DMA ugeseddel 6

Litteratur

- CLRS kapitel 12.
- CLRS kapitel 8. Dog ikke analysen af bucket sort startende på side 217.
- CLRS C.1 om permutationer, side 1179–1180. 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

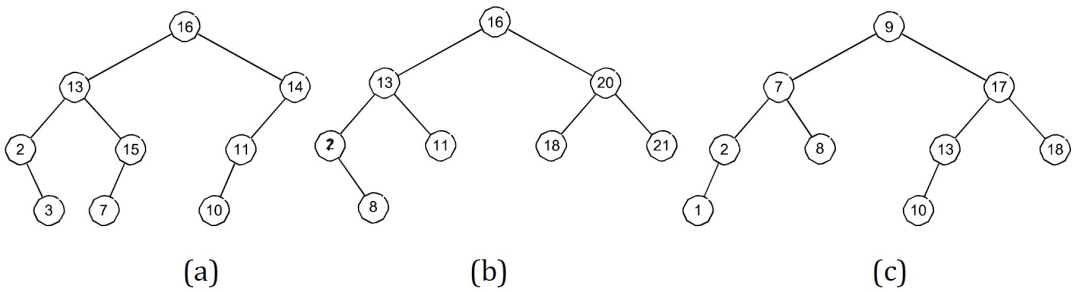
Pas på: Opgaver markeret med * er svære, ** er meget svære, og * har du ikke en chance for at løse.**

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.

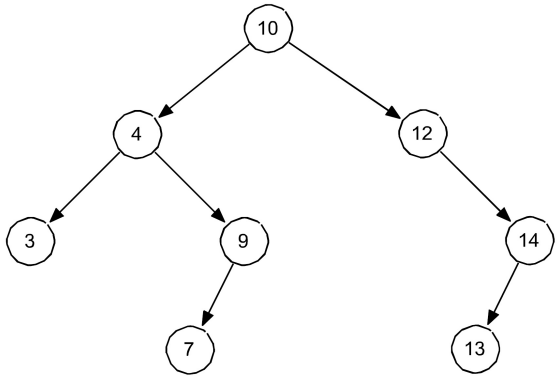
Opgaver til mandag

1. **Opvarmning.**

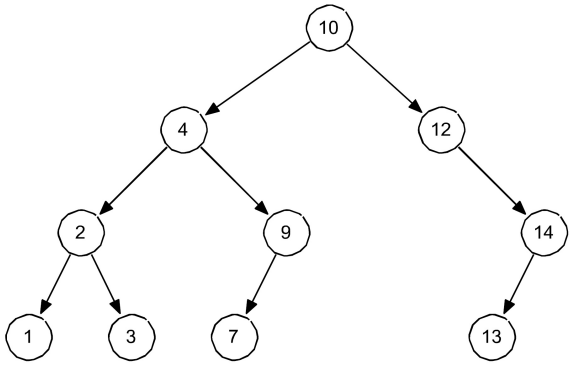
   (a) Hvilket af følgende træer er et binært søgetræ?



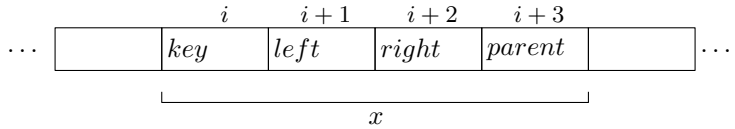
(b) Angiv hvordan følgende binære søgetræ nedenfor ser ud efter indsættelse af elementet med nøglen 8.



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

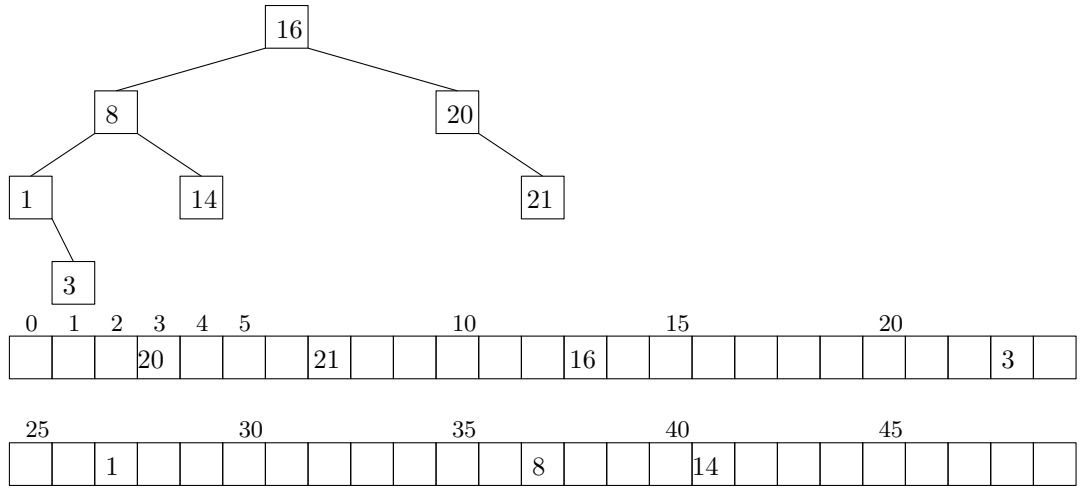


2. **Repræsentation af binære søgetræer.** Vi repræsenterer hver knude x i et binært søgetræ vha. fire på hinanden følgende felter i hukommelsen, hvor vi gemmer nøgleværdien og dernæst pointerne $left$, $right$ og $parent$:

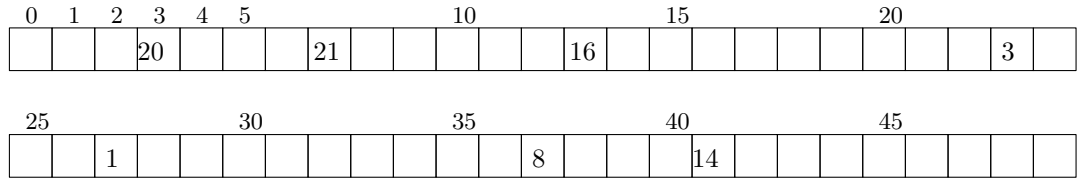


Hver pointer indeholder indekset på det felt hvor knudens nøgleværdi er gemt.

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

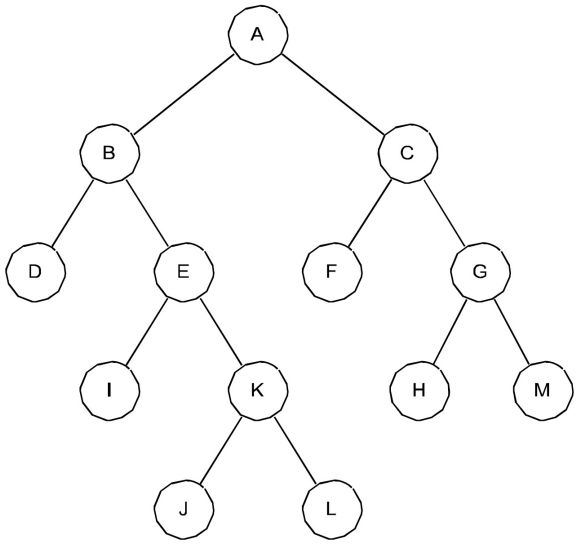


(b) Vi ønsker at indsætte en ny knude med nøgleværdi 12. Tegn det opdaterede søgetræ, vælg et sted at gemme knuden i hukommelsen, og skriv de opdaterede pointere ind.



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.

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, dog ikke den sidste del "Then rewrite ...," da det er uklart hvad der menes.

2. **Radix sort**

   (a) CLRS 8.3-1

   (b) CLRS 8.3-5

3. **Bucket sort**

   (a) CLRS 8.4-1

   (b) CLRS 8.4-2

4. **Sortering i forskellige situationer**

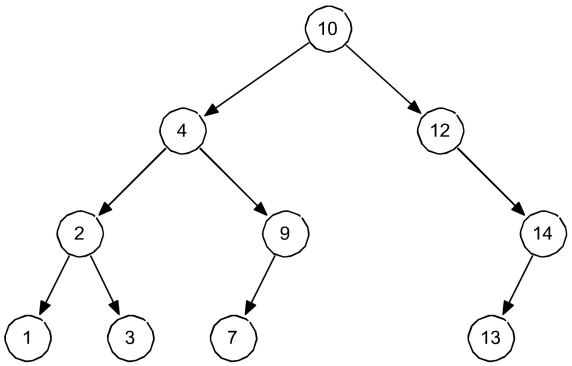
   (a) Antag at n personer skal sorteres efter deres højde hvor højden er angivet i antal hele cm. Hvilken algoritme vil du bruge og hvad er køretiden?

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

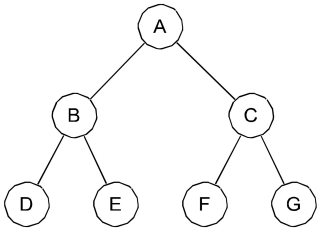
   (c) Antag du skal sortere n objekter. Du kan ikke sammenligne objekterne direkte. Men du kan spørge et orakel A hvilket af to objekter er mindst. For hvert spørgsmål bliver du opkrævet $\Theta(\log \log n)$ guldmønter. Hvilken algoritme vil du bruge og hvor mange guldmønter skal du aflevere til oraklet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

5. **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

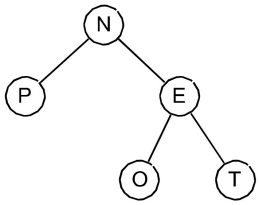
(c) Antag at der til hver knude i et binært træ er der knyttet en farve; knuden x har farven $x.farve$. Betragt følgende algoritme:

Algorithm 1: Udskriv(x)

---

if ($x \neq$ nil)
  print $x.farve$
  Udskriv($x.left$)
  Udskriv($x.right$)

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.

Opgaver til fredag

1. **Beslutningstræer.** Som beskrevet CLRS afsnit 8.1, svarer enhver sammenligningssortering af et array $M = [a_1, \ldots, a_n]$ af størrelse n til et *beslutningstræ*, hvilket er et binært træ med mindst $n!$ blade. Hver knude svarer til en sammenligning $i : i$, hvor a_i og a_j er to elementer i M, og det venstre barn svarer til $a_i \leq a_j$, mens det højre svarer til $a_i > a_j$. 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.

   (a) Tegn beslutningstræet der svarer til at køre Merge-Sort på et array $T = [a_1, a_2, a_3, a_4]$ med 4 tal, svarende til CLRS figur 8.1. Du kan nøjes med at tegne de knuder der kan nås hvis $a_1 \leq a_2$ og $a_3 \leq a_4$. Hvor mange sammenligninger bliver der højst lavet?

(b) (Ekstra) Tegn beslutningstræet der svarer til at køre HEAP-SORT på et array $T = [a_1, a_2, a_3]$ med 3 tal, svarende til CLRS figur 8.1.

(c) CLRS 8.1-1.

(d) (ekstra) CLRS 8.1-3.

2. **Langsomme sorteringsalgoritmer.**

(a) Nedenfor ses pseudokoden til den meget langsomme sorteringsalgoritme `SlowSort`. Se visualisering af `SlowSort`:

`https://www.youtube.com/watch?v=QbRoyhGdjnA`

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`. Se visualisering af `StoogeSort`:

`https://www.youtube.com/watch?v=vIDkfrSdID8`

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 $\log_{3/2} 3 \approx 2.71$.

3. **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.

4. **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?

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.