DMA ugeseddel 6

Litteratur

- CLRS kapitel 12 dog ikke afsnit 12.4.
- CLRS kapitel 8. Dog ikke analysen af bucket sort startende på side 202.
- CLRS C.1 om permutationer. Kun som støtte til afsnit 8.1.

Mål for ugen

- At kende til binære søgetræer.
- At kende til sorteringsmetoder i lineær tid.
- At forstå at sortering i visse tilfælde kræver mere end lineær tid, og i andre tilfælde kan løses i lineær tid.

Plan for ugen

- Mandag: Binære søgetræer, CLRS kapitel 12.
- Tirsdag: Sortering i lineær tid, CLRS kapitel 8, afsnit 8.2 til 8.4.
- Fredag: Nedre grænse for sammenligningsbaseret sortering, CLRS afsnit 8.1. Opsamling og afrunding.

Opgaver

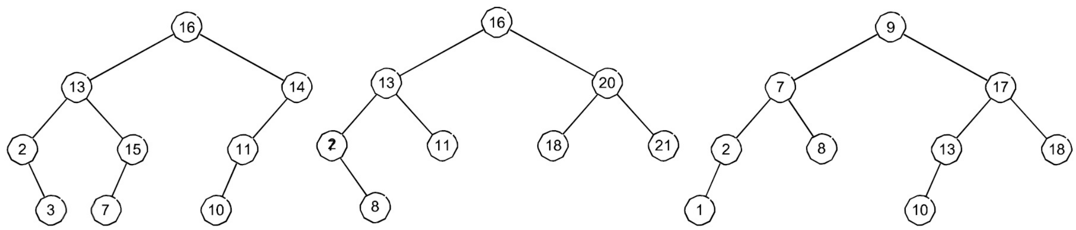
Pas på: Opgaver markeret med * er svære, ** er meget svære, og * har du ikke en chance for at løse.**

Opgaver markeret med (ekstra) eller stjerner kan gemmes til resten af opgaverne er regnet. I bunden af ugesedlen finder du nogle ekstra opgaver hvis du er færdig med dagens opgaver.

Opgaver til mandag

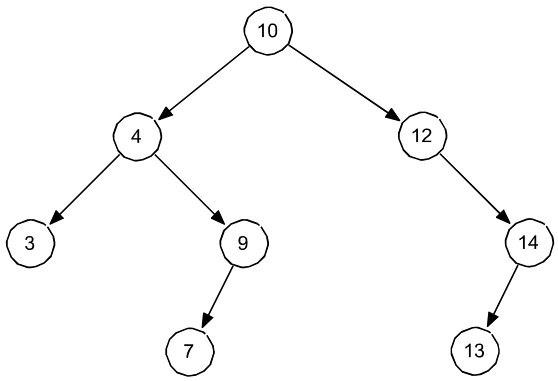
1. **Opvarmning.**

   (a) Hvilket af følgende træer er et binært søgetræ?

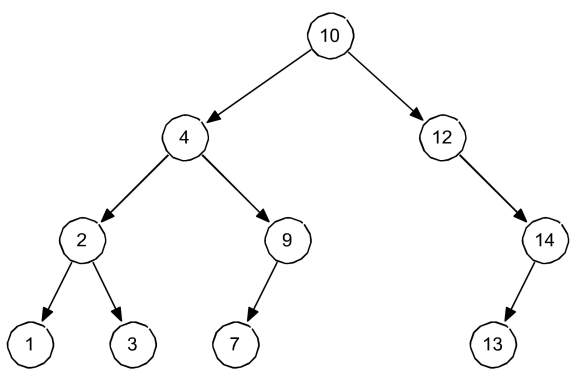


(a) (b) (c)

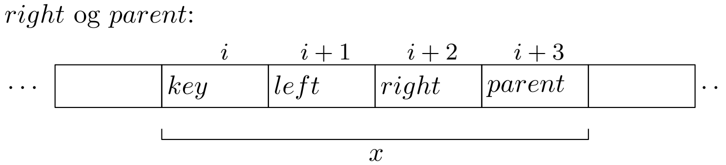
(b) Angiv hvordan følgende binære søgetræ nedenfor ser ud efter indsættelse af elementet med nøglen 8.



(c) Angiv hvordan følgende binære søgetræ nedenfor ser ud efter sletning af elementet med nøglen 4.

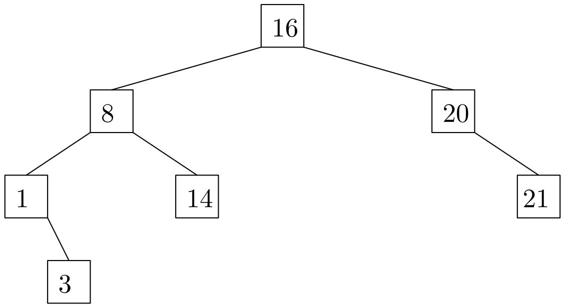


2. **Repræsentation af binære søgetræer.** Vi repræsenterer hver knude x i et binært søgetræ vha. fire på hinanden følgende felter i hukommelsen, hvor vi gemmer nøgleværdien og dernæst pointerne $left$, $right$ og $parent$:



Hver pointer indeholder indekset på det felt hvor knudens nøgleværdi er gemt.

(a) Nedenfor ses et binært søgetræ T og det er vist hvor nøgleværdierne er gemt i hukommelsen. Skriv værdierne for pointerne ind.



0	1	2	3	4	5					10					15					20				
			20				21						16										3	

25					30					35					40					45				
		1										8				14								

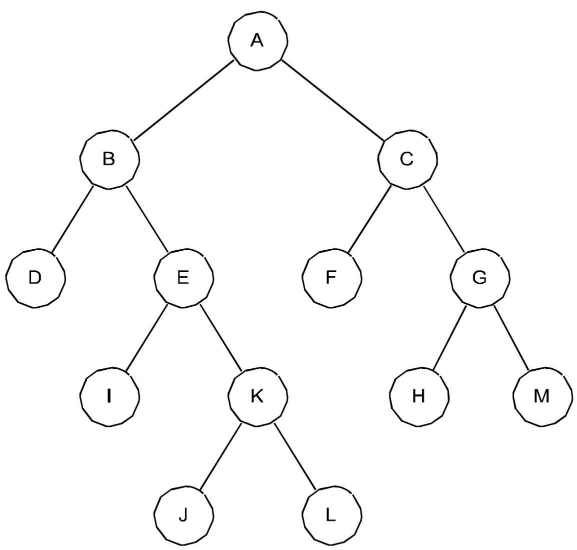
(b) Vi ønsker at indsætte en ny knude med nøgleværdi 12. Tegn det opdaterede søgetræ, vælg et sted at gemme knuden i hukommelsen, og skriv de opdaterede pointere ind.

0	1	2	3	4	5					10					15					20				
			20				21						16										3	

25					30					35					40					45				
		1										8				14								

3. **Længste og korteste stier.**

(a) Betragt nedenstående træ.



Hvad er længden af den korteste hhv. længste rod-til-blad sti i træet?

(b) Giv en rekursiv algoritme $KortesteSti(x)$ der givet rodknuden x til et binært træ returner længden af den korteste rod-til-blad sti i træet. Angiv gerne algoritmen i pseudokode. Angiv køretiden af din algoritme i asymptotisk notation og argumentér for at algoritmen er korrekt.

4. **Blade og højde.** Lad T være et binært træ med n knuder og rod v.

(a) Giv en rekursiv algoritme, der givet v beregner antallet af blade i T. Skriv pseudokode for din løsning.

(b) Giv en rekursiv algoritme, der givet v beregner højden af T. Skriv pseudokode for din løsning.

5. **Gennemløb af binære søgetræer.** Giv en algoritme, der givet et binært træ T med en nøgle i hver knude, afgør om T overholder søgetræsinvarianten.

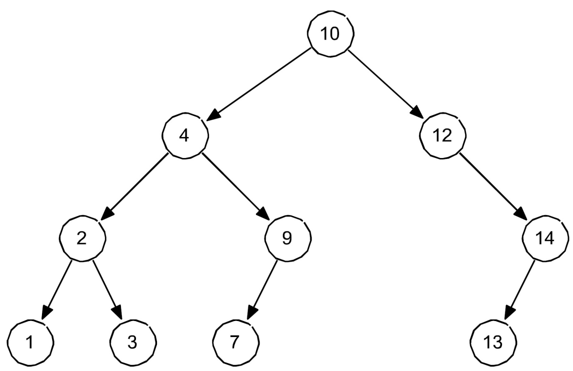
6. **Perfekt balancerede binære søgetræer.**

   (a) Tegn et binært søgetræ med nøglerne $(1, 2, \ldots, 7)$ således at træet har højde 2. Angiv en rækkefølge af tallene således at Tree-Insert algoritmen fra CLRS vil konstruere dit træ,hvis udført på tallene i den rækkefølge.

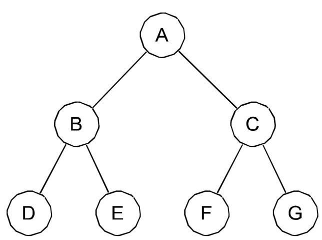
   (b) Lad A være en sorteret tabel af $n = 2^{h+1} - 1$ forskellige tal (et perfekt balanceret træ med højde h har præcist $2^{h+1} - 1$ knuder). Giv en sekvens af indsættelser af tallene i A i et binært søgetræ T således at T bliver et komplet binært træ af højde h.

7. **Trægennemløb.**

   (a) Angiv rækkefølge af knuderne som de bliver skrevet ud ved preorder gennemløb af træet nedenfor.



   (b) Betragt nedenstående træ.



   Hvilke af følgende sekvenser af bogstaver bliver udskrevet ved et preorder, inorder og postorder gennemløb af ovenstående træ?

   1: A B C D E F G 2: G F E D C B A 3: A B D E C F G
   4: D B E A F C G 5: D E B F G C A 6: C B D A F E G

   (c) Antag at der til hver knude i et binært træ er der knyttet en farve; knuden x har farven $farve[x]$. Betragt følgende algoritme:

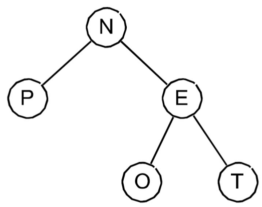
Algorithm 1: $Udskriv(x)$

---

if $(x \neq nil)$

print $farve[x]$
$Udskriv(left[x])$
$Udskriv(right[x])$

Lad x være rodknuden for træet nedenfor. Et kald til algoritmen $Udskriv(x)$ vil da udskrive farvesekvensen "NPEOT".



Du skal ændre algoritmen $Udskriv$, således at det rekursive gennemløb ved kaldet til $Udskriv(x)$ for rodknuden x udskriver farvesekvensen "NETOP" i stedet. Argumentér for at din algoritme er korrekt.

8. **Mere rekursion på træer.** Denne opgave handler om rekursion og rodfæstede binære træer. Hver knude har enten to eller ingen børn. Hvis knuden x ikke har nogle børn, har $left[x]$ og $right[x]$ den specielle værdi *nil*. Hvis knude x ikke har nogle børn, kaldes den et *blad*. Ellers kaldes den en *intern knude*. Såfremt rodknuden for et træ er *nil*, er træet tomt. Til hver knude i træet er der knyttet en farve; knuden x har farven $farve[x]$. Med den definition af binære træer, vi benytter i denne opgave, kan vi beregne antallet af knuder og antallet af blade i et træ ved hjælp af følgende to algoritmer:

Algorithm 2: $Nodes(x)$

---

if ($x == nil$)
  return 0
else
  return $1 + Nodes(left[x]) + Nodes(right[x])$

Algorithm 3: $Leafs(x)$

---

if ($x == nil$)
  return 0
else
  if ($left[x] == nil$)
    return 1
  else
    return $Leafs(left[x]) + Leafs(right[x])$

(a) En *indre* knude i et træ er en knude, som ikke er et blad. Konstruér en algoritme $Intern(x)$, der givet en rodknude x for et træ returnerer antallet af indre knuder i træet. Angiv tidskompleksiteten/køretiden af din løsning i O-notation.

(b) Et træ har en *blå rodvej*, hvis der er en vej i træet fra roden x til et blad, hvor alle knuderne på vejen fra x til bladet har farven blå. Konstruér en rekursiv algoritme $BluePath(x)$, der returnerer tallet 1, hvis træet med roden x har en blå rodvej. Hvis en sådan vej ikke eksisterer, skal algoritmen returnere 0. Argumentér for at din algoritmen er korrekt. Angiv tidskompleksiteten/køretiden af din løsning i O-notation. Begrund dit svar.

9. **CLRS-opgaver mv.**

   (a) Hvor findes element med hhv. mindste og største nøgle i et binært søgetræ?

   (b) CLRS 12.1-1.

   (c) CLRS 12.2-5. *Hint:* Argumentér vha. modstrid, først for at successor/predecessor er i højre/venstre deltræ, og dernæst for opgavens påstand.

   (d) CLRS 12.2-6. *Hint:* Argumentér vha. modstrid, først for at successor må være en ancestor og dernæst for opgavens påstand.

   (e) CLRS 12.2-7. *Hint:* Find en grænse for hvor mange gange den foreslåede algoritme besøger en given knude.

   (f) CLRS 12.3-3.

Opgaver til tirsdag

Lav først mindst én opgave om hver af counting sort, radix sort og bucket sort.

1. **Counting sort**

   (a) CLRS 8.2-1

   (b) CLRS 8.2-2

   (c) CLRS 8.2-3

   (d) CLRS 8.2-4 *Hint:* Brug tabellen C fra counting sort.

2. **Radix sort**

   (a) CLRS 8.3-1

   (b) CLRS 8.3-4

3. **Bucket sort**

   (a) CLRS 8.4-1

   (b) CLRS 8.4-2

4. **Sortering i forskellige situationer**

   (a) Antag at n personer skal sorteres efter deres højde hvor højden er angivet i antal hele cm. Hvilken algoritme vil du bruge og hvad er køretiden?

   (b) Antag at n heltal fra mængden $\{1, \ldots, n^{10}\}$ skal sorteres. Hvilken algoritme vil du bruge og hvad er køretiden?

(c) Antag du skal sortere n objekter. Du kan ikke sammenligne objekterne direkte. Men du kan spørge et orakel A hvilket af to objekter er mindst. For hvert spørgsmål bliver du opkrævet $\Theta(\log\log n)$ guldmønter. Hvilken algoritme vil du bruge og hvor mange guldmønter skal du aflevere til oraklet A?

5. **In place-sortering i lineær tid.** Lav CLRS problem 8-2, delopgave a, b og c.

6. Lav opgaver om binære søgetræer som du ikke nåede mandag.

Opgaver til fredag

1. **Overflødige sammenligninger.** I denne opgave betragter vi sammenligningsbaserede sorteringsalgoritmer og antager at alle tallene som skal sorteres er forskellige. En sammenligning mellem to tal x og y kaldes *overflødig* hvis det på baggrund af tidligere sammenligninger vides om $x < y$ eller $x > y$. F.eks. er sammenligningen overflødig hvis de to tal x og y allerede er blevet sammenlignet med hinanden tidligere, men den er også overflødig hvis det vides at $x < z$ og $z < y$, eller mere generelt, hvis det vides at
$$x < z_1,\ z_1 < z_2,\ \ldots,\ z_{k-1} < z_k,\ z_k < y.$$

   (a) Laver Insertion-Sort nogensinde en overflødig sammenligning?

   (b) Hvad med Merge-Sort?

   (c) Hvad med Heap-Sort?

2. **Beslutningstræer.** Som beskrevet CLRS afsnit 8.1, svarer enhver sammenligningssortering af en mængde M af størrelse n til et *beslutningstræ*, hvilket er et binært træ med mindst $n!$ blade. Hver knude svarer til en sammenligning $x : y$, hvor x og y er to elementer i M, og det venstre barn svarer til $x < y$, mens det højre svarer til $x > y$. Hvert blad b svarer til den sorterede rækkefølge af elementerne i M, og denne rækkefølge kan fastlægges ud fra resultaterne af de sammenligninger, der er på stien fra roden ned til b i beslutningstræet. I denne opgaver antager vi for simpelhedens skyld at alle tal er forskellige i de mængder vi sorterer.

   (a) Tegn beslutningstræet der svarer til at køre Merge-Sort på en tabel $T = [a, b, c, d]$ med 4 tal. Hvor mange sammenligninger bliver der højst lavet?

   (b) (Ekstra) Tegn beslutningstræet der svarer til at køre Heap-Sort på en tabel $T = [a, b, c, d]$ med 4 tal. På baggrund af denne og den foregående delopgave, hvilken algoritme virker mest effektiv: Merge-Sort eller Heap-Sort?

   (c) CLRS 8.1-1.

   (d) CLRS 8.1-3.

3. **Langsomme sorteringsalgoritmer.**

   (a) Nedenfor ses pseudokoden til den meget langsomme sorteringsalgoritme `SlowSort`. Bevis vha. stærk induktion over n at hvis `A` er et 0-indekseret array af længde n, så vil kaldet `SlowSort(A, 0, n-1)` føre til at `A` bliver sorteret.

```
SlowSort(A, i, j)
  if i < j
    m = floor((i+j)/2)
    SlowSort(A, i, m)
    SlowSort(A, m+1, j)
    if A[m] > A[j]
      swap A[m] and A[j]
    SlowSort(A, i, j-1)
```

(b) Argumentér for at køretiden $T(n)$ af `SlowSort` opfylder

$$T(n) = \begin{cases} c_1, & n = 1 \\ 2 \cdot T(n/2) + T(n-1) + c_2, & n > 1, \end{cases}$$

for konstanter $c_1, c_2 > 0$.

(c) Nedenfor ses pseudokoden til den knapt så langsomme sorteringsalgoritme `StoogeSort`. Argumentér for at køretiden $T(n)$ af `StoogeSort` opfylder

$$T(n) = \begin{cases} c_1, & n \leq 2 \\ 3 \cdot T(2n/3) + c_2, & n > 2, \end{cases}$$

for konstanter $c_1, c_2 > 0$.

```
StoogeSort(A, i, j)
  if A[i] > A[j]
    swap A[i] and A[j]
  if j >= i + 2
    t = floor((j-i+1)/3)
    StoogeSort(A, i, j-t)     // Sort first 2/3 of the array
    StoogeSort(A, j-2*t+1, j) // Sort last  2/3 of the array
    StoogeSort(A, i, j-t)     // Sort first 2/3 of the array
```

(d) (*) Bevis vha. stærk induktion over n at hvis `A` er et 0-indekseret array af længde n, så vil kaldet `StoogeSort(A, 0, n-1)` føre til at `A` bliver sorteret. *Hint:* I induktionsskridtet kan du lave induktionsantagelsen at algoritmen virker når $j - i + 1 < n$. Start da med at antage at 3 går op i n.

(e) (*) Bevis at køretiden af `StoogeSort` er $O(n^3)$ ved at bruge rekursionen fra delopgave (c).

(f) (*) Brug Master Theorem, CLRS Theorem 4.1, side 94, til at bevise at køretiden af `StoogeSort` er $\Theta(n^{\log_{3/2} 3})$. Her er $\log_{3/2} 3 \approx 2.71$.

4. **Sammenligninger ved sortering af 5 tal.**

   (a) Hvor mange sammenligninger bruger MERGE-SORT højst til at sortere et array $T = [a, b, c, d, e]$ med 5 forskellige tal?

   (b) (*) Vis den sorterede rækkefølge af 5 forskellige tal $\{a, b, c, d, e\}$ kan bestemmes vha. højst 7 sammenligninger.

Ekstraopgaver

1. **(**) Dynamiske træer.** Lad der være givet en funktion `Fjern(F,e)` som fjerner en kant e fra et træ i en skov F. Lad e forbinde knuderne x og y i et træ T i skoven F. Efter at e er fjernet, er træet T blevet til to mindre træer T_x og T_y, hvor x er en knude i T_x og y er en knude i T_y. Antag at `Fjern` har en køretid som er lineær i antallet af knuder i det mindste af træerne T_x og T_y.

   (a) Lad F initielt have n knuder (og dermed højst $n-1$ kanter). Vis at de højst $n-1$ `Fjern`-operationer samlet tager $O(n \log n)$ tid.

   (b) Giv en implementering af `Fjern(F,e)` med den angivne køretid.

2. **CLRS-opgaver.**

(a) CLRS 12.1-5.
(b) CLRS 12.2-1.
(c) CLRS 12.2-2.
(d) CLRS 12.2-3.
(e) CLRS 12.3-1.
(f) CLRS 12.3-2.
(g) CLRS 12.3-6.