ÖSSZJEGYZET

EA JEGYZET

1)

//Tömb monoton növekvő rendezése

A: T[n] == A/0: T[n] <- 0tól indexelünk a tömbben

A/1: T[n] <- 1től indexelünk a tömbben

A[0..n) <- 0tól n-ig indexelünk az n nincs benne

//Beszúró rendezés = Insertion Sort

5 2 714638

25 7 14638

257 1 4638

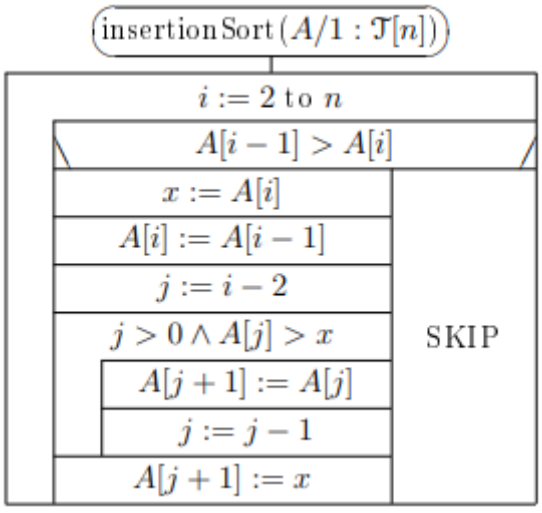
1257 4 638

12457 6 38

124567 3 8

1234567 8

12345678



(insertionSort stukk)

-> n-1 db iteráció

-> minimális idő n

-> maximális absztrakt idő n + (n-1)*(n-2)/2

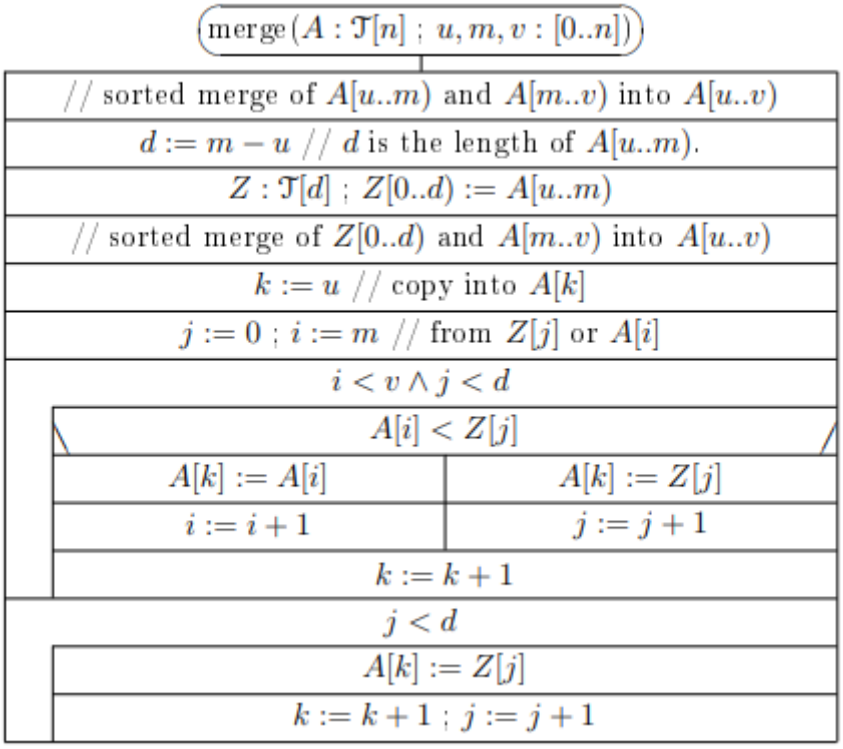
-stabil, ha valami a helyén van nem mozgatja

-optimális, ha előre rendezett a tömb

//Összefésülés=Mergesort

"oszd meg és uralkodj elv"

=> felosztjuk részekre és a részeket rendezzük sorba, majd a részeket rendezzük tovább



(mergeSort stukk)

2)

//Gyorsrendezés=Quicksort

=> oszd meg és uralkodj elv

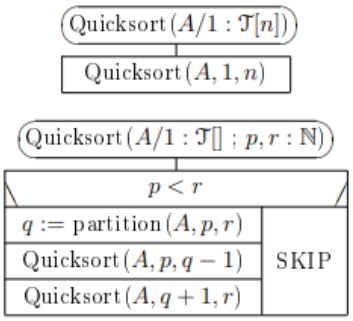
- kiválasztunk egy pivot elemet

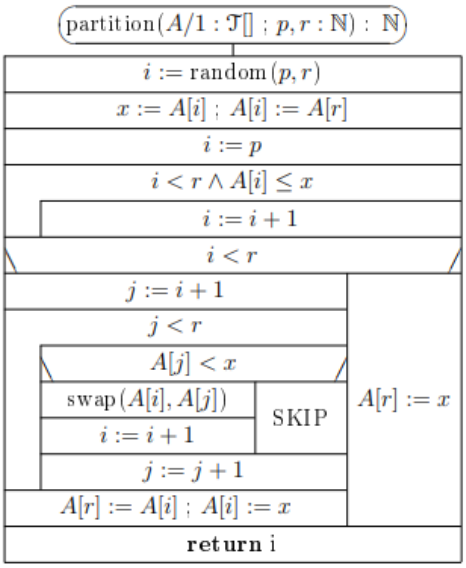
- részre bontás <- a kisebbeket balra, a nagyobbakat jobbra az egyenlőt mindegyhova

- a részeket rekúrzivan rendezi tovább a fenti lépések alapján

- az üres és az egyelemű tömbök a rekurzió alapesetei

- műveletigénye lineáris





(quickSort stukk)

mT(n)

AT(n) eleme omega(n log n)

MT(n) eleme omega(n^2)

//Vegyes gyorsrendezés

- a kisméretű tömbökre beszúró rendezést használunk <-hatékonyabb

3)

//Verem = stack

- LIFO = Last In First Out

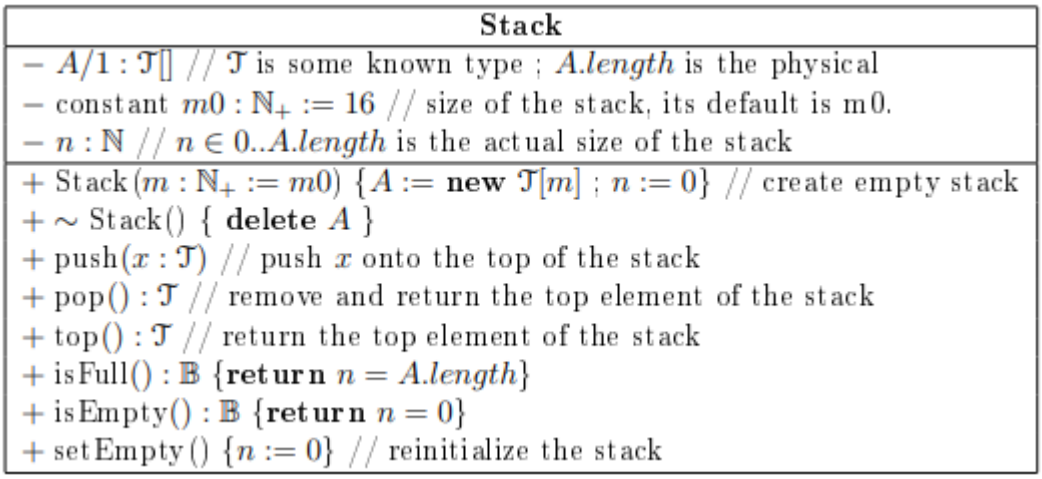
- dinamikus tömbbel reprezentáljuk

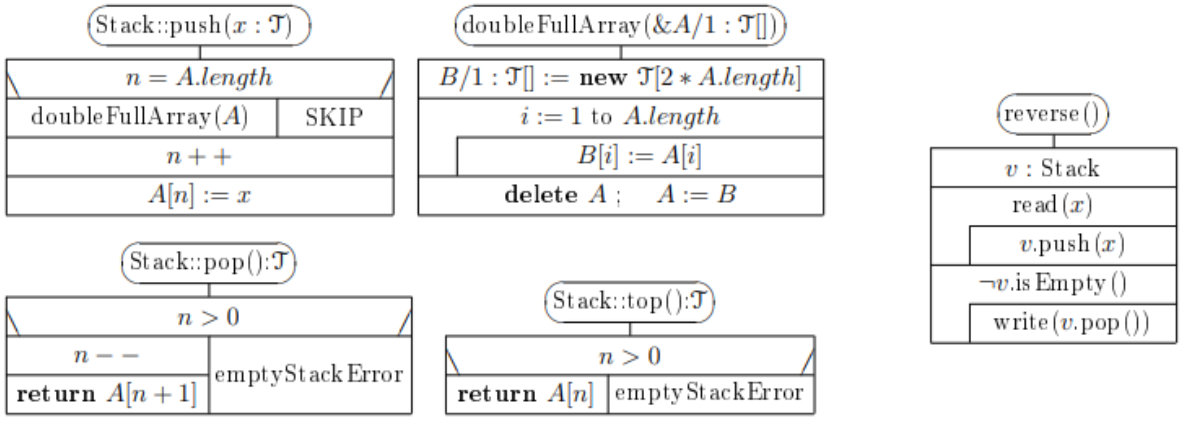
- A.length a verem fizikai mérete

- T a verem elemeinek típusa

pl. A/1: T[]

- tegyük fel h a read a kurrens inputról olvas be és a write a kurrens outputra ír ki





(verem műveletek stukk)

mT(n) eleme omega(1)

MT(n) eleme omega(n)

AT(n) eleme omega(1)

//Sor = queue

- FIFO = Firs In First Out

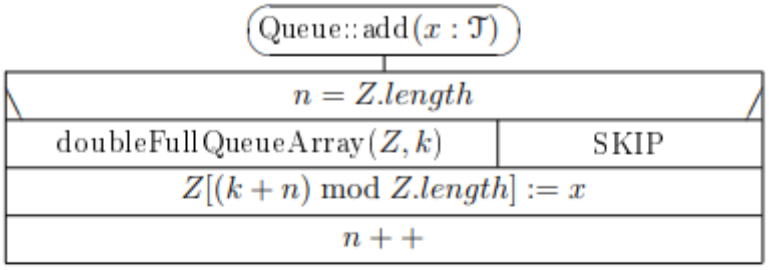
- nullától indexelt dinamikus tömb

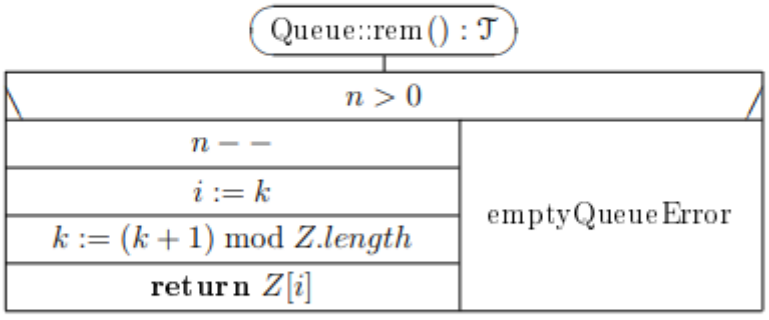
- Z.length a verem fizikai mérete

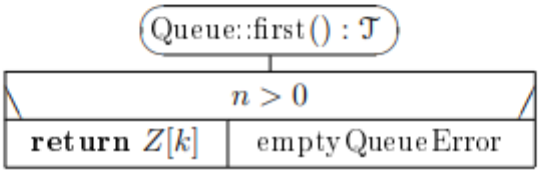
- T a verem elemeinek típusa

pl. Z: T[]

Queue
$-\ Z : \mathcal{T}[]$ // $\mathcal{T}$ is some known type ; $Z.length$ is the physical $-$ constant $m0 : \mathbb{N}_+ := 16$ // length of the queue, its default is m0. $-\ n : \mathbb{N}$ // $n \in 0..Z.length$ is the actual length of the queue $-\ k : \mathbb{N}$ // $k \in 0..(Z.length - 1)$: the starting position of the queue in Z
$+$ Queue$(m : \mathbb{N}_+ := m0)\{\ Z := \textbf{new}\ \mathcal{T}[m]\ ;\ n := 0\ ;\ k := 0\ \}$ // create an empty queue $+$ add$(x : \mathcal{T})$ // join x to the end of the queue $+$ rem$() : \mathcal{T}$ // remove and return the first element of the queue $+$ first$() : \mathcal{T}$ // return the first element of the queue $+$ length$() : \mathbb{N}$ $\{\textbf{return}\ n\}$ $+$ isFull$() : \mathbb{B}$ $\{\textbf{return}\ n = Z.length\}$ $+$ isEmpty$() : \mathbb{B}$ $\{\textbf{return}\ n = 0\}$ $+ \sim$ Queue$()$ $\{$ **delete** Z $\}$ $+$ setEmpty$()$ $\{n := 0\}$ // reinitialize the queue







(sor stukk)

mT(n) eleme omega(1)

MT(n) eleme omega(n)

AT(n) eleme omega(1)

//Láncolt listák = linked lists

- véges sorozatok tárolására alternatív megoldás

lineáris adatszerkezet = a tömbök és láncolt listák véges sorozatokat tárolnak <- lineáris struktura

- lehetnek egyirányú és kitirányú listák

//Egyirányú listák = one way/singly linked lists

- egyszerű <- S1L = Simple 1way List

- fejelemes <- H1L = Header node + 1way List

- két része van az elemeinek

    - a bal oldali az érték

    - a jobb oldali egy pointer ami a következő elem értékére mutat

//Egyszerű egyirányú listák S1L

L1->key <- az értékre mutat

L1->next <- a következő elemre mutat

L1->next->key <- a következő elem értékére mutat

4)

//Fejelemes listák H1L

- tartalmaz egy nulladik úgynevezett fejelemet

- a fejelem key mezője definiálatlan

- az üres H1L listának is van fejeleme

//Egyirányú listák kezelése

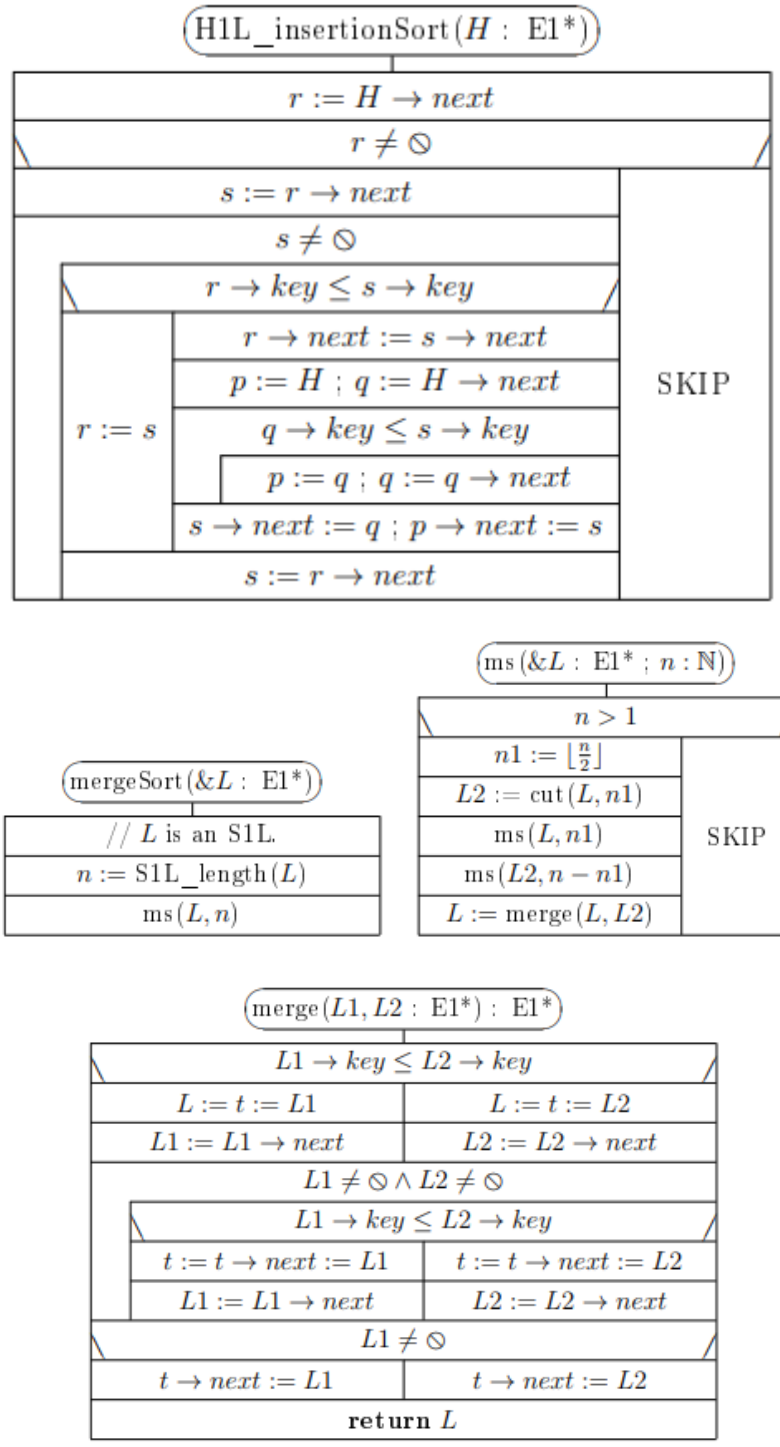
- memória és futási idő szempontjából egyszerű listák használata javasolt

- a fejelem helyett van amikor érdemesebb végelemet generálni

//Dinamikus memóriagazdálkodás

new és delete utasítások használata

//Rendezések H1Lre



- a cut levágja az első n elemet

//Ciklikus egyirányú listák

- az utolsó listaelem nextje nem null, hanem visszamutat az első elemre ha fejelemes lista akkor a fejelemre mutat

- ha üres a fejelemes lista akkor önmagára mutat

//Kétirányú listák = two way/doubly linked lists

- egyszerű <-S2L = Simple 2way List

- ciklikus <-C2L = Cyclic 2way List

- next és prev pointerek

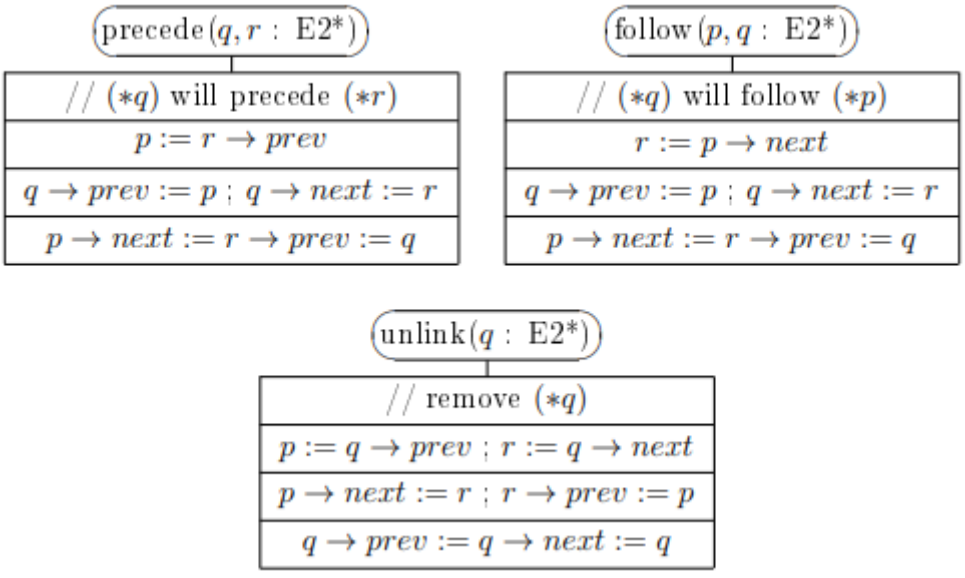
//Egyszerű kétirányú lista S2L

- kevésbé használt mert a beszúrás más ha előre szúród vagy ha középre vagy ha hátra

//Ciklikus kétirányú lista C2L

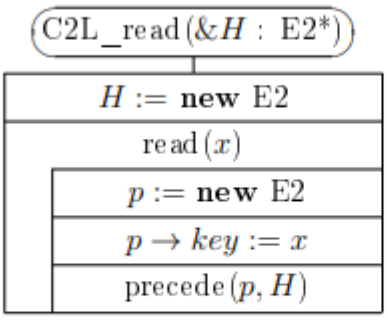
- lehet fejelemes vagy anélküli, alapértelmezésben maradjon fejelemes<- könnyebben használható

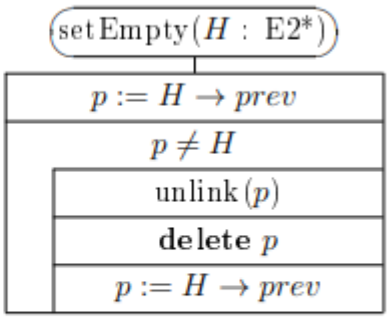
- műveletei: precede, follow, unlink



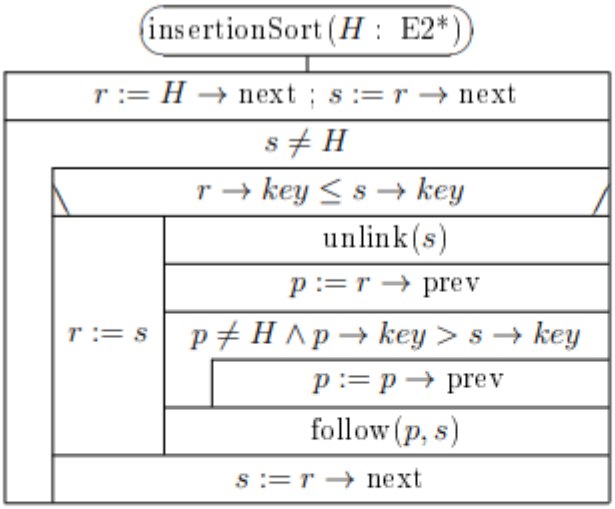
//Rendezések C2Lre

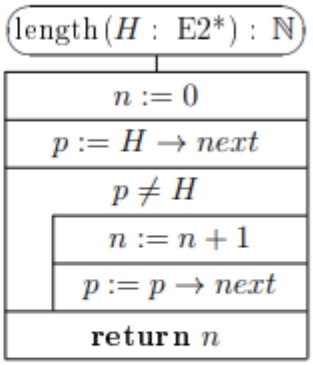
$H \to [/]—[5]—[2]—[7]—[/] \leftarrow H$ a $\langle 5; 2; 7 \rangle$ sorozat egy lehetséges ábrázolása,
$H \to [/]—[/] \leftarrow H$ a $\langle\rangle$ üres sorozaté.





$H \to [/]—[/] \leftarrow H$
$H \to [/]—[5]—[/] \leftarrow H$
$H \to [/]—[5]—[2]—[/] \leftarrow H$
$H \to [/]—[5]—[2]—[7]—[/] \leftarrow H$





$H \to [/]—\underline{[5]—[2]—[7]}—[2]—[/] \leftarrow H$
$H \to [/]—\underline{[2]—[5]—[7]}—[2]—[/] \leftarrow H$
$H \to [/]—\underline{[2]—[5]—[7]—[2]}—[/] \leftarrow H$
$H \to [/]—\underline{[2]—[2]—[5]—[7]}—[/] \leftarrow H$

5)

//Függvények aszimptomikus viselkedése

=> f fg akkor aszimptotikusan pozitív (=AP) fg, ha elg nagy n-re f(n) > 0

- AP fgk nagyságrendje = log n < n < n log n < n^2 < n^2 log n < n^3

- tulajdonságok: szimmetria, felcserélt szimmetria, tranzitivitás, reflexivitás, irreflexivitás

//NxN értelmezési tartományú fg

g: NxN -> R

fg AP, ha elég nagy n és elég nagy m értékre g(n,m)>0

GYAK JEGYZET

1)

//polinom helyttesítés

- n+1 méretű tömb, ha n-edfokú polinomunk van

A/1:T[n] <-1től indexelünk a tömbbe

A:T[n] <-0tók indexelünk

A.length <-tömb hossza

it(n) <-ciklusiterációk

S(n) <-szorzások

Ö(n) <-összeadások

pl. p(x) = 3x^3 + 2x^2 -x + 5

=> Z = [5, -1, 2, 3]

0 1 2 2 <-indexek

Ordo = O(g) <- olyan f függvényeket tartalmaz amelyre létezik c > 0, és elég nagy eleme n-re c*g(n) > f(n)

Omega = omegajel(g) <- olyan f függvényeket tartalmaz amelyre létezik d > 0, és elég nagy eleme n-re d*g(n) < f(n)

Teta <= körben -jel <- ha működik az ordo és omega akkor onnan következik a teta

négyzetes vs lineáris futási idő <- teta n^2 vs teta n <- teta n jobb

//Buborékos rendezés

- megnézi az első kettőt és ha az első nagyobb kicseréli
- célja hogy a legnagyobb elem hátra kerüljön
- mindig eggyel kevesebb elemet veszünk
- a cserék száma megegyezik az inverziókkal

mCs(n) <- minimális csere szám

MCs(n) <- maximális csere szám

ACs(n)<- átlagos csere szám

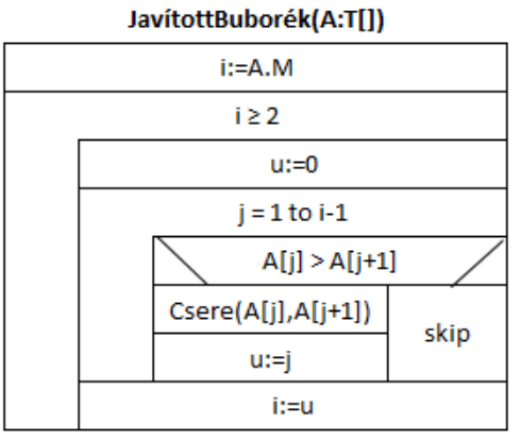
Buborék példa:

					Csere
3	5	2	4	1	0
3	5 ↔	2	4	1	1
3	2	5 ↔	4	1	1
3	2	4	5 ↔	1	1
3	2	4	1	5	*1. menet vége, 5 a helyén van*
3 ↔	2	4	1	5	1
2	3	4	1	5	0
2	3	4 ↔	1	5	1
2	3	1	4	5	*2. menet vége*
2	3	1	4	5	0
2	3 ↔	1	4	5	1
2	1	3	4	5	*3. menet vége*
2 ↔	1	3	4	5	1
1	2	3	4	5	*4. menet vége, rendezett a tömb*

Csere összesen: 7

Összehaonlítás összesen: 10

(buborék példa)



(javított buborék stukk)

//Maximum kiválasztásos rendezés

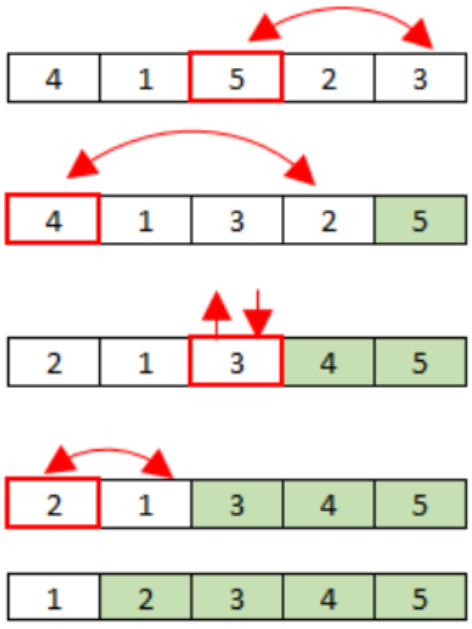
- az első elem lesz automatikusan a maximum

- végigmegy és összehasonlítja az elemeket a maximummal, ha nagyobb akkor a nagyobb lesz a maximum

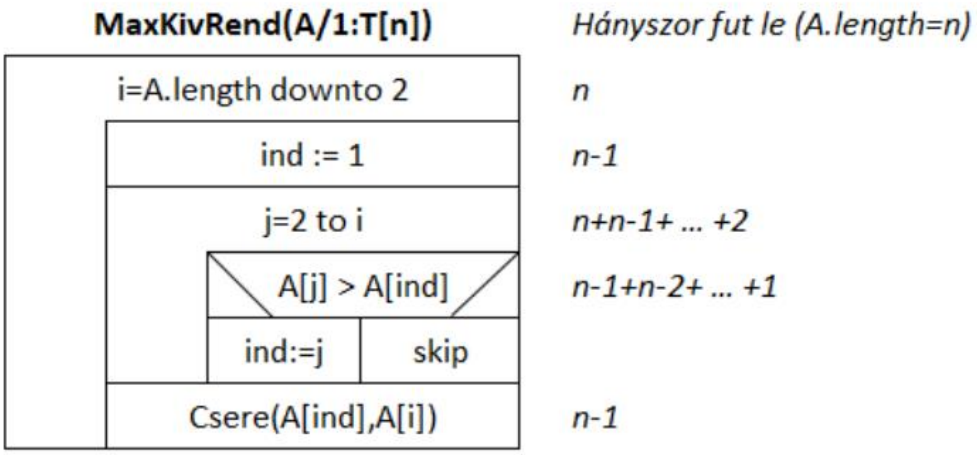
- a végén a maximum helyet cserél az utolsó elemmel

- következő futásoknák a végéről mindig -1 elemmel cserélődik ki

- ha saját magával cserélünk valamit, azt is cserének számolja



(maximum kiválasztás példa)



(maximum kiválasztás stukk)

2)

//Nevezetes nagyságrendek

konstans idejű <- verem vagy sor bármely műveletre

logaritmikus algoritmus <- bináris keresés pl. hw1

lineáris algoritmus <- maximum kiválasztás

n*log n algoritmus <- összefésülő rendezés

n^2 algoritmus <- beszúró rendezés

n^3 algoritmus <- mátrixszorzás

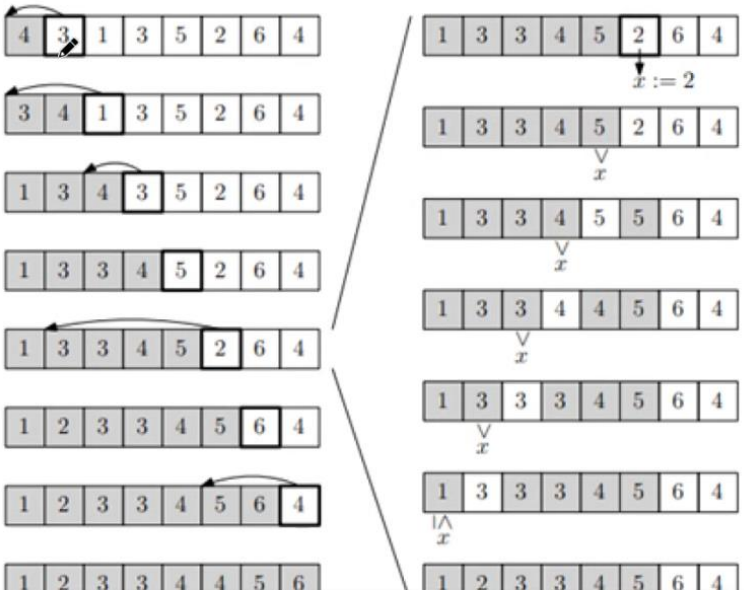
2^n algoritmus <- Hanoi-tornyai (=korongok/kövek átpakolása egyesével, kisebbre nem kerülhet nagyobb)

n! algoritmus <- utazóügynök-probléma

//Beszúró rendezés

- első elem fix

- sorra veszi az elemeket, hogy az első elem előtt vagy mögött vannak, majd beszúrja



(insertionSort példa)

Stabil rendezés = ha több egyforma elem van, a balra lévő elem bal oldalt marad

pl. 1 5 b2 3 j2 => 1 b2 j2 3 5

//Összefésülő rendezés

- szétszedi az elemeket fele fele
- majd azt is fele fele
- addig megy a fele fele, amíg már csak 1 vagy 2 elemű részek maradnak
- majd ezeket a részeket összehasonlítja külön külön
- majd lépésenként összevonja a részeket és összehasonlítja azokat is

//Verem

- folyamatosan dobálok bele elemeket
- mindig a legutoljára bekerült elemet vesszük ki

3)

LIFO = Last In First Out

push <- új elem

pop <- visszaadja a legutoljára bekerült elemet, törli a veremből

top <- visszaadja a legutoljára bekerült elemet és NEM törli

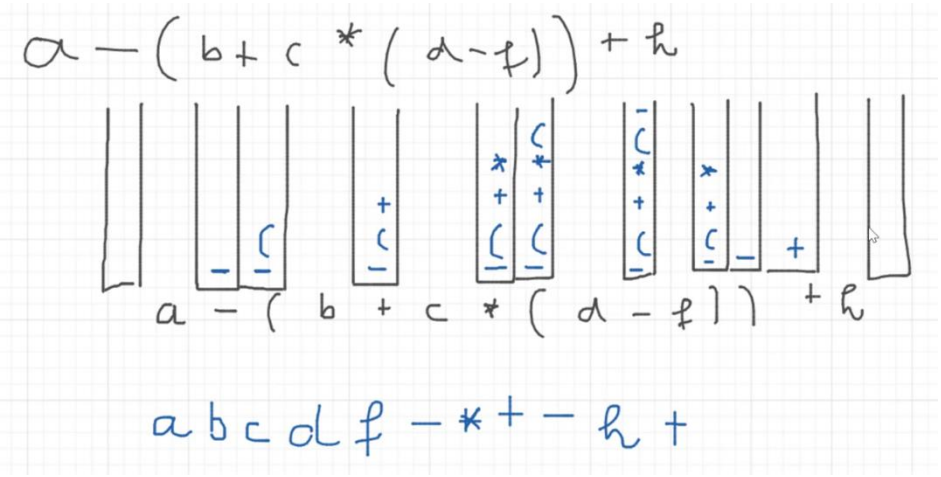
read() <- beolvas

//Lengyel forma = aritmetikai kifejezés postfix alakja

- infix alak pl. A + B, A + B * C
- prefix alak pl. +AB, +A*BC <- az operátor legelől van
- postfix alak pl. AB+, ABC*+ <- az operátor leghátul van
- hatványozás jobbról balra, minden mást balról jobbra

Feladat: x ab + cd - * f g h -^ / i + e - i - =

A) (a + b) * (c – d) / f ^ (g – h) + j – l – i kifejezés l
felírni, még nem az algoritmust használva.)
B) Értékadó operátor hatása, Hova illik az értékadó op
x = (a + b) * (c – d) / f ^ (g – h) + j – l – i



operandust kiír, ha operátor bekerül a verembe, ha nyitó zárójel bekerül a verembe, ha csukó zárójel kiírni mindent a nyitó zerójel fölött

//Lengyel forma kiértékelés <-visszafele végigmegy az algoritmuson

//Quick sort = Gyors rendezés

- legrosszabb esetben n^2
- várható érték: nlogn

- véletlenszerűen kiválasztjuk a pivot elemet, elintézzük hogy minden nála kisebb egyenlő balra és minden nagyobb egyenlő jobbra kerüljön, majd rekúrzívan folytatódik a rendezés

4)

Feladat

x=a+(-b^c^2+d*e)/((f+g)*h/-k)-p*z

lengyel forma: xab-c2^^de*+fg+h*k-//+pz*-=

//Gyorsrendezés/Quick Sort

-random választ pivot elemet <- !!ZHn az első elem legyen

pl.

A=[24,9,2,19,10,28]

x=24 [28,9,2,19,10,] => [9,2,19,10,24,28] => x=9 [10,9,2.] => [2,9,10,19]

=>[2,9,10,19,24,28]

//Egyirányú lista

- pointerekkel lehet bejárni
- elem elérése: p->key, p->next
- új listaelem: p=new E1
- elem felszabadítása: delete p
- fajtái:
    - egyszerű egyirányú láncolt lista(SL1): ha L1 null pointer, a lista üres, másképp rámutat az első elemre
    - fejelemes egyirányú láncolt lista(HL1): van egy L2, ami null pointer <- mindig van egy eleme, ha üres akkor is
- műveletei: keresés, beszúrás, törlés

5)

//Fejelemes kétirányú ciklikus litások (C2L)

prev <- az aktuális elemet megelőző elem

next <- a következő elem

- műveletei:

precede: listaelem beszúrása egy másik listaelem elé

follow: beszúrás a listaelem mögé

unlink: megadott listaelem kifűzése a listából

6)

//Sor

- FIFO

- egyirányú lista

KVÍZ JEGYZET

I.

1) IGAZ

buborékrendezés átlagos futási ideje teta n^2 (körben mínusz)

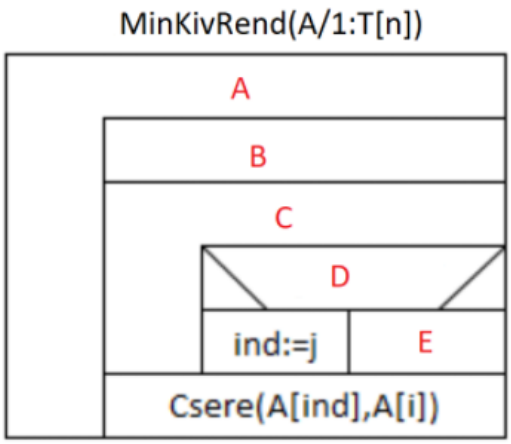
a buborékrendezésbe a cserék száma egyenlő az inverziók számával

2) Tömb: [15,19,6,8,11]

10 összehasonlítást végez a buborékrendezés: 4 + 3 + 2 + 1

a cserék száma 6

3) minimumKiválasztás stukk



i = 1 to n-1

ind := 1

j = n-1 downto i

A[j] < A[ind]

skip

4) Tömb: [12,6,9,8,10,1]

a tömb a maximum kiválasztás 3. menetének lefutása után: 1,6,8,9,10,12

II.

1) IGAZ

A beszúró rendezés helyben rendező algoritmus.

Az összefésülő rendezés több részfeladatra osztja a rendezést és azokat rekúrzívan oldja meg.

A vermek minden műveletének költsége teta 1

2) Beszúró rendezés, Tömb: [24,9,2,10,19,28,24,12]

kezdeti rendezett résztömb: A[1..1] = 24

2. menet után rendezett résztömb: A[1..3] = 2,9,24

5. menet után rendezett résztömb: A[1..6] = 2,9,10,19,24,28

2. menetben az összehasonlítások száma: 2, mozgatások száma 4

5. menetben az összehasonlítások száma: 1, mozgatások száma 0

3) Összefésülő rendezés, Tömb: [36,27,12,24,32,15,22,35,10]

rekúrzív felbontások

27,36

12,24

15,32

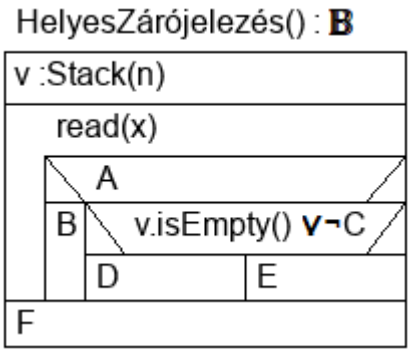
10,35

10,22,35

10,15,22,32,35

10,12,15,22,24,27,32,35,36

4) Verem



A => x = ’(’ v x = ’[’ v x = ’{’

B => v.push(x)

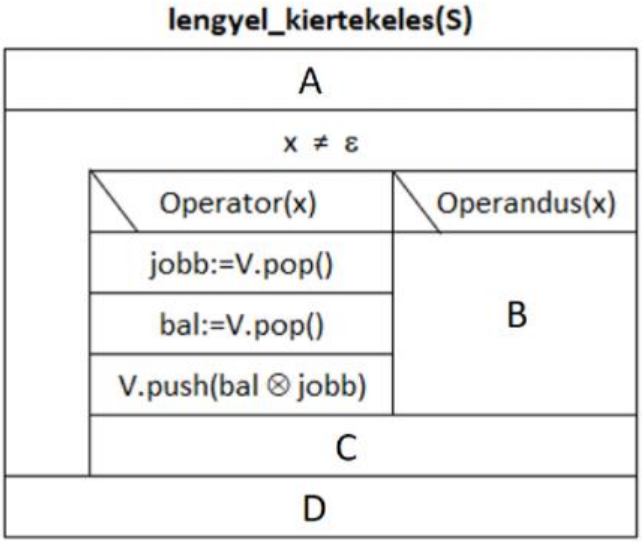
C => (v.top() = ‘(‘ ∧ x=’)’) ∨ (v.top() = ‘[‘ ∧ x=’]’) ∨ (v.top() = ‘{‘ ∧ x=’}’)
D => return false
E => v.pop()
F => return v.isEmpty()

III.

1) Igaz

lengyel forma egy aritmetikai kifejezés postfix alakja
a gyorsrendezés egy randomizált algoritmus

2) lengyelforma kiértékelés stukk



V: Stack; x:=read(S)

V.push(x)

x:=read(S)
write(V.pop())

3) kifejezés: 2 * (3 + 7 - 1) ^ (16 / 8 + 3)

lengyelforma = 2,3,7,+,1,-,16,8,/,3,+,^,*

IV.

1) Igaz quicksortra

A quicksort oszd meg és uralkodj elvű
A tengely kiválasztása és a részekre bontás mindig lineáris időben fejeződik be
A quicksort particionáló eljárása közben a tengellyel egyenlők a tengely elé, és mögé is kerülhetnek
A particionálás egy-elemű résztömbre nem hajtódik végre

2) befűzés r-t p mögé

(*r).next:=p→next; (*p).next:=r;
r→next:=p→next; p→next:=r;

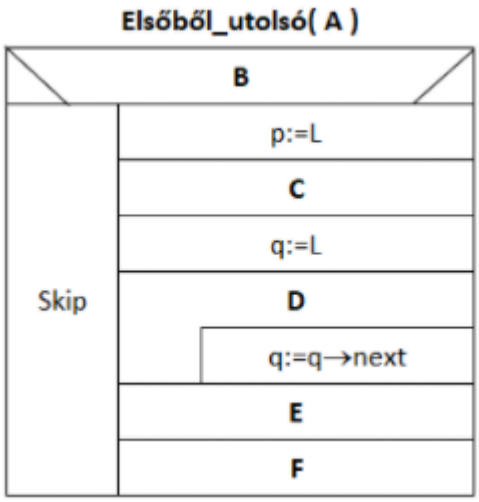
3) Igaz S1L-re

A lista utolsó elemének elérése θ(n), ahol n a lista elemszáma
p→next:=p→next→next; hatása: kifűzi a p utáni elemet a listából
Ha a lista egy p című eleme elé szeretnénk befűzni egy új elemet, akkor annak műveletigénye O(n), ahol n a lista elemszáma

4) Gyorsrendezés, Tömb: [8,7,13,6,11,3,1,9,4] (első elem=pivot)

4,7,6,3,1,+8,11,9,13
1,3,+4,7,6
+1,3
6,+7
9,+11,13
16 <- összehasonlítások száma

5) Első elem átfűzése leghátra



&L : E1*

L = 0 v L->next= 0

L := L->next

q->next != 0

q ->next := p
p -> next := 0

V.

1) metódusok

precede(q,r:E2*) = q beszúrása r elé
follow(p,q:E2*) = q heszúrása p mögé
unlink(q:E2*) = q kifűzése a listából

2) melyik művelet

$p := q \rightarrow prev \; ; \; r := q \rightarrow next$
$p \rightarrow next := r \; ; \; r \rightarrow prev := p$
$q \rightarrow prev := q \rightarrow next := \mathbf{this}$

⇨ unlink(q:E2*)

3) melyik művelet

$r := p \rightarrow next$
$q \rightarrow prev := p \; ; \; q \rightarrow next := r$
$p \rightarrow next := r \rightarrow prev := q$

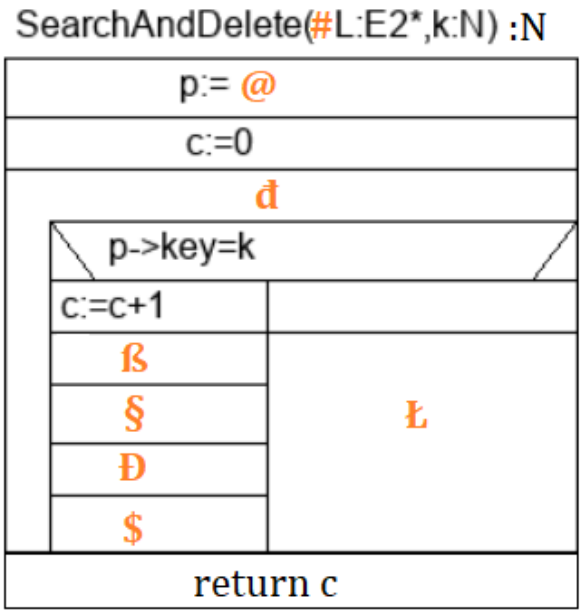
⇨ follow(p,q:E2*)

4) melyik művelet

$p := r \rightarrow prev$
$q \rightarrow prev := p \; ; \; q \rightarrow next := r$
$p \rightarrow next := r \rightarrow prev := q$

⇨ precede(q,r:E2*)

5) C2L hányszor szerepel és törlés



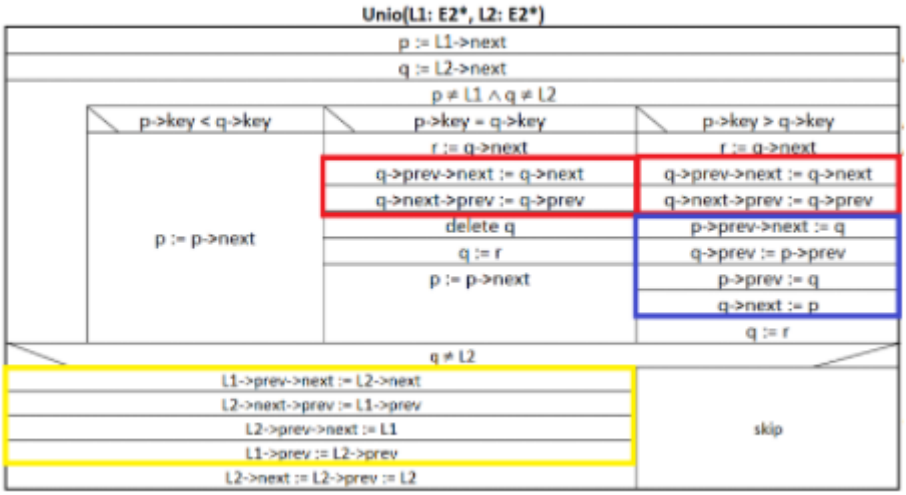
(üres)

p := L->next

p != L

r:=p->next
unlink(p)
delete(p)
p:=r

p:=p->next



6) IGAZ
a piros téglalap utasításai helyettesíthetők az unlink(q) utasítással

7) IGAZ
a kék téglalap utasításai helyettesíthetők a precede(q,p) utasítással

8) IGAZ
a kék téglalap utasításai helyettesíthetők a follow(p->prev, q)

9) HAMIS
a sárga téglalap utasításai helyettesíthetők a follow… <-doesn't matter hamis

10) mT(n,m) és MT(n,m)
ha L2 minden eleme nagyobb L1 legnagyobb eleménél: Theta(n)
ha L2 minden eleme kisebb L1 legkisebb eleménél: Theta(m)
MT(n,m) = Theta(n+m)

11) hibás sorműveletek
Q.rem(x)
x:=Q.top()

MINTAZH

1. feladat

Rendezd a megadott tömböt a beszúró rendezés segítségével, majd válaszolj a következő kérdésekre.
Input: [86,29,90,31,47,91,20,40]

(a) Hány összehasonlítás és hány mozgatás történt a 3. iterációban?

(b) Melyik iterációban történt pontosan 6 mozgatás?

(c) Összesen hány összehasonlítást végzett az algoritmus?

(d) Add meg a tömb tartalmát a 4. iteráció végén.

(1)
a) 3 összehasonlítás, 2 mozgatás
b) 6. iterációban
c) 20 összehasonlítás
d) 29,31,47,86,90

2. feladat

Rendezd gyorsrendezéssel az [5;2;7;1;6;4;8;3] tömböt! Feltesszük, hogy a particionálás minden esetben az aktuális résztömb első elemét választja tengelynek. Add meg sorban a $partition(A, p, r)$ segédfüggvény hívásai által kiszámolt résztömböket, az elemeik felsorolásával, a tengelyt + előjellel különböztetve meg!

(2) Tömb: [5,2,7,1,6,4,8,3]
2,1,4,3,+5,7,6,8
1,+2,4,3
3,+4
6,+7,8

(3) x = −x^2 + 5 ∗ k/(y − z ∗ 3 + s)^x^2 − b ∗ d − w
lengyel forma: xx-2^5k*yz3*-stx2^^/+bd*-w-=