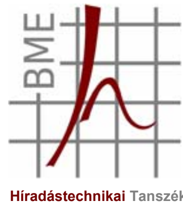
# A programozás alapjai 1

9. előadás




Dinamikus adatszerkezetek:

Dinamikus adatszerkezetek:

Adott építőelemekből,
adott szabályok szerint felépített,
de nem rögzített méretű adatszerkezetek.

A tényleges méret futás közben alakul ki.

A méret és a szerkezet is megváltoztatható.

Megvalósításához szükséges:

önhivatkozó struktúra

olyan összetett adattípus,
amely összetevőként tartalmaz olyan mutatót,
amely saját típusára képes mutatni.

## Önhivatkozó struktúrák

Struktúra deklaráció alakja:

```
struct   [<struktúra címke>]
{
<struktúra tag deklarációk>
}
[<struktúra változó azonosítók>] ;
```

## Önhivatkozó struktúrák

Önhivatkozó struktúra deklaráció alakja:

```
struct  <struktúra címke>
{
[<struktúra tag deklarációk>]
struct  <struktúra címke> * <azonosítók>;
[<struktúra tag deklarációk>]
}
[<struktúra változó azonosítók>] ;
```

## Önhivatkozó struktúrák

Példa:

```
struct elem1 {
        int adat;
        struct elem1 * kovetkezo;
       } e1,*e2;
struct elem1 e3,*e4;
```

## Önhivatkozó struktúrák

Példa:

```
typedef struct kamu {
        int adat;
        struct kamu * kovetkezo;
       } elem2;
elem2 e5,*e6;
```

Egyszerűbb esetben az adatszerkezet egy fa gráfot valósít meg.

A fa olyan irányított gráf, amelyben nincs kör, vagyis egyetlen pontjáról sem lehet visszajutni a kiindulás helyére.

K-ágú fa:

Minden csomópontból legfeljebb K él indul ki.

K = 1

egyirányban láncolt lista

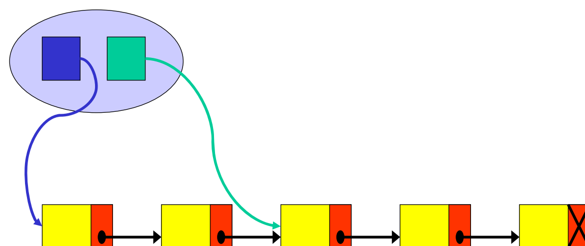
Egyirányban láncolt lista:

Hogy kezelni tudjuk,
két mutatóra van szükségünk:

Az egyik az adatszerkezet elejét jelzi,
A másik az éppen feldolgozott elemre mutat.

A listát ez a két mutató testesíti meg.

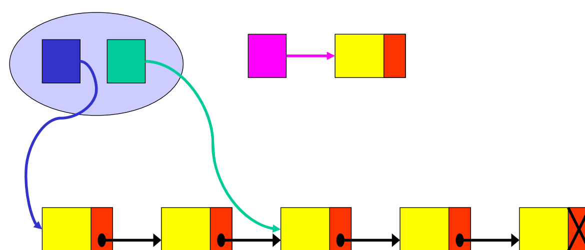
Egyirányban láncolt lista:



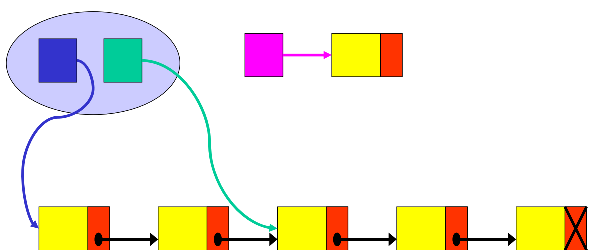
Hogy a tárolt adatokat vissza tudjuk keresni,
rendezni kell
(vagy eleve rendezetten felépíteni)

Kézenfekvő megoldás:
közvetlen beszúrásos rendezés.

A beszúráshoz egy segédmutató kell.

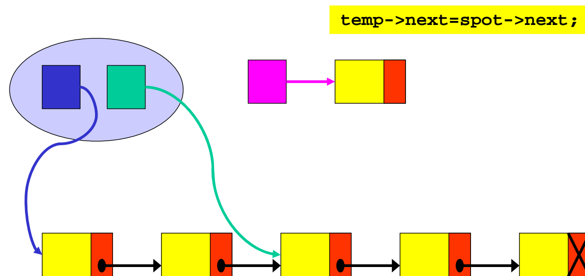


Beszúrni csak az aktuális elem mögé lehet!

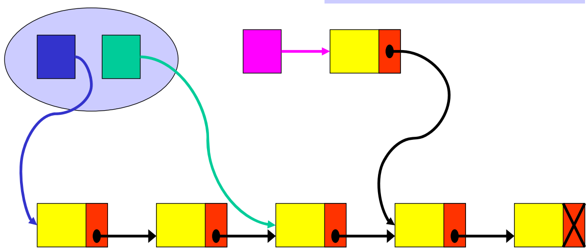


Beszúrni csak az aktuális elem mögé lehet!

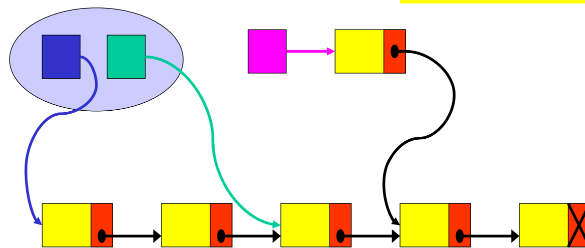
```
temp->next=spot->next;
```



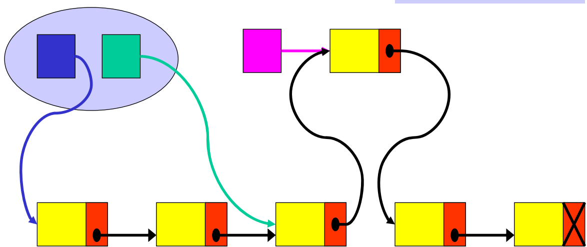
Beszúrni csak az aktuális elem mögé lehet!



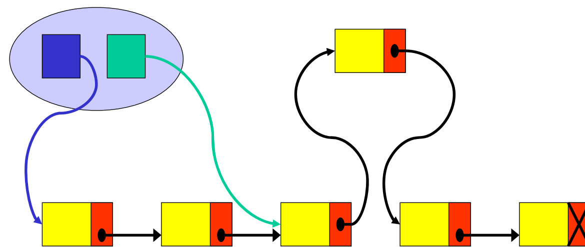

Beszúrni csak az aktuális elem mögé lehet!




Beszúrni csak az aktuális elem mögé lehet!




Beszúrni csak az aktuális elem mögé lehet!



A beszúrás helyét lineáris kereséssel találhatjuk meg:

A legelső elemtől kiindulva megkeressük az első olyan elemet, amely után következő elem kulcsa nagyobb a beszúrni kívánténál.

Ez a módszer hibás!

Nem lehet beszúrni az első elem elé, és az utolsó elem után.

Ezek esetleges szükségességét minden esetben külön meg kell vizsgálni.

Ez körülményes, ezért nem így járunk el.

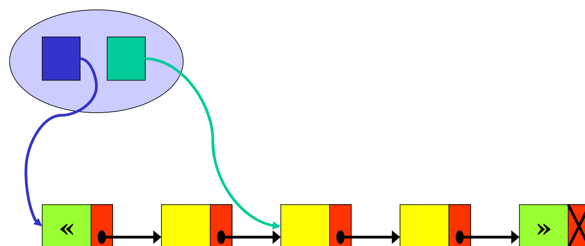
Helyette:

Mindkét végén kiegészítjük a listát egy-egy elemmel.

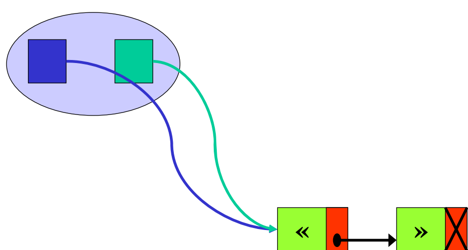
Egy olyannal, amelynek kulcsa kisebb minden elvileg tárolható eleménél, és egy olyannal, amelynek kulcsa nagyobb minden elvileg tárolható eleménél.

Ezeket az elemeket strázsának (sentinel) nevezzük.

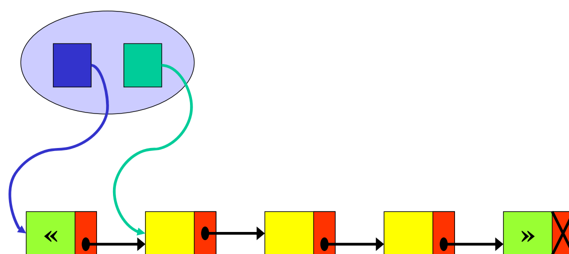
Egyirányban láncolt lista strázsával:




Az üres lista ezek szerint:

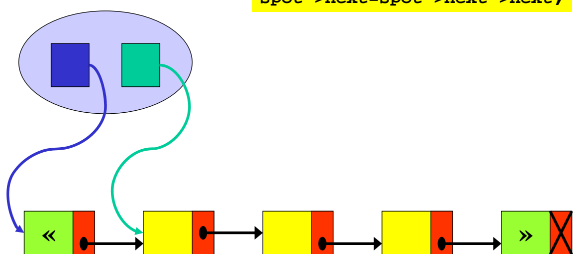



Elemet törölni is csak az aktuális elem mögül lehet!



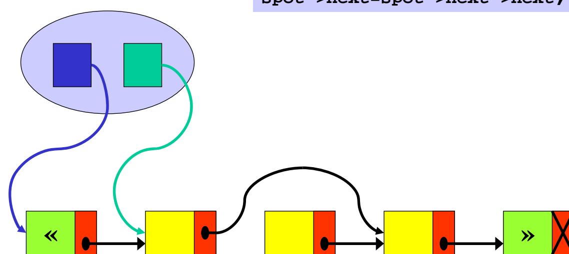

Elemet törölni is csak az aktuális elem mögül lehet!

```
spot->next=spot->next->next;
```

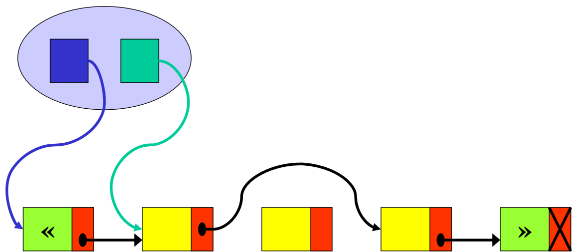



Elemet törölni is csak az aktuális elem mögül lehet!

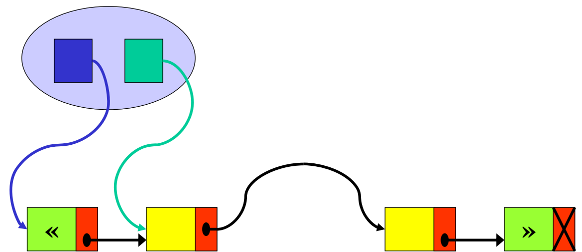
```
spot->next=spot->next->next;
```



Elemet törölni is csak az aktuális elem mögül lehet!




Elemet törölni is csak az aktuális elem mögül lehet!




Listák különleges alkalmazásai:

- verem (stack) (LIFO),
- várakozási sor (queue) (FIFO)
- prioritásos sor (priority queue)

Verem (Last In First Out):

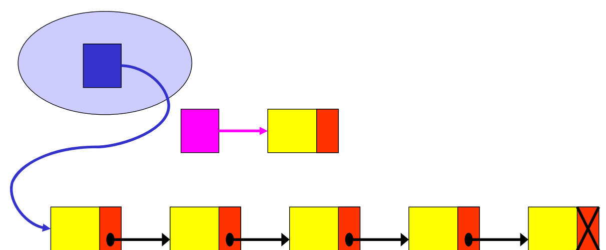
Elemet mindig csak a lista elejére szúrunk be; mindig csak a lista elejéről veszünk ki.

A veremet egyetlen mutató testesíti meg.

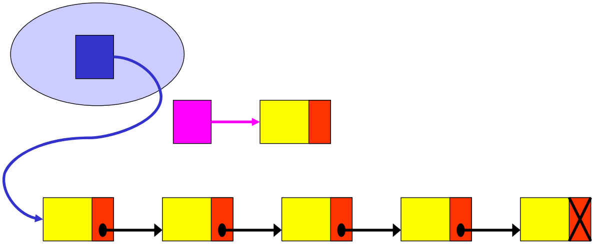
Beszúrni üres verembe is ugyanúgy lehet. Törölni az utolsó elemet is ugyanúgy lehet.

Strázsákra sincs szükség.

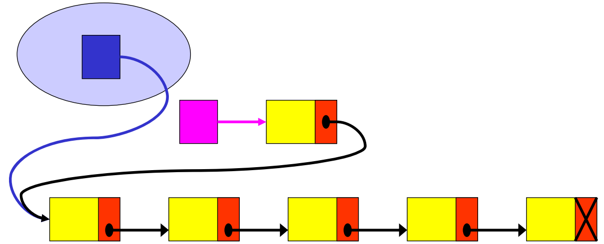
Beszúrás a verembe:



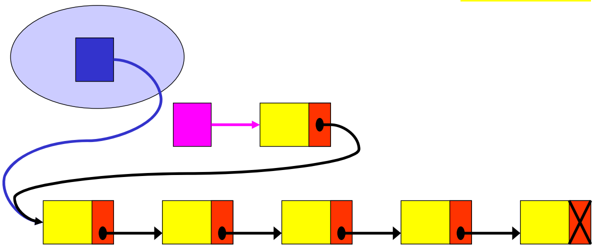
## Beszúrás a verembe:



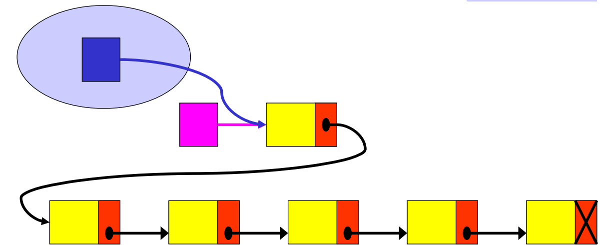

## Beszúrás a verembe:



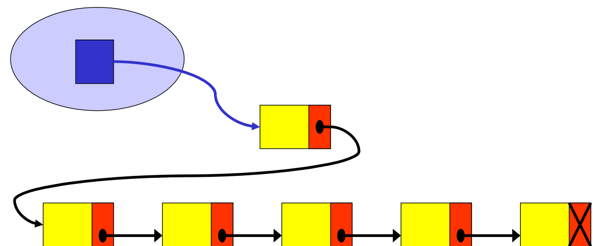

## Beszúrás a verembe:



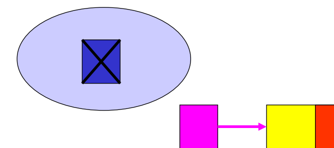

## Beszúrás a verembe:



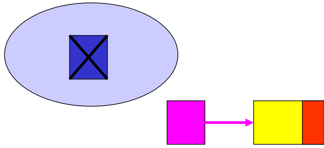

## Beszúrás a verembe:



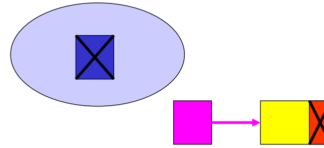
## Beszúrás üres verembe:



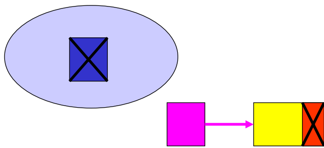
Beszúrás üres verembe:



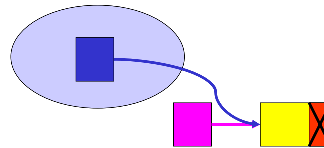

Beszúrás üres verembe:



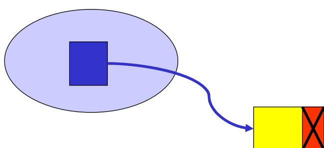

Beszúrás üres verembe:



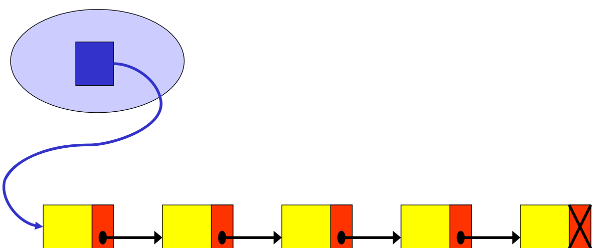

Beszúrás üres verembe:



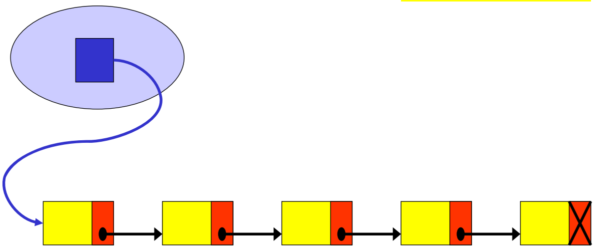

Beszúrás üres verembe:



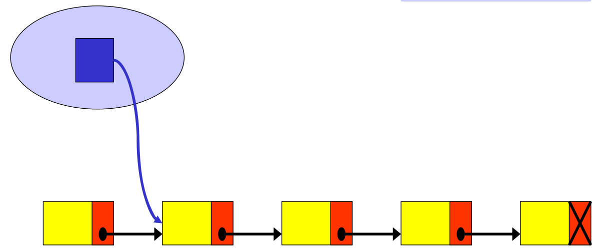
Elem törlése a veremből:



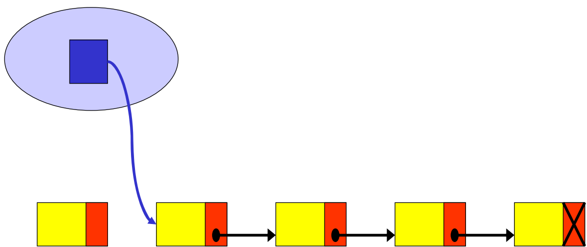
Elem törlése a veremből:



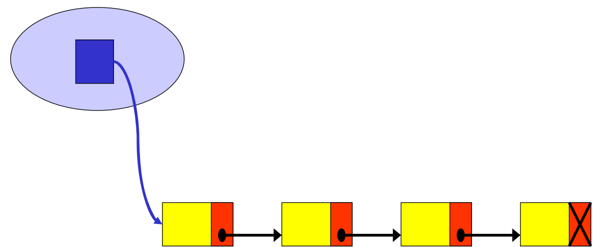

Elem törlése a veremből:



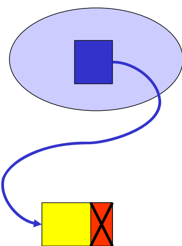

Elem törlése a veremből:



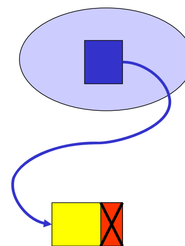
Elem törlése a veremből:



Az utolsó elem törlése a veremből:

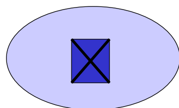


Az utolsó elem törlése a veremből:

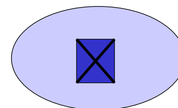


Az utolsó elem törlése a veremből:

```
head=head->next;
```



Az utolsó elem törlése a veremből:



Várakozási sor (First In First Out):

Elemet mindig csak a lista végére szúrunk be;
mindig csak a lista elejéről veszünk ki.

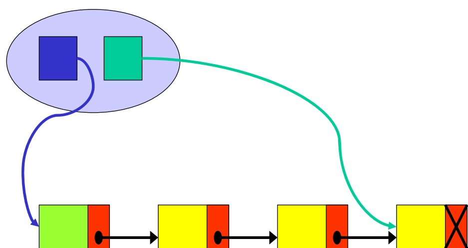
A várakozási sort két mutató testesíti meg.

Az egyik az első elemre mutat;
a másik pedig az utolsóra.

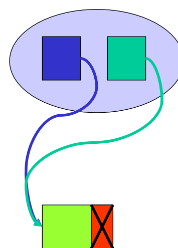
Beszúrni az utolsó elem mögé lehet.
Egy strázsa elég az elejére,
így mindig lesz utolsó elem,
akkor is, ha a sor üres.

Törölni mindig a strázsa utáni elemet kell.

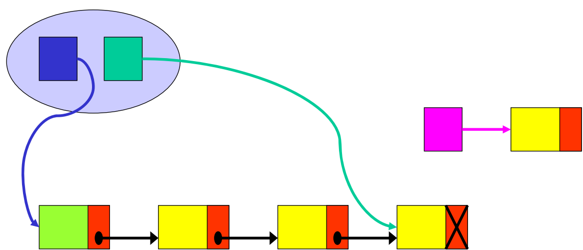
Várakozási sor:



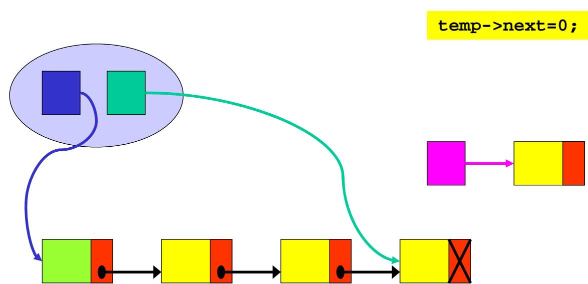
Üres várakozási sor:



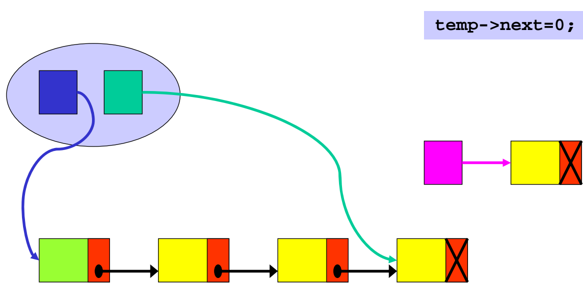
Beszúrás várakozási sorba:



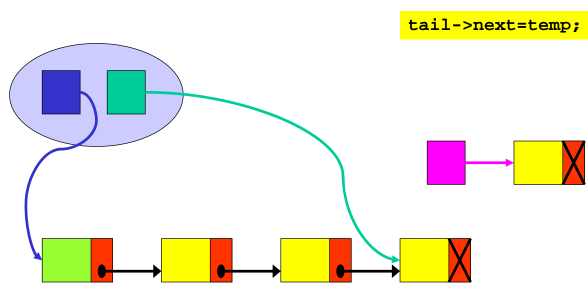
Beszúrás várakozási sorba:



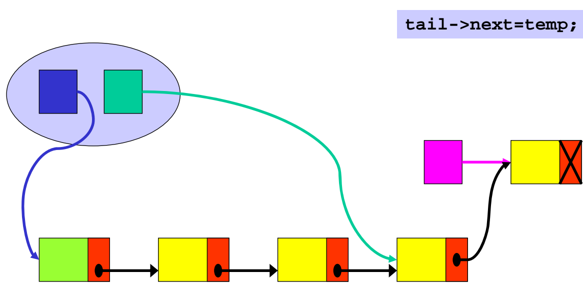

Beszúrás várakozási sorba:



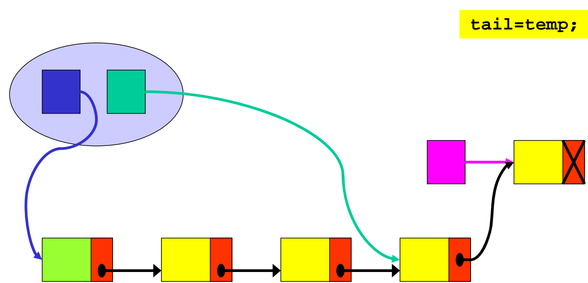

Beszúrás várakozási sorba:




Beszúrás várakozási sorba:

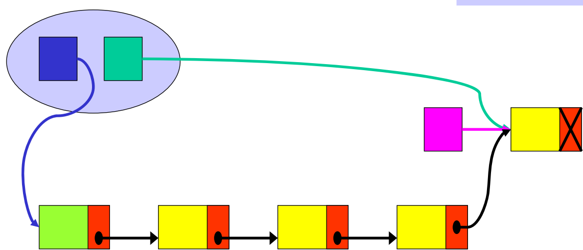



Beszúrás várakozási sorba:

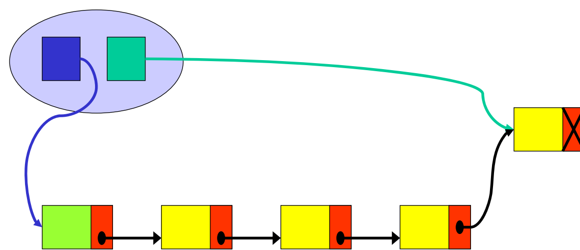


Beszúrás várakozási sorba:

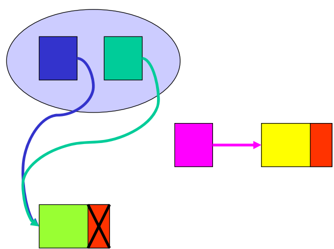
`tail=temp;`



Beszúrás várakozási sorba:

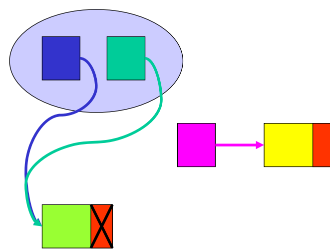


Beszúrás üres várakozási sorba:



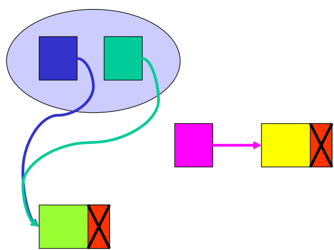
Beszúrás üres várakozási sorba:

`temp->next=0;`



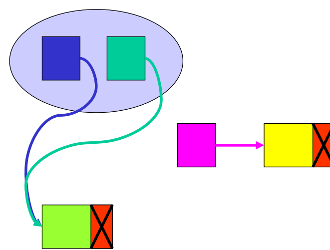
Beszúrás üres várakozási sorba:

`temp->next=0;`



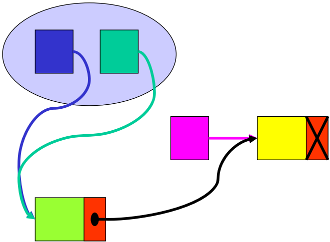
Beszúrás üres várakozási sorba:

`tail->next=temp;`



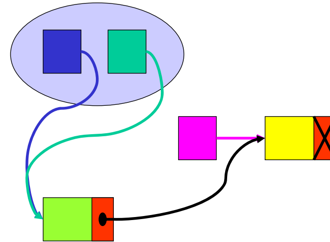
Beszúrás üres várakozási sorba:

`tail->next=temp;`



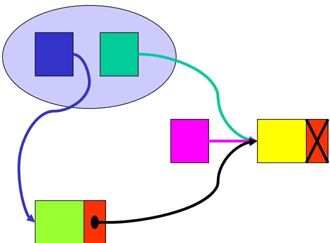
Beszúrás üres várakozási sorba:

`tail=temp;`

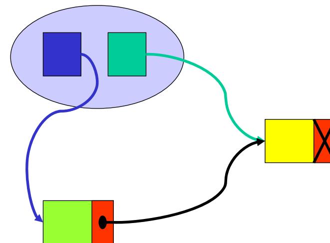


Beszúrás üres várakozási sorba:

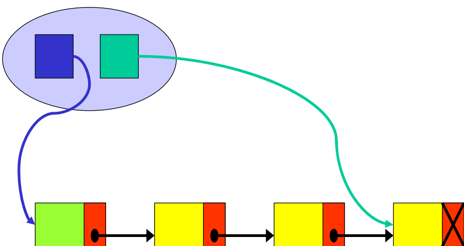
`tail=temp;`



Beszúrás üres várakozási sorba:

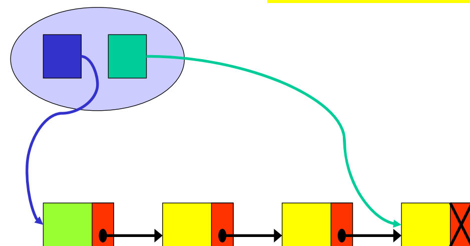


Elem törlése várakozási sorból:



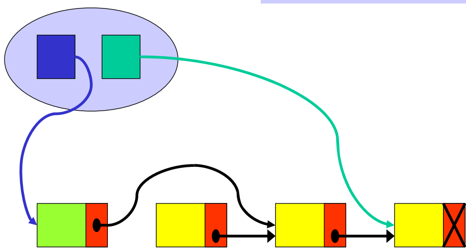
Elem törlése várakozási sorból:

`head->next=head->next->next;`

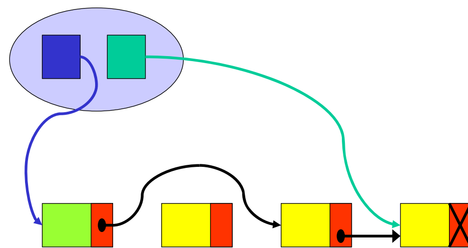


Elem törlése várakozási sorból:

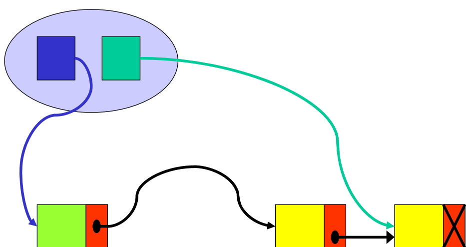
`head->next=head->next->next;`



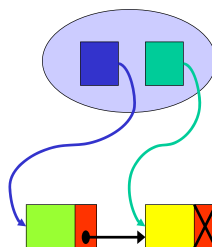
Elem törlése várakozási sorból:



Elem törlése várakozási sorból:

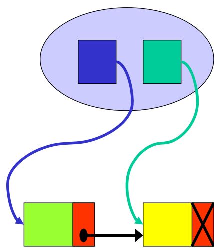


Utolsó elem törlése várakozási sorból:



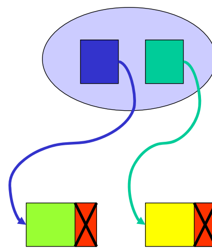
Utolsó elem törlése várakozási sorból:

`head->next=head->next->next;`

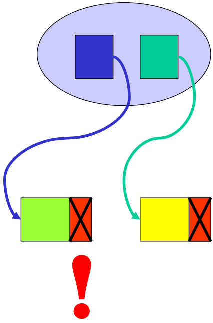


Utolsó elem törlése várakozási sorból:

`head->next=head->next->next;`

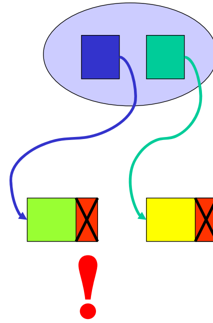


Utolsó elem törlése várakozási sorból:



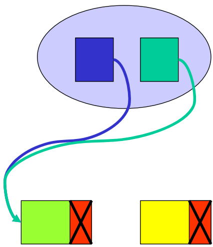
Utolsó elem törlése várakozási sorból:

```
if(!(head->next))tail=head;
```

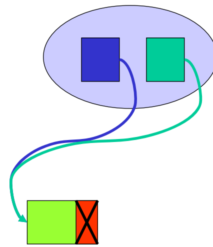


Utolsó elem törlése várakozási sorból:

```
if(!(head->next))tail=head;
```



Utolsó elem törlése várakozási sorból:



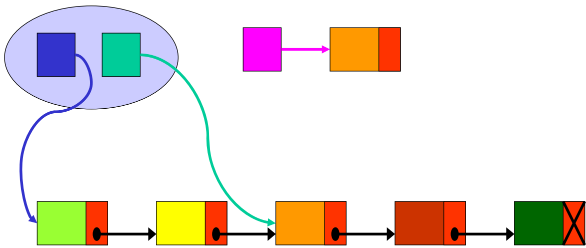
Prioritásos sor:

Elemet mindig csak a magasabb prioritású
vagy megegyező prioritású
de régebben várakozó
elemek mögé szúrunk be;
mindig csak a lista elejéről veszünk ki.

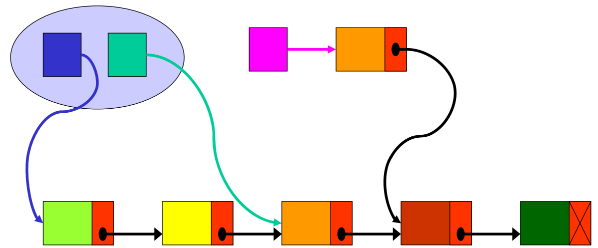
Megvalósítható a lista első változatával.

Mivel beszúrásra a legelején és a legvégén is sor kerülhet, kell mindkét strázsa.

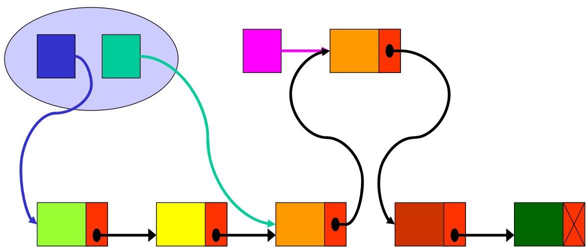
Beszúrás prioritásos sorba:



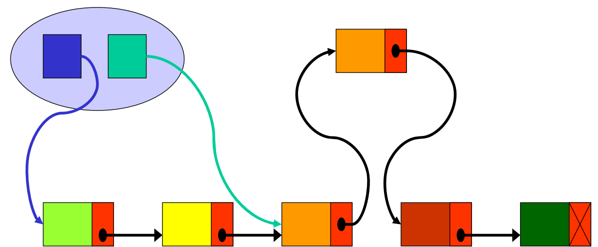
Beszúrás prioritásos sorba:



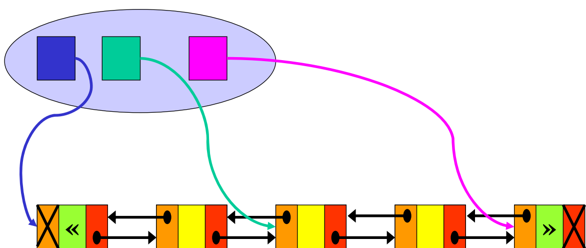
Beszúrás prioritásos sorba:



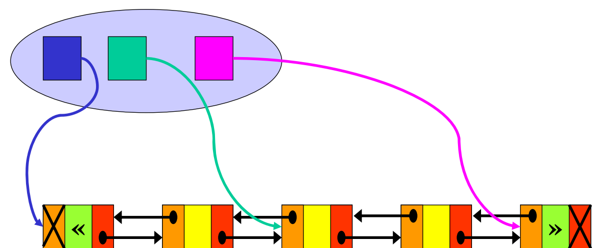
Beszúrás prioritásos sorba:



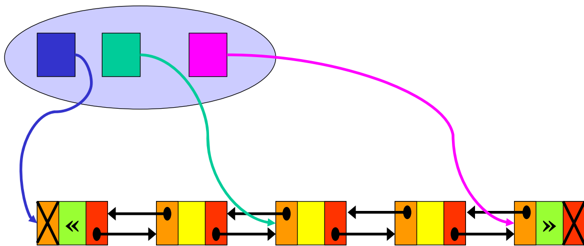
Ez NEM fa, de nagyon gyakran használjuk!




Ez NEM fa, de nagyon gyakran használjuk!
Két irányban láncolt lista strázsával:

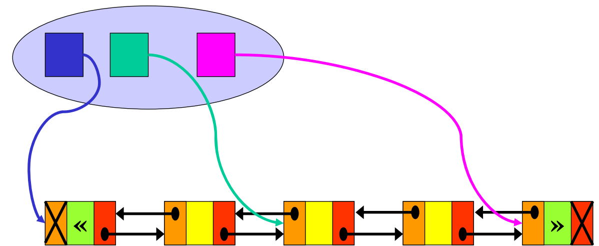


Ez NEM fa, de nagyon gyakran használjuk!
Két irányban láncolt lista strázsával:




Rekurzívan rendezhető gyorsrendezéssel!

Ez NEM fa, de nagyon gyakran használjuk!
Két irányban láncolt lista strázsával:




Cserénél vigyázni kell
a mutatók átállításának sorrendjére!

gyorsrendezés
(quick sort)

Alapgondolata:
Válasszuk szét a tömb elemeit a mediánnál kisebb, és annál nagyobb kulcsúakra!
A két felet rendezzük ugyanígy!

Helyben szétválogatási feladat

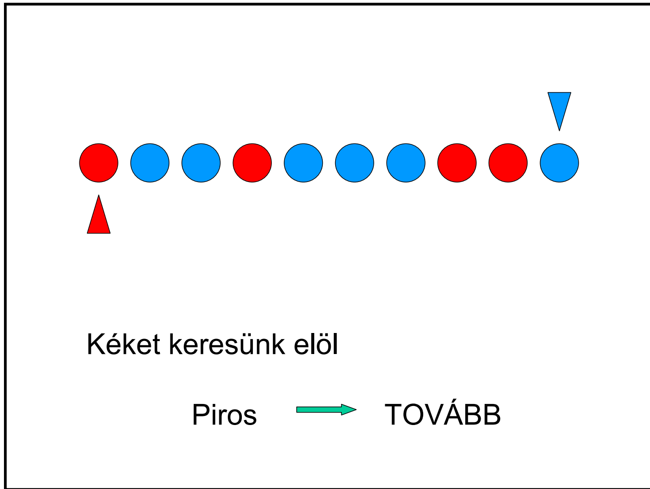
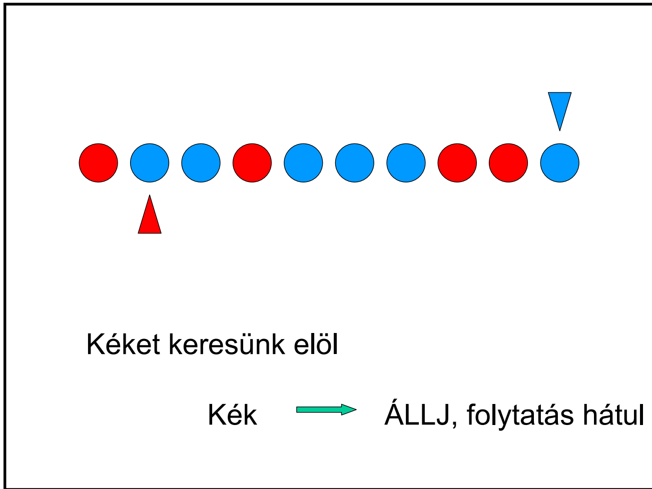
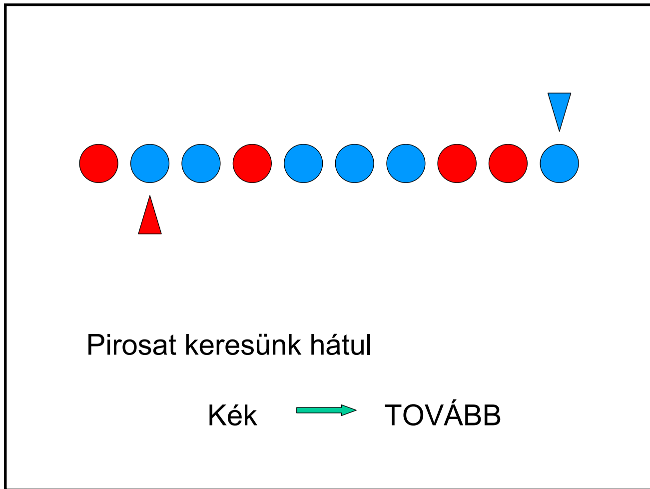
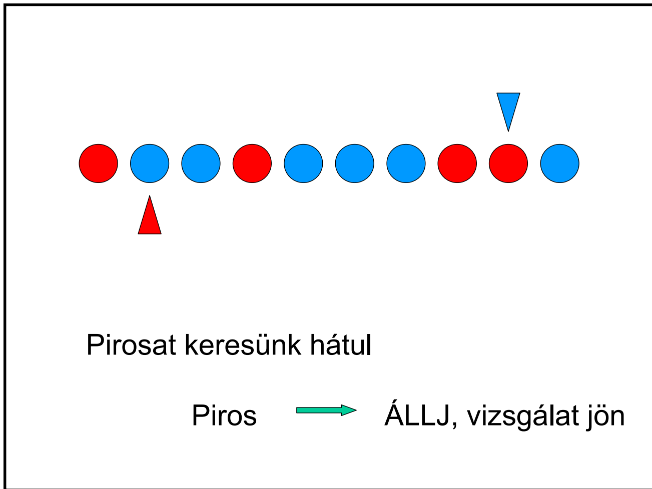
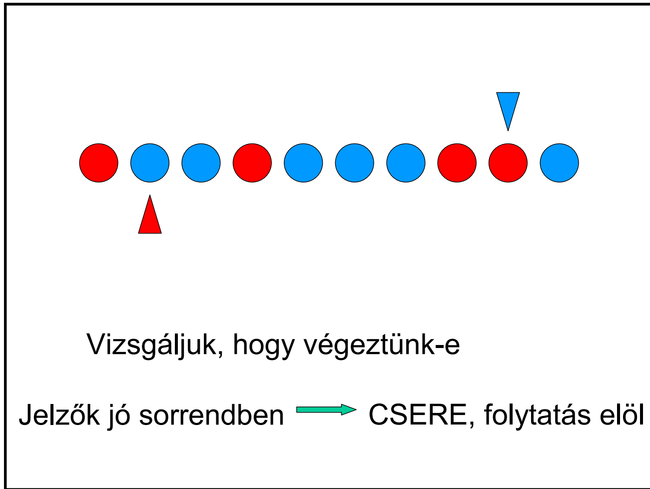
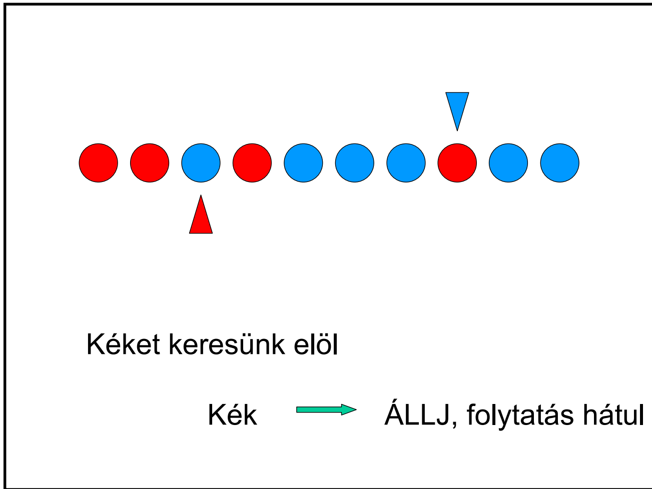
A vektor elemei közül vegyük előre azokat, amelyek rendelkeznek egy adott tulajdonsággal!
A vektor végén helyezzük el azokat az elemeket, amelyek nem rendelkeznek az adott tulajdonsággal.
A két csoporton belül a sorrendnek nincs jelentősége.

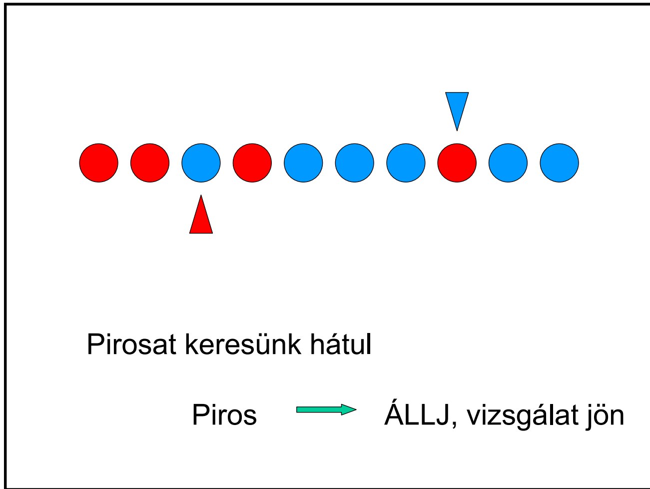
Egy lehetséges megoldás:
(ha biztosan van ilyen is, olyan is)

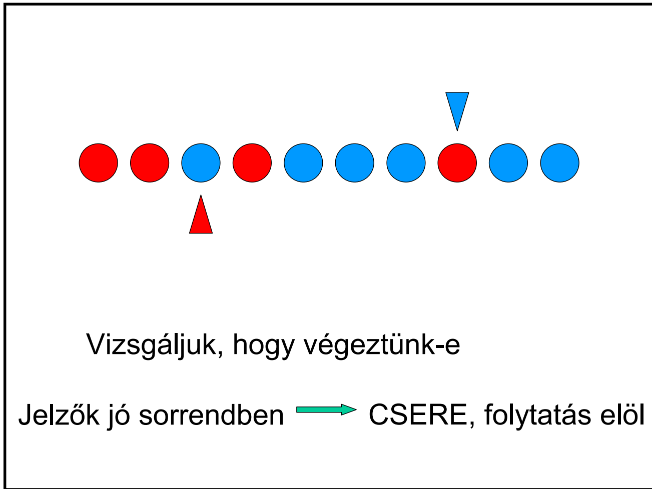
Induljunk el elölről, és keressük meg az első olyan elemet, amely hátulra való!

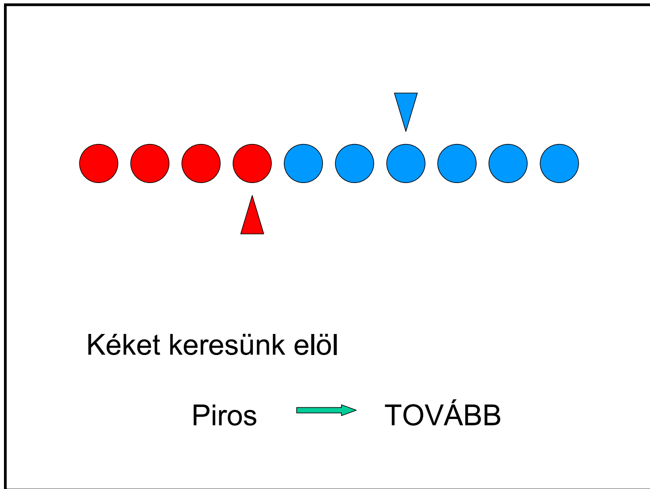
Induljunk el hátulról, és keressük meg az első olyan elemet, amely előre való!

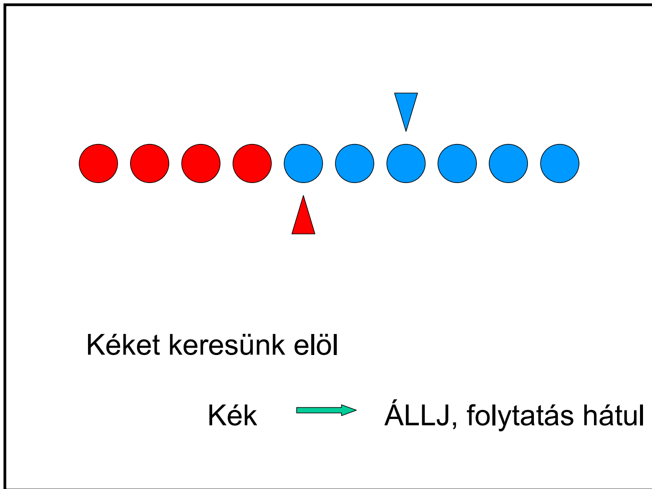
Ha még nem értünk a vizsgálat végére, cseréljük meg a két elemet, és folytassuk a keresést újra elölről hátra!

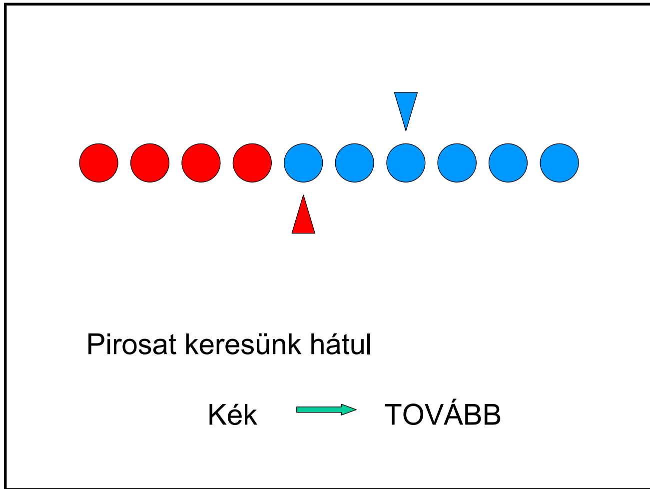

Kéket keresünk elöl
Piros
TOVÁBB

Kéket keresünk elöl
Kék
ÁLLJ, folytatás hátul

Pirosat keresünk hátul
Kék
TOVÁBB

Pirosat keresünk hátul
Piros
ÁLLJ, vizsgálat jön

Vizsgáljuk, hogy végeztünk-e
Jelzők jó sorrendben
CSERE, folytatás elöl

Kéket keresünk elöl
Kék
ÁLLJ, folytatás hátul

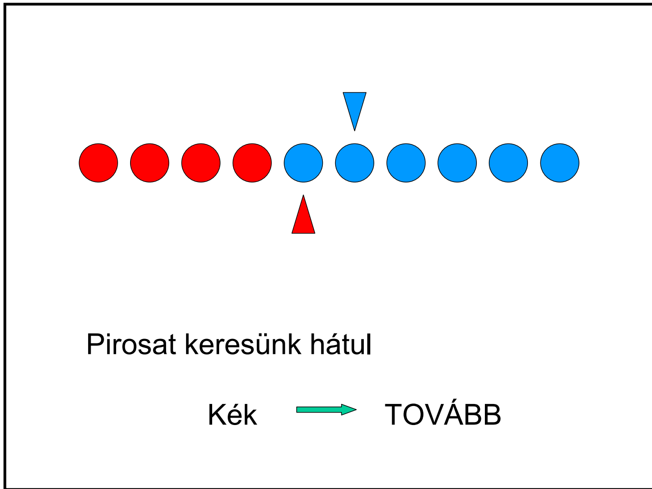

Pirosat keresünk hátul
Piros
ÁLLJ, vizsgálat jön

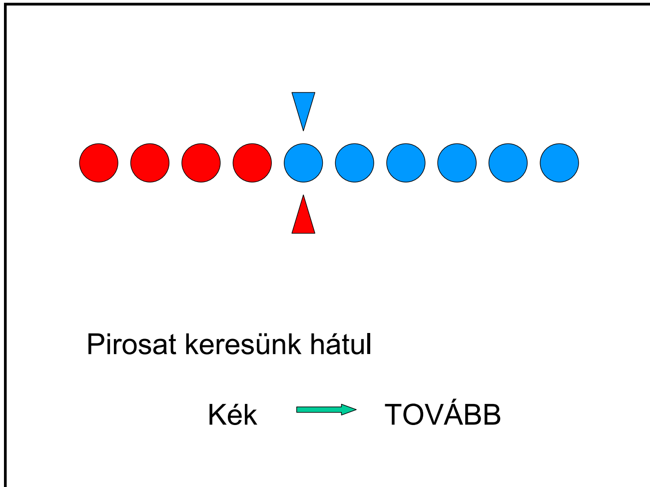

Vizsgáljuk, hogy végeztünk-e
Jelzők jó sorrendben
CSERE, folytatás elöl

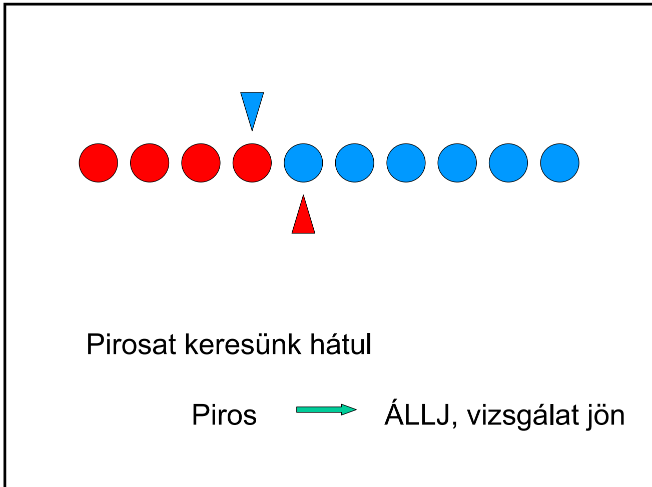

Kéket keresünk elöl
Piros
TOVÁBB

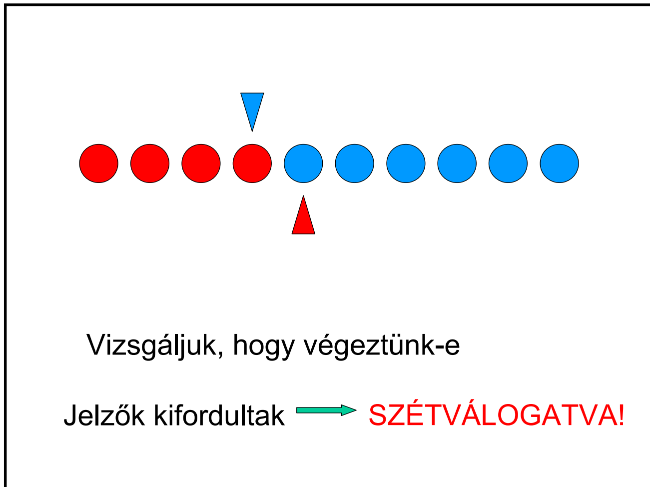

Kéket keresünk elöl
Kék
ÁLLJ, folytatás hátul


Pirosat keresünk hátul
Kék
TOVÁBB


Pirosat keresünk hátul
Kék
TOVÁBB



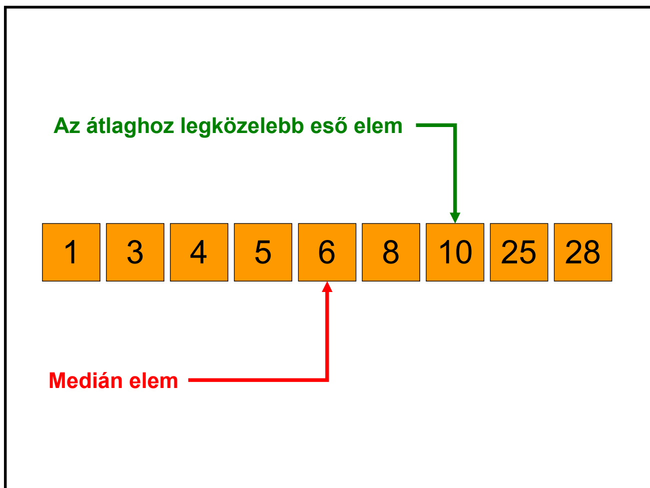




Hogyan lesz ebből gyorsrendezés?

A tulajdonság legyen az, hogy az elem kulcsa kisebb a mediánénál

(a medián középső elem)
(nem az átlag!)




Hogyan találhatjuk meg a medián elemet?
(a vektor végigolvasása nélkül)

Sajnos, SEHOGY!

## Hogyan becsülhetjük meg a medián elemet?

Minden jó lehet, aminél van kisebb, és van nagyobb kulcsú elem.

Mi csak a vektor két szélét látjuk.

Használjuk ezek átlagát!
Esetleg torzítsuk egy kicsit, hogy valóban vágjon.

## Hogyan áll le a rekurzió?

A rendezendő szakaszok hossza egyre csökken.

Ha szerencsénk van, minden lépésben feleződik.

Végül 1-elemű szakaszaink maradnak, ami mindig rendezett.

## Mennyi a várható lépésszám?

Minden finomításkor a teljes vektort végig kell olvasni, kivéve az egy hosszúságú szakaszokat.

Erre $\log_2 N$-szer van szükség.

Tehát a várható lépésszám $N \cdot \log_2 N$

K = 2

bináris fa

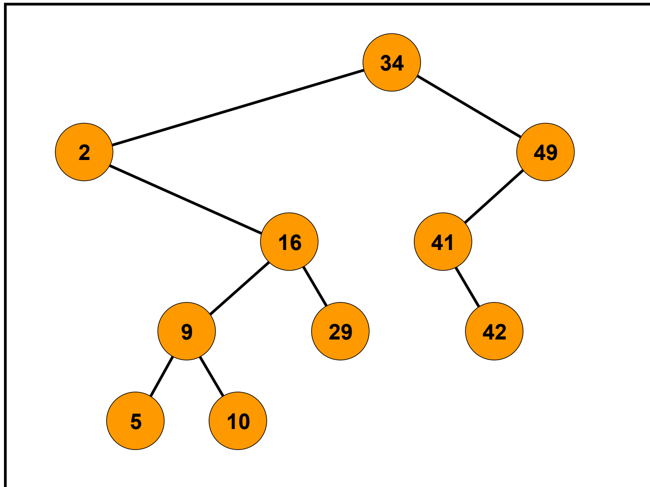
Bináris rendezőfa definíciója:

Olyan bináris fa, amelyben minden lokális gyökérelem baloldali részfájában csak a lokális gyökér kulcsánál kisebb kucsú elemek vannak.

A jobboldali részfában csak a lokális gyökér kulcsánál nagyobb kulcsú elemek vannak.

## bináris rendezőfa inorder bejárása

Alapgondolata:
Járjuk be a baloldali részfát, ha van!
Dolgozzuk fel a lokális gyökérelemet!
Járjuk be a jobboldali részfát, ha van!




Hogyan áll le a rekurzió?

A levélelemeknek nincsenek oldalsó részfái, sőt a csomópontok között is lehetnek olyanok, amelyeknek csak egy van.

Beszúrás:

Megkeressük, hol lenne az elem helye, és oda akasztjuk.

Törlés:

Az eddigi módszerek nem alkalmazhatók!

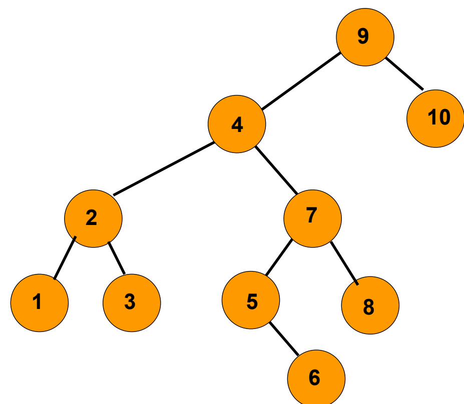
A kiiktatott elemet nem lehet kifűzni, ha két részfa lóg rajta.

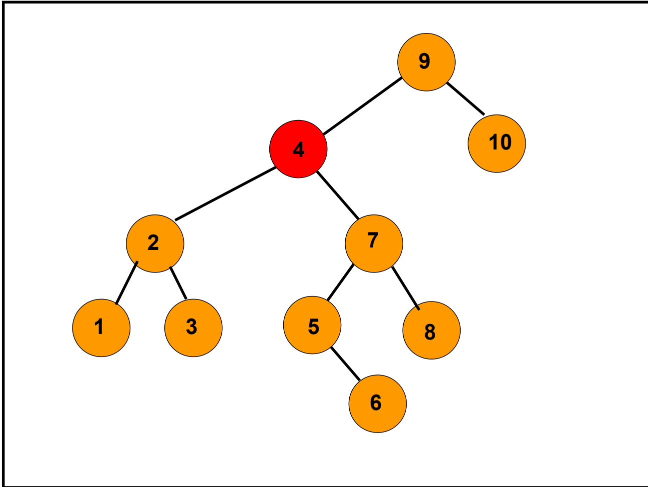
Ha csak az egyik oldali részfa létezik, a probléma nem jelentkezik.

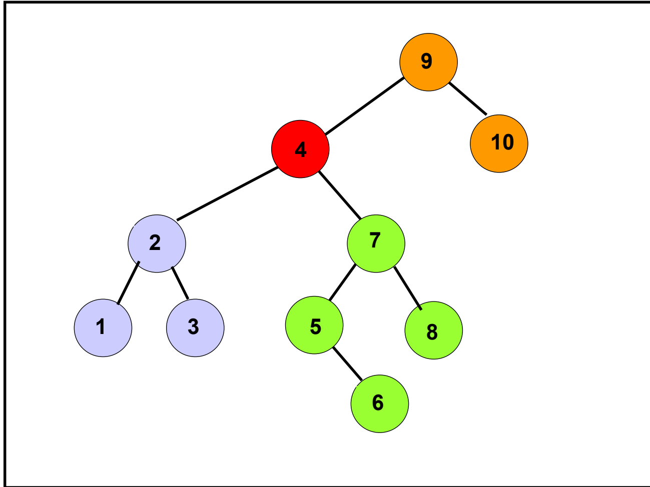
Megoldás a definíció alapján:

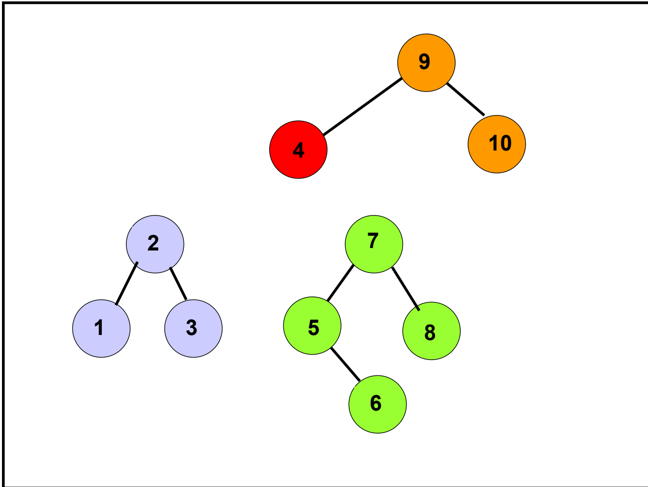
Az egyik (pl. a jobboldali) részfát gyökéreleménél fogva felvisszük a törölt elem helyére.

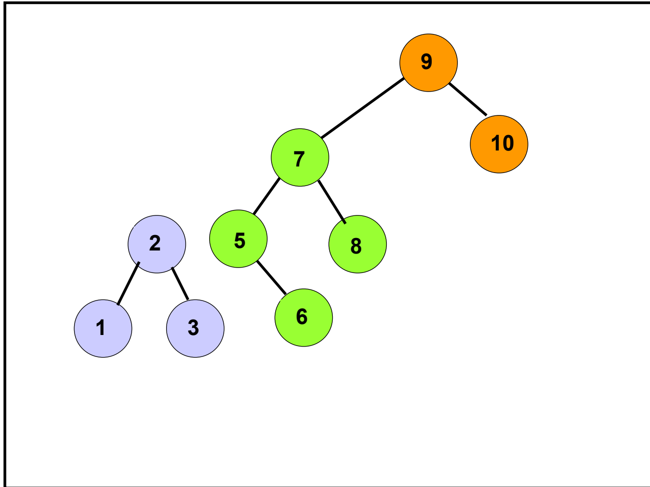
A másik (tehát baloldali) részfát felakasztjuk az előbbi részfa leg-baloldalibb elemének bal oldalára.

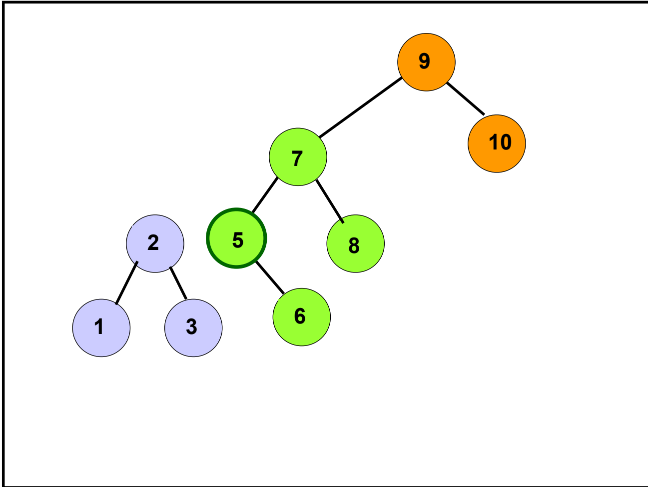


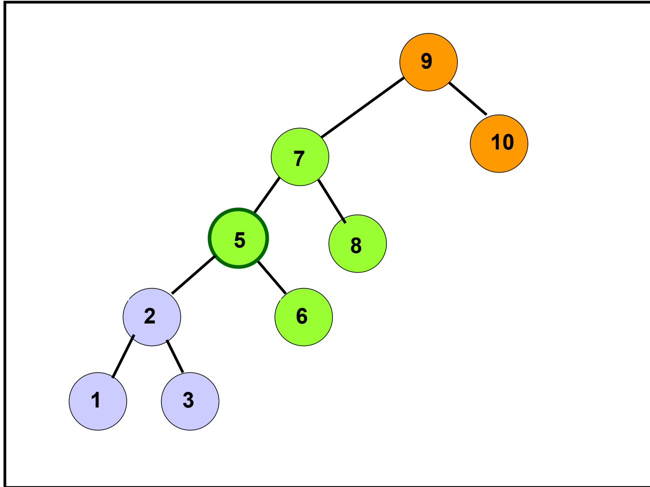

9
4
10
2
7
1
3
5
8
6

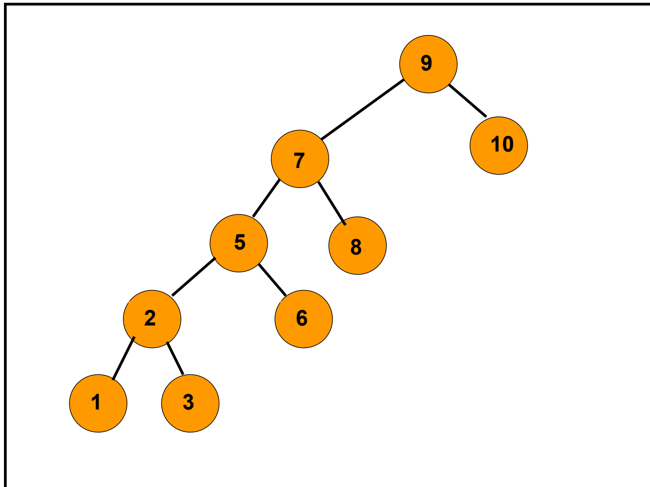

9
4
10
2
7
1
3
5
8
6


9
4
10
2
7
1
3
5
8
6


9
7
10
2
5
8
1
3
6


9
7
10
2
5
8
1
3
6


9
7
10
5
8
2
6
1
3




Bináris rendezőfa célja:

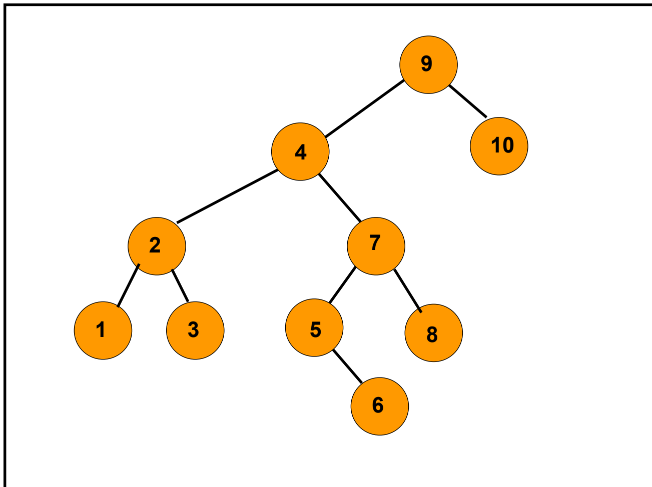
Gyors visszakeresést tesz lehetővé.

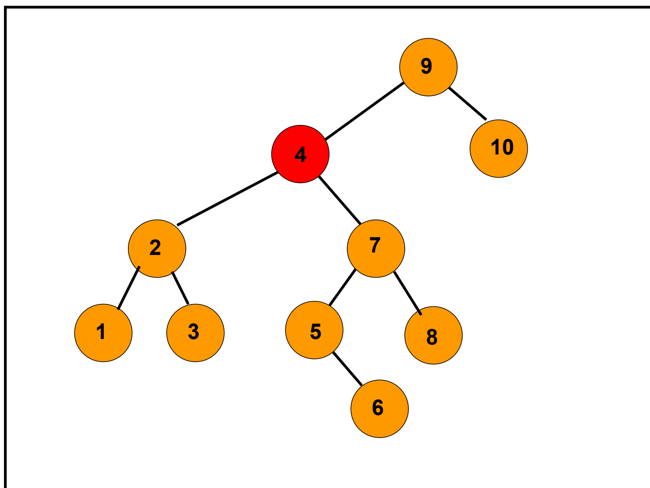
Logaritmikus keresés lehetséges, de csak ha a fa kiegyensúlyozott.

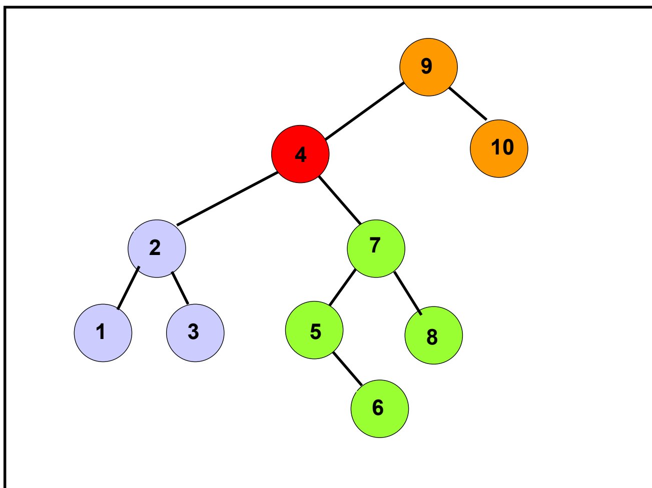
Elem törlése a kiegyensúlyozottságot kevésbé rontó algoritmussal:

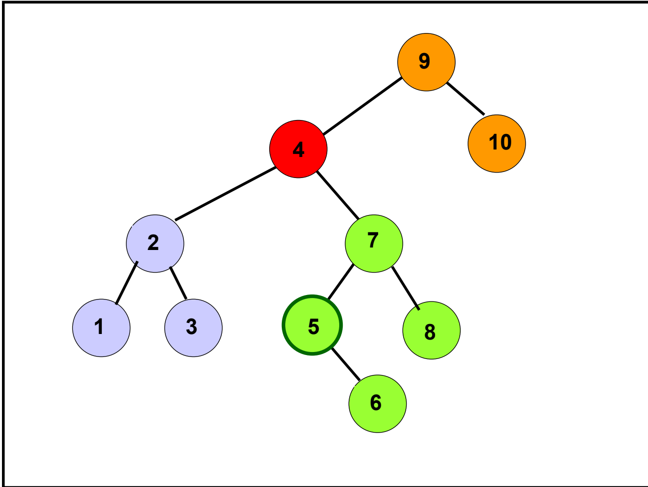
Az egyik, (például a jobboldali) részfa másik oldali legszélső elemét (tehát a legbaloldalibbat) írjuk be a törölni kívánt elem helyére.

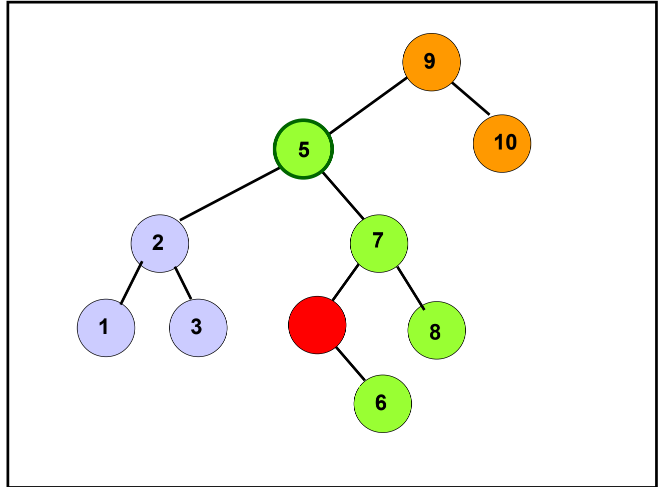
Ezután úgy kell eljárni, mintha ez utóbbi elemet akarnánk törölni, csakhogy ennek biztosan nincs legalább az egyik oldali részfája (esetünkben baloldali részfája), így rekurzió nem alakul ki.

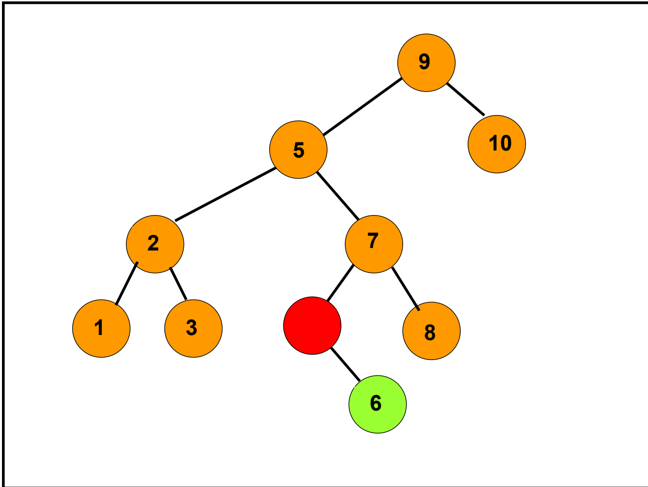


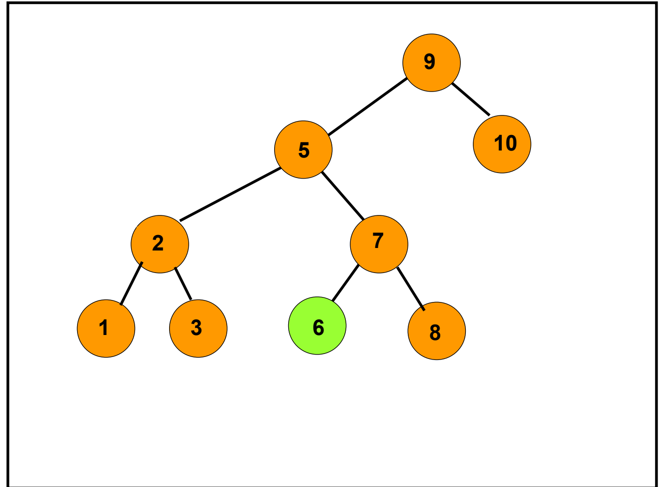



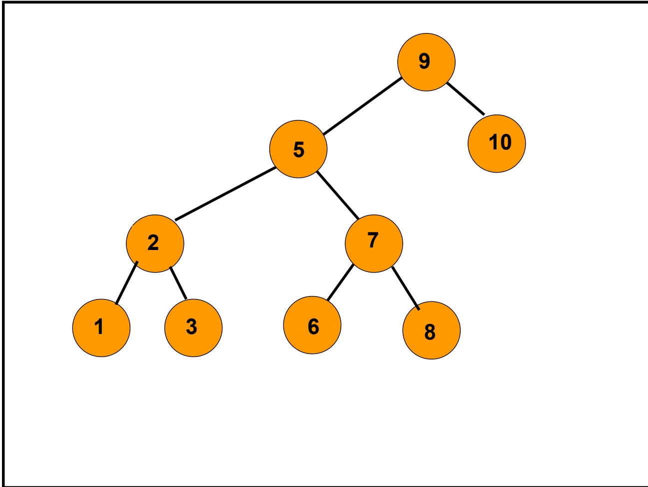



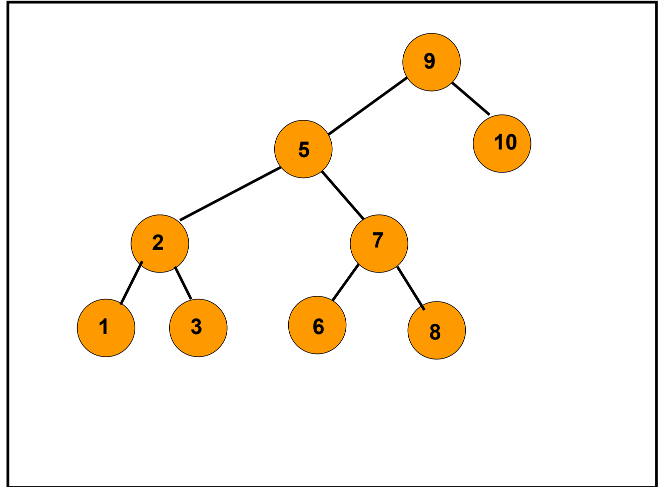

9
4
10
2
7
1
3
5
8
6


9
5
10
2
7
1
3
8
6


9
5
10
2
7
1
3
8
6


9
5
10
2
7
1
3
6
8


9
5
10
2
7
1
3
6
8


9
5
10
2
7
1
3
6
8