, **5** 6, 10]
 [1, 2, 3, 4, 5, 9, 7, 8, **6** 10]
 [1, 2, 3, 4, 5, 6, **7** 8, 9, 10]
 [1, 2, 3, 4, 5, 6, 7, **8** 9, 10]
 [1, 2, 3, 4, 5, 6, 7, 8, **9** 10]
 [1, 2, 3, 4, 5, 6, 7, 8, 9, **10**]

```
public static void utvalgssortering(int[] a)
{
    for (int i = 0; i < a.length - 1; i++)
        bytt(a, i, min(a, i, a.length)); // to hjelpemetoder
}
```

Quick sort (kvikksortering)

- Bygger på konseptet partisjonering med en «pivot»
- Sorter tabellen slik at alle tall mindre enn pivot ligger til venstre og alle tall større ligger til høyre
- Eksempel: Pivot = 10

Insertion sort (innsettingssortering)

- Ta en verdi ut av tabellen på plass k
Krav: alt før plass k er sortert

3	5	6	10	10	11	13	14	16	20	12	4	7	2	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- Bruk ordnet innsetting i intervallet $[0, k-1]$

12	3	5	6	10	10	11	13	14	16	20		4	7	2	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- $n(n + 3)/4 - H_n$ operasjoner i gjennomsnitt

	3	5	6	10	10	11	12	13	14	16	20	4	7	2	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

4	3	5	6	10	10	11	12	13	14	16	20		7	2	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

	3	4	5	6	10	10	11	12	13	14	16	20	7	2	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Merge sort

- Enkel idé:
- Gitt to lister med sorterte tall, velg det minste fra de to listene til enhver tid

[illegible]

Kvadratrotsøk

- Som lineært søk (usortert søk), men øk i med kvadratroten av tabellens lengde istedenfor i
- for (int i=0; i<a.length; ++i)
- for (int i=0; i<a.length; i+=sqrt(a.length)) {
if (a[i] > verdi) { ... }

2	5	6	9	10	12	15	17	19	19	19	19	22	23	25
1	2		4	5		7	8	9	10	11	12	13	14	15

Binærsøk

- Søk etter 30:
 - Er a[m] lik 20?
Søket er ferdig!
 - Er 30 større enn midt
Søk i intervallet [m+1, h]
 - Ellers
Søk i intervallet [l, m]

3	8	10	14	14	16	21	24	27	30	32	33	34	37	40
v								m						h

3	8	10	14	14	16	21	24	27	30	32	33	34	37	40
								v		m				h

3	8	10	14	14	16	21	24	27	30	32	33	34	37	40
								v	m	h				

Flerdimensjonelle tabeller

- To dimensjoner

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

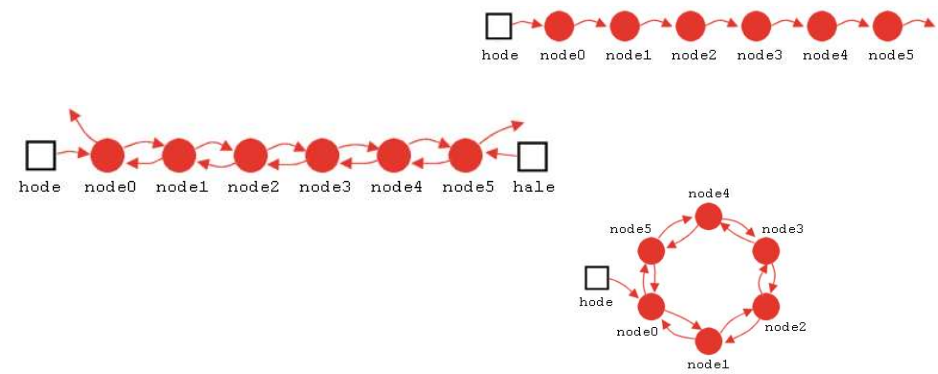
```
int[][] a = {{1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15}};
```

- a[j][i] gir oss rad j, kolonne i
- Eksempel: a[3][2] = 12

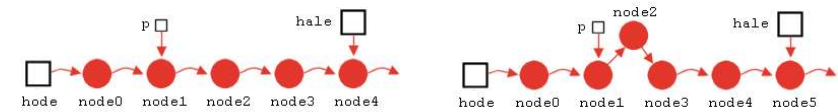
Lineære datastrukturer

- Kapittel 3: Lineære datastrukturer
 - 3.1 En beholder 1
 - 3.2 Tabellbasert liste 1, 2, 3, 4, 5, 6, 7
 - 3.3 Lenket liste 1, 2, 3, 4, 5, 6
- Kapittel 4: Stakker og køer
 - 4.1 En stakk 1, 2, 3, 4
 - 4.2 En kø 1, 2, 4, 5
 - 4.3 Toveiskø 1, 2, 3, 4
 - 4.4 Prioritetskø 1, 2, 3, 4, 5
- Kapittel 6: Hashing og hashingteknikker
 - 6.1 Hashing 1, 3, 4, 7

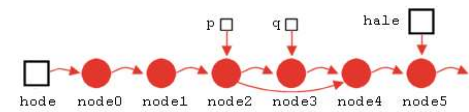
Lenket liste



Legge inn på gitt posisjon i listen

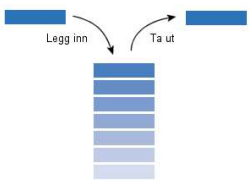


Fjerne fra listen

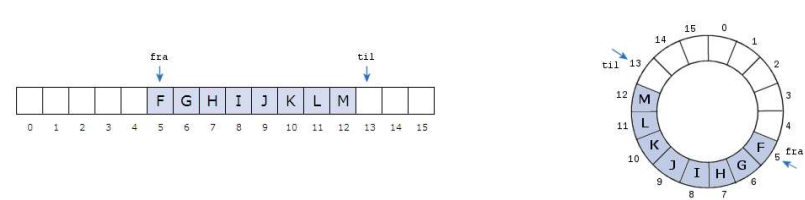


Stack

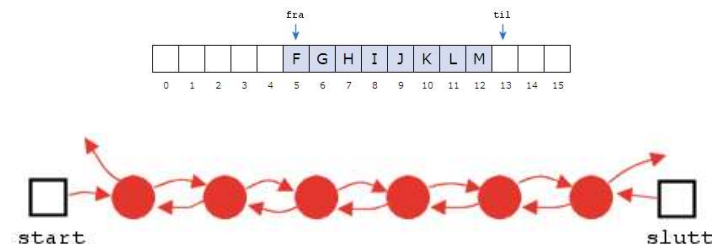
- Stabbel med tallerkner



Circular Queue



Deque – double ended queue

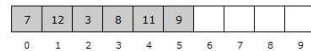


Unsorted Priority Queue

- Legg inn 9

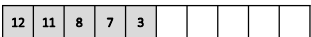


- Finn og fjern høyest prioritet (3)

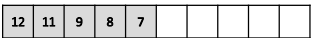
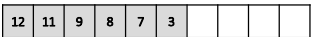


Sorted Priority Queue

- Legg inn 9 på riktig sortert plass



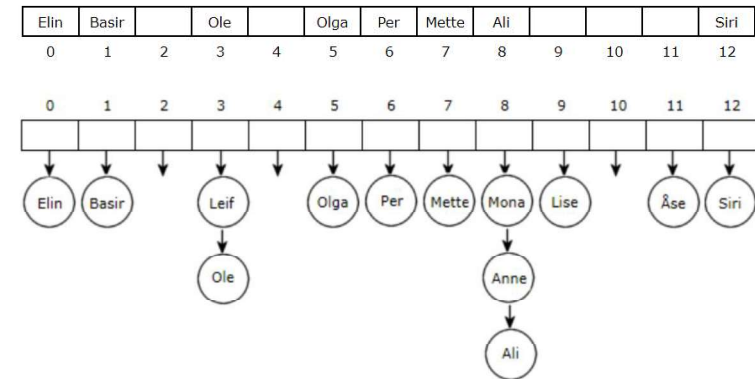
- Fjern høyest prioritet (3)



Hash tabell – Lukket adressering

UsortertTabellPrioritetsKø		
Metode	Operasjoner	Orden
leggInn	Et nytt element legges alltid bakerst.	<i>konstant</i>
taUt	Hvis det er n elementer, trengs $n - 1$ sammenligninger for å finne den minste.	<i>lineær</i>

SortertTabellPrioritetsKø		
Metode	Operasjoner	Orden
leggInn	Det må letes etter rett sortert plass og de til høyre må forskyves.	<i>lineær</i>
taUt	Den minste ligger alltid bakerst.	<i>konstant</i>



Hashtabell – Åpen adressering

- Hashverdi til «Bodil» er 5 => men vi søker linært til vi finner en ledig plass (9)
- Kan også søke med «kvadratisk søk», dvs 5, 5+1, 5+4, 5+9, ...

Elin	Basir		Ole		Olga	Per	Mette	Ali				Siri
0	1	2	3	4	5	6	7	8	9	10	11	12

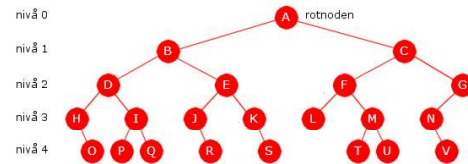
Elin	Basir		Ole		Olga	Per	Mette	Ali	Bodil			Siri
0	1	2	3	4	5	6	7	8	9	10	11	12

Binære trær og grafer

- Kapittel 5: Binære trær
 - 5.1 Generelle binære trær 1, 3, 4, 5, 6, 7, 10, 11, 12, 14
 - 5.2 Binære søketrær 1, 2, 3, 5, 6, 7, 8, 9, 14
 - 5.3 Minimums og makstrær 1, 2, 3, 4, 6
 - 5.4 Huffmantrær 1, 2, 3, 4, 5, 6, 7
- Kapittel 9: Balanserte binærtrær
 - 9.2 Rød-svart og 2-3-4 trær 1, 2, 4, 5
- Kapittel 11: Grafer
 - 11.1 Datastrukturer for grafer 6
 - 11.2 Korteste vei i graf 1

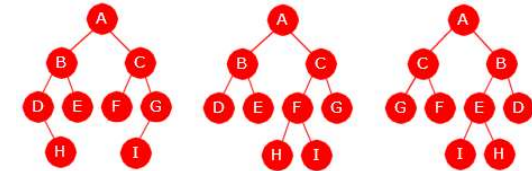
Binærtre

- Level – nivå
- Slektskap
- Rotnode
- Subtrær, gren
- Bladnoder, indre noder
- Vei / path i treet
- Avstand – lengde av vei
- Høyde av tre – lengde av lengste vei
- Dybde av node
- «Ned i treet» => bort fra rotnoden



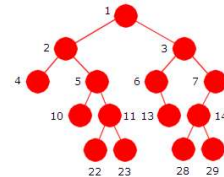
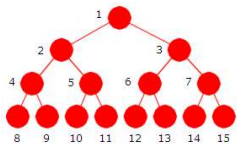
Forskjellige binærtrær

- Catalan-tallet $C(n)$
 - 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862

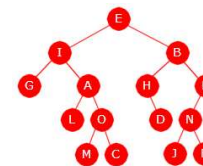


$$(**) \quad C(n) = \frac{1}{n+1} \binom{2n}{n}$$

Nummerering av noder



Traversering – bredde først – breadth first



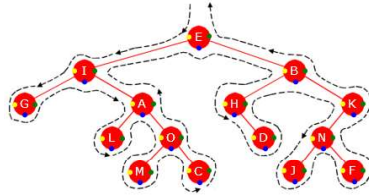
Runde	Ut av køen	Inn i køen	Køens innhold
1	E	I, B	I, B
2	I	G, A	B, G, A
3	B	H, K	G, A, H, K
4	G		A, H, K
5	A	L, O	H, K, L, O
6	H	D	K, L, O, D

E, I, B, G, A, H, K, L, O, D, N, M, C, J, F

- Implementeres med kø

Traversering – dybde først – depth first

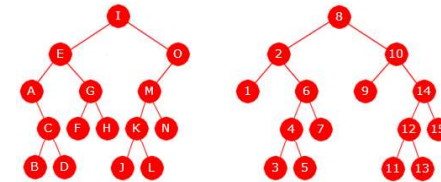
- Preorder (venstre, gul),
 - E, I, G, A, L, O, M, C, B, H, D, K, N, J, F
- Inorder (bunnen, blå),
 - G, I, L, A, M, O, C, E, H, D, B, J, N, F, K
- Postorder (høyre, grønn)
 - G, L, M, C, O, A, I, D, H, J, F, N, K, B, E



- Implementeres med stack eller rekursjon

Binære søketrær

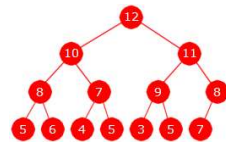
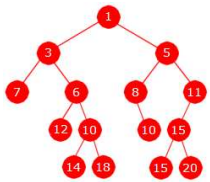
- Stigende inorder sortering



- Alle venstre subtrær er **mindre enn** foreldrenode
- Alle høyre subtrær er **større eller lik** foreldrenode

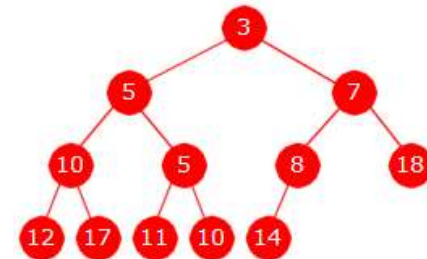
Minimumstrær

- Sortert stigende/synkende etter nivå



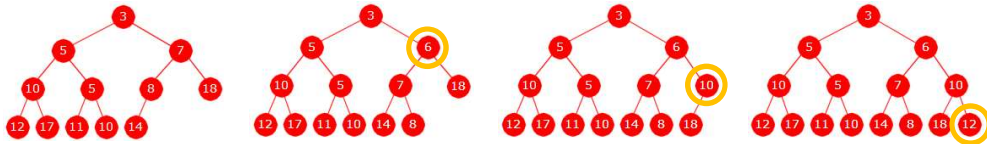
Minheap - minimumsheap

- Komplette binærtre
- Minimumstre



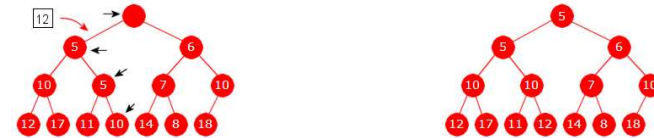
Legge inn noder

- Legg inn i bunnen av treet
- Bytt med forelder så lenge forelder er større



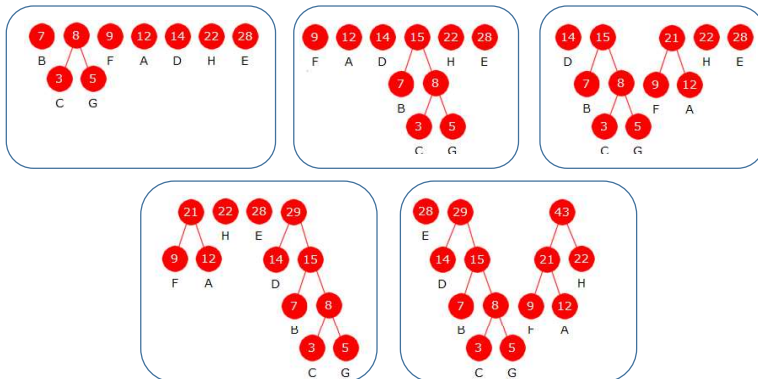
Fjerne noder

- Ta ut rotnoden
- Sift down – bytt med minste verdi nedover i treet

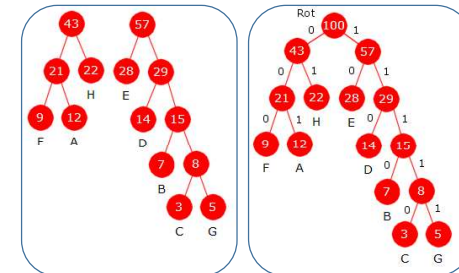


Bygge Huffmantrær

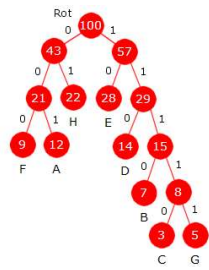
- Velg noder med minst verdi og slå sammen
- Repeter til hele treet er fullt



Bygge Huffmantrær



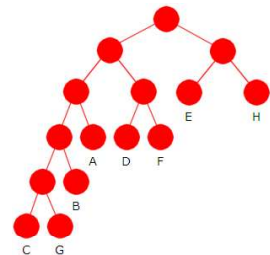
Bygge Huffmantrær



Tegn	A	B	C	D	E	F	G	H
Bitkode	001	1110	11110	110	10	000	11111	01
Bitkodelengde	3	4	5	3	2	3	5	2

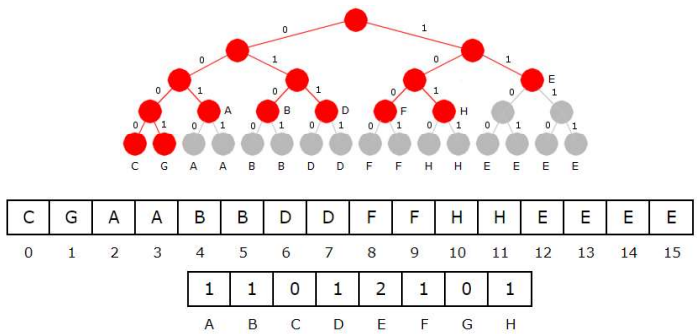
Venstreorientert kanonisk tre

- Fylt ut fra venstre
- Noder på samme nivå er sortert

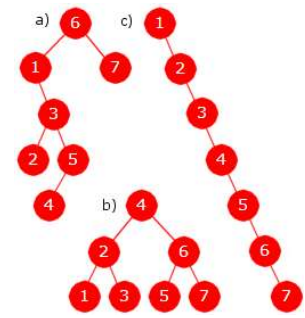


Tegn	A	B	C	D	E	F	G	H
Bitkode	001	0001	00000	010	10	011	00001	11
Bitkodelengde	3	4	5	3	2	3	5	2

Komplette huffmantrær – effektiv dekomprimering

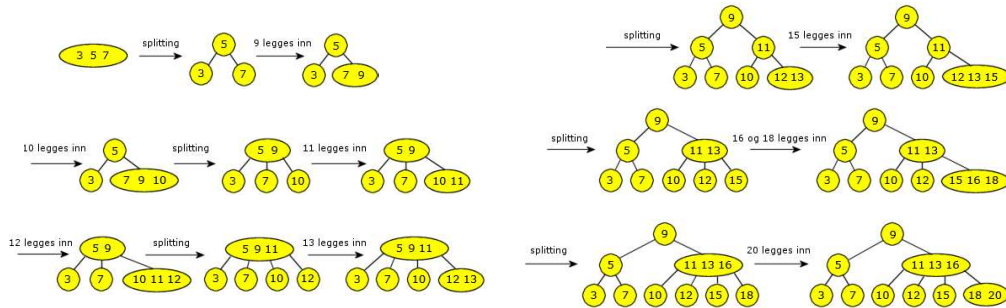


Forskjellige ordninger – balanserte trær



2-3-4 B-trær

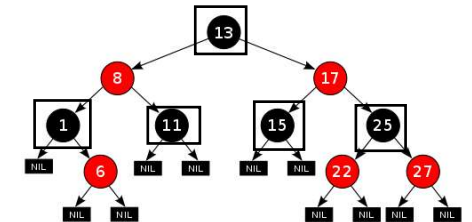
- 3, 5, 7, 9, 10, 11, 12, 13, 15, 16, 18, 20



19.11.2021

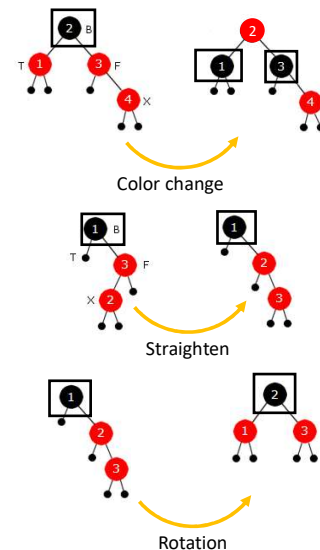
Rød-sorter trær

- Rotnoden skal være sort
- Alle grener skal passere like mange sorte noder
- En rød node har kun sorte barn
- Hver nye node man setter inn er rød



Scenarier

- If Parent == 0 => rotnode
- Else if Parent == Black
vanlig insert
- Else if Uncle == RED
Color change!
Then check grandparent
- Else
if triangle: Straighten
Then rotation



Dijkstras algoritme

