Tietorakenteet 2018
Harjoitukset 4, ratkaisut (Viikko 40)

1.-2. Katso tiedostot DynArrStack.java ja DynArrStackTest.java.

3. Mahdollisia laskutapoja on useita, esim.

   a) $1\ 3\ +\ 5\ +\ 7\ -$

   b) $6\ 3\ -\ 2\ *\ 1\ +$

   c) $3\ 4\ 3\ *\ 2\ +\ *$

   d) $3\ 4\ +\ 20\ 3\ 4\ *\ 2\ +\ -\ *$

4. Algoritmi 1. Kompleksisuus on $\mathcal{O}(n)$, koska while-silmukkaa suoritetaan $n-1$ kertaa, missä n on listan pituus.

Algorithm 1 Listan L toiseksi viimeisen alkion etsintä

```
SecondLast(L)
  if L.first = null or L.first.next = null then
    virhe: listassa L ei ole toiseksi viimeistä alkiota
  end if
  n ← L.first
  while n.next.next ≠ null do
    n ← n.next
  end while
  return n
```

5. Algoritmi 2. Kompleksisuus on $\mathcal{O}(n+m)$, missä n on listan A pituus ja m on listan B pituus. Rekursiivinen versio 3.

Algorithm 2 Listojen limittäminen (ei-rekursiivinen versio). Muuttuja h on tuloslistan R alku ja t on tuloslistan R loppu.

```
Merge(A, B)
  if A.head = null then
    h ← B.head
    B.head ← null
  else if B.head = null then
    h ← A.head
    A.head ← null
  else
    if A.head < B.head then
      h ← A.head
      A.head ← A.head.next
    else
      h ← B.head
      B.head ← B.head.next
    end if
    t ← h
    while A.head ≠ null and B.head ≠ null do
      if A.head < B.head then
        t.next ← A.head
        A.head ← A.head.next
      else
        t.next ← B.head
        B.head ← B.head.next
      end if
      t ← t.next
    end while
    if A.head = null then
      t.next ← B.head
      B.head ← null
    else
      t.next ← A.head
      A.head ← null
    end if
  end if
  R ← uusi lista
  R.head ← h
  return R
```

Algorithm 3 Listojen limittäminen (rekursiivinen versio).

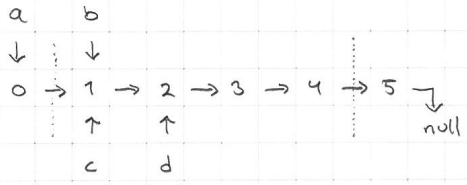
```
MergeRec(a, b)
  if a = null then
    return b
  else if b = null then
    return a
  else if a < b then
    a.next ← MergeRec(a.next, b)
    return a
  else
    b.next ← MergeRec(a, b.next)
    return b
  end if
Merge(A, B)
  R ← uusi lista
  R.head ← MergeRec(A.head, B.head)
  A.head ← null
  B.head ← null
  return R
```

6. Muuttujaan A haetaan listan alkio $L[x-1]$ (jos $x > 0$), B:hen alkio $L[x]$. Toisessa for-lauseessa käännetään alkion $L[i+1]$ (muuttuja D) next-osoitin osoittamaan alkioon $L[i]$ (C). C:ssä on lopulta alkio $L[y]$, D:ssä alkio $L[y+1]$ tai null, jos $y = n-1$. Kun $y = n-1$, joudutaan koko lista käymään kerran läpi. Algoritmin aikavaatimus on siis $\mathcal{O}(n)$. Algoritmin toimintaa on havainnollistettu oheisella kuvalla.

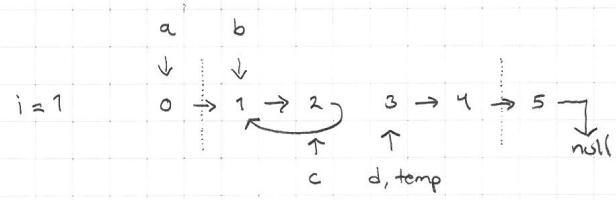
```
ReversePart(L, x, y)
  if L.head = null then
    virhe : tyhjä lista
  end if
  A ← null
  B ← L.head
  for i ← 0  to  x − 1 do
    A ← B
    B ← B.next
    if B = null then
      virhe : liian suuri indeksi
    end if
  end for
  C ← B
  D ← C.next
  for i ← x  to  y − 1 do
    if D = null then
      virhe : liian suuri indeksi
    end if
    temp ← D.next
    D.next ← C
    C ← D
    D ← temp
  end for
  B.next ← D
  if x = 0 then
    L.head ← C
  else
    A.next ← C
  end if
  return  L
```

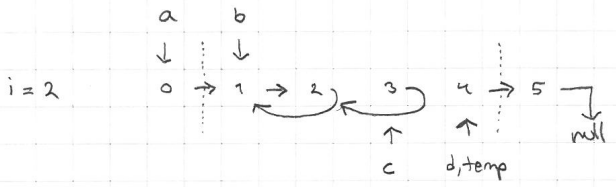
L = [0, 1, 2, 3, 4, 5], x = 1, y = 4

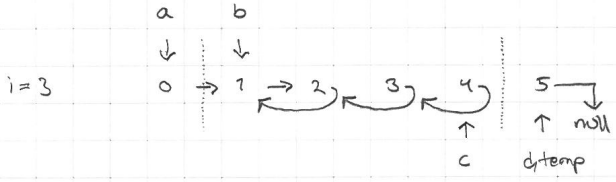
Situation after the first for loop and initialisation of pointers c and d:



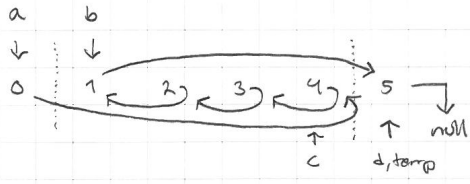
Situation after each iteration of the second for loop:







Situation after line "b.next ← d" and final conditional statements:



7. Toimintaidea: lukujen A ja B numeroita vertaillaan pareittain oikealta vasemmalle (eli samassa järjestyksessä kuin numerot on listoissa). Näin toimittaessa viimeisin toisistaan eroava numeropari määrää lukujen A ja B suuruusjärjestyksen. Muuttuja *diff* pitää kirjaa viimeksi havaitun vertailun tuloksesta. Jotta pareittain vertailu onnistuisi eripituisten lukujen kanssa, voidaan ajatella että lyhyemmän luvun edessä on nollia (esim. lukuja 125 ja 23 verratessa verrataan lukuesityksiä 125 ja 023). Miinusmerkki voidaan tulkita negatiiviseksi luvuksi. Esitetyssä ratkaisussa sille annetaan arvo -1. Näin esim. lukujen -234 ja 95 vertailu toteutetaan ikäänkuin verrattaisiin yksi kerrallaan listojen (4,3,2,-1) ja (5,9,0,0) numeroita. Rivillä 11 tutkitaan poikkeustapaus, jossa A ja B sisältävät yhtä monta numeroa ja ovat kumpikin negatiivisia. Tällöin aiemmin tutkittujen numeroiden perusteella tehty ratkaisu pitää kääntää toisinpäin (jos $A < B$, niin $-A > -B$).

8. (a) i) A
   ii) E, I, G, H, D
   iii) A, B, C, F
   iv) A

   (b) i) E, F, G
   ii) A, B
   iii) B, E, F, G, I
   iv) C, D

   Huom. b)-kohdassa solmu B on oma esivanhempansa ja jälkeläisensä. Esivanhemmuuden ja jälkeläisyyden lisäksi määritellään *aidot* esivanhemmat ja jälkeläiset, joihin solmu itse ei kuulu.

`ReadNumber`(x)

1: **if** $x =$ `null` **then**
2: $num \leftarrow 0$
3: **else if** x.val = '$-$' **then**
4: $num \leftarrow -1$
5: **else**
6: $num \leftarrow x$.val
7: **end if**
8: **return** num

`Next`(x)

1: $n \leftarrow x$
2: **if** $n \neq$ `null` **then**
3: $n \leftarrow n$.next
4: **end if**
5: **return** n

`IsGreater`(A, B)

alkuehto: A and B contain correct list representations of an integer

1: $a \leftarrow A$.head
2: $b \leftarrow B$.head
3: $diff \leftarrow 0$
4: **while** $a \neq$ `null` **or** $b \neq$ `null` **do**
5: $aNum \leftarrow$ `ReadNumber`(a)
6: $bNum \leftarrow$ `ReadNumber`(b)
7: **if** $aNum > bNum$ **then**
8: $diff \leftarrow 1$
9: **else if** $aNum < bNum$ **then**
10: $diff \leftarrow -1$
11: **else if** $aNum = bNum = -1$ **then**
12: $diff \leftarrow -diff$
13: **end if**
14: $a \leftarrow$ `Next`(a)
15: $b \leftarrow$ `Next`(b)
16: **end while**
17: **if** $diff = 1$ **then**
18: **return** `true`
19: **else**
20: **return** `false`
21: **end if**