disclaimer: néhány vizsgakérdés kidolgozva hallgatók által, vagyis nem garantált hogy helyesek a megoldások

# 1. Függvények aszimptotikus viselkedése

1. Tegyük fel, hogy g : N → R, aszimptotikusan nemnegatív függvény!

# 1.a, Adjuk meg az O(g) és az Ω(g) függvényhalmazok denícióját!

* O(g) = {f : ∃d ∈ ℙ, hogy elég nagy n-ekre d ∗ g(n) ≥ f(n).} g aszimptotikus felső korlátja f-nek, ha f ∈ O(g).
* Ω(g) = {f : ∃c ∈ ℙ, hogy elég nagy n-ekre c ∗ g(n) ≤ f(n).} g aszimptotikus alsó korlátja f-nek, ha f ∈ Ω(g).

# 1.b, Milyen alapvető összefüggést ismer az O(g), az Ω(g) és a Θ(g) függvényhalmazok között?

A képen szöveg, Betűtípus, fehér, képernyőkép látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, képernyőkép látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, Grafika, embléma látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, fehér, tervezés látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, Grafika, fehér látható

Automatikusan generált leírás



A képen szöveg, Betűtípus, fehér, tipográfia látható

Automatikusan generált leírás

## 1.c, Igaz-e, hogy (3n + 4)^2 ∈ Θ(n^2 ) ? Miért?

---felbontás, megnézed az n hatványát—

---felülről becsülje—

n hatvany egyenlő e theta n-jével

## 1.d, Igaz-e, hogy n^n ∈ Ω(2^n ) ? Miért?

Igen, igaz, hogy n^n ∈ Ω(2^n). A n^n függvény exponenciálisan gyorsabban nő, mint a 2^n függvény. Tehát minden n ≥ 2 esetén teljesül, hogy n^n ≥ 2^n, vagyis a n^n függvény legalább olyan gyorsan nő, mint a 2^n függvény. Ezért n^n ∈ Ω(2^n).

--alulról becsülje-- ????

## 1.e, Igaz-e, hogy 1000n^2 (ln n)^2 ∈ O(n^3 ) ? Miért?

Igen, mert 1000n^2 (ln n) az lényegében n\*(log n), ami kisebb, mint O(n^3).

# 2.c, Igaz-e, hogy (2n + 1)(3n − 4) ∈ Θ(n^2 ) ? Miért?

Igen, mivel kiszámolva 6n^2 lesz a legnagyobb hatvány, ami lényegében n^2, tehát az pontosan Θ(n^2)

# 2.d, Igaz-e, hogy 2^n ∈ O(n!) ? Miért?

Igaz, hiszen O(n!) a lehetséges legnagyobb nagyságrendű aszimptotika, tehát a 2^n, ami lényegében n az kisebb.

# 2.e, Igaz-e, hogy (n ln n)^2 ∈ Θ(n^3 ) ? Miért?

Nem igaz, hiszen (n ln n) az lényegében n(log n), ami nem egyezik meg nagyságrendileg Θ(n^3).

# 2.1. Beszúró rendezés

## 1.a, Szemléltesse a beszúró rendezést az előadásról ismert módon az ⟨5; 7; 6; 4; 8; 4⟩ tömbre! Az utolsó beszúrást részletezze!

## 1.b, Adja meg a megfelelő struktogramot!

A képen szöveg, Betűtípus, szám, képernyőkép látható

Automatikusan generált leírás

## 1.c, Számolja ki a struktogramjának megfelelő pontos MT(n) és mT(n) értékeket!



A képen szöveg, Betűtípus, tipográfia, fehér látható

Automatikusan generált leírás

## 1.d, Mit jelentenek ezek az eredmények aszimptotikusan?

A képen szöveg, Betűtípus, fehér, tipográfia látható

Automatikusan generált leírás

## 2.a, Mikor nevezünk egy rendezést stabilnak?

**Stabilitás:** Egy rendező eljárást stabilnak nevezünk, ha nem változtatja meg az egyenlő kulcsú elemek egymáshoz viszonyított sorrendjét.

## 2.b, Adja meg fejelemes, egyirányú, nemciklikus listára (H1L) a beszúró rendezés struktogramját! A listát a key mezők szerint monoton növekvően kell rendezni. (A listaelemeknek a szokásos key és next mezőkön kívül más részei is lehetnek, de ezeket nem ismerjük.)

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

## 2.c, Hogyan biztosítottuk a fenti rendezés stabilitását?

A beszúró rendezés stabil! A rendezett szakaszba való beszúráskor a beszúrandó elem helyét jobbról balra keresi meg, és a vele egyenlőket már nem lépi át.

## 2.d, Mekkora lesz a rendezés minimális és maximális műveletigénye? Miért?

A képen szöveg, Betűtípus, kézírás, fehér látható

Automatikusan generált leírás(EA)

## 3.b, Adja meg fejelemes, kétirányú, ciklikus listára (C2L) a **beszúró rendezés** struktogramját! A listát a key mezők szerint monoton növekv®en kell rendezni. (A listaelemeknek a szokásos key és a két mutató mez®n kívül járulékos mez®i is lehetnek, de ezeket nem ismerjük. A listát kizárólag az unlink(q), precede(q, r) és follow(p, q) eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk.)

A képen szöveg, Betűtípus, szám, képernyőkép látható

Automatikusan generált leírás

## 3.c, Hogyan biztosítottuk a fenti rendezés stabilitását?

A beszúró rendezés stabil! A rendezett szakaszba való beszúráskor a beszúrandó elem helyét jobbról balra keresi meg, és a vele egyenlőket már nem lépi át.

## 3.d, Mekkora lesz a rendezés minimális és maximális műveletigénye? Miért?

A képen Betűtípus, szöveg, fehér, kézírás látható

Automatikusan generált leírás

# 2.2. Összefésülő rendezés

## 1.a, Szemléltesse az összefésülő rendezés (mergesort) működését az előadásról 3 ismert módon az ⟨4; 3; 5; 2; 1; 8; 3⟩ sorozatra! (Sem a szétvágásokat, sem az összefuttatásokat nem kell részletezni.)

## 1.b, Adja **meg egyszerű láncolt listákra** a **rekurzív eljárás** és a „cut” függvény struktogramját!

A képen szöveg, Betűtípus, képernyőkép, szám látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, sor, fehér látható

Automatikusan generált leírás

## 1.c, Mekkora a rendezés műveletigénye? Röviden indokolja állítását! (Csak a bizonyítás vázlatát kell leírni.)

A képen szöveg, Betűtípus, képernyőkép, Grafika látható

Automatikusan generált leírás

Bizonyítás:

* A műveletek túlnyomó részét a merge eljárás végzi el
* mTmerge(𝑙),MTmerge(𝑙) ∈ Θ(𝑙) (ahol l az aktuális résztömb hossza (l = v − u))
* A rekurzív ms(A, u, v) eljárás minden hívásban felezi a rendezendő résztömb hosszát (A rekurziónak kb. log n + 1 szintje van)
* Minden rekurziós szinten: a merge hívások résztömbjei együtt lefedik az egész A tömböt
* Kivétel az alsó egy vagy két szint: kevesebb (Egy tetszőleges szint összes merge hívásának műveletigényét összeadva: Θ(n) )
* A szintenkénti műveletigényt a szintek számával szorozva nagyságrendben Θ(n log n)

## 2.a, Adja meg **vektorokra** a mergeSort(A) és segédeljárásai struktogramjait!

## A képen szöveg, nyugta, Betűtípus, szám látható Automatikusan generált leírás

## 2.b, Mekkora lesz a műveletigénye és a tárigénye? Miért? (Csak vázlatos indoklást kérünk.)

A képen Betűtípus, tipográfia, kalligráfia, szöveg látható

Automatikusan generált leírás

Az összes elem összehasonlításával és az összes elem mozgatásával nő lineárisan a bemenet méretével.

## 3.a, Szemléltesse az összefésülő rendezés (merge sort) működését az előadásról ismert módon az ⟨5; 3; 2; 1; 4; 9; 2⟩ sorozatra!

## 3.b, Adja meg egyszerű láncolt listákra az összefésül® rendezés (merge sort) merge(L1, L2) függvényének struktogramját!

A képen szöveg, nyugta, diagram, Betűtípus látható

Automatikusan generált leírás

## 3.c, Mekkora a merge(L1, L2) függvény műveletigénye? Miért?

A képen szöveg, Betűtípus, képernyőkép, Kármin látható

Automatikusan generált leírás

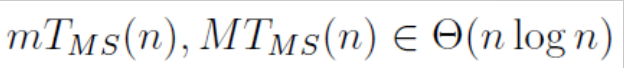
## 4.a, Szemléltesse az összefésülő rendezés (merge sort) működését az előadásról ismert módon az ⟨1; 9; 2; 5; 3; 2; 4⟩ sorozatra! (Sem a szétvágásokat, sem az összefuttatásokat nem kell részletezni.)

## 4.b, Adja meg egyszerű láncolt listákra az összefésülő rendezés (merge sort) fő eljárásának és rekurzív eljárásának struktogramját!

A képen szöveg, nyugta, diagram, Betűtípus látható

Automatikusan generált leírás

## 4.c, Mekkora a rendezés műveletigénye? Tárigénye?



5.a, Szemléltesse az összefésülő rendezés (mergesort) működését az előadásról ismert módon a ⟨4; 5; 2; 3; 5; 7; 3; 9; 4⟩ sorozatra! (Az utolsó összefésülésnél azt is jelezze, hogy az input elemei milyen sorrendben kerülnek az outputra!)

## 5.b, Egyszerű láncolt listákra a fenti rendezést úgy szeretnénk módosítani, hogy az azonos kulcsú elemekből csak egy-egy legyen az eredmény listán, így az szigorúan monoton növekvő legyen. A mergesort melyik eljárását/függvényét kell módosítanunk? Adja meg ennek az új struktogramját!

## 5.c, Mit tud mondani az újfajta összefésül® rendezés műveletigényéről és tárigényéről?

## 6.a, Szemléltesse az összefésülő rendezést az előadásról ismert módon a ⟨7; 5; 1; 3; 8; 6; 3; 9; 4⟩ sorozatra! (Az utolsó összefésülésnél azt is jelezze, hogy az input elemei milyen sorrendben kerülnek az outputra!)

## 6.b, Adja meg egyszerű láncolt listákra (S1L) az összefésülő rendezésből (merge sort) a rekurzív eljárás struktogramját!

A képen szöveg, nyugta, diagram, Betűtípus látható

Automatikusan generált leírás

## 6.c, Tegyük fel, hogy a rendezendő lista hossza n. Hányszor fog meghívódni a rendezés során a rekurzív eljárás? Miért? 🡪 Ricsi jegyzet

# 2.3. Kupacrendezés

## 1.a, Egy bináris fa mikor szigorúan bináris? Mikor teljes? Mikor majdnem teljes? Ez utóbbi mikor balra tömörített, és mikor kupac?

* **Szigorú bináris fa:** Minden belső csúcsnak két gyereke van.
* **Teljes bináris fa:** Szigorú bináris fa + Ha minden levele azonos szinten van.
* **Majdnem teljes bináris fa:** Ha egy teljes bináris fa levélszínijéről 0 vagy több levelet elveszünk, de nem az összeset. (üres fa, teljes bináris fa is az)
* **Majdnem teljes bináris fa balra tömörített:** Ha az alsó szintjén egyetlen levéltől balra sem lehet új levelet beszúrni.
* Maximum-**kupacnak:** Ha minden belső csúcs kulcsa >=, mint a gyerekeié.

## 1.b, Szemléltesse a kupacrendezést a következő tömbre! ⟨3; 9; 8; 2; 4; 6; 7; 5⟩ Minden lesüllyesztés előtt jelölje a csúcs mellett egy kis körbe tett sorszámmal, hogy ez a rendezés során a hányadik lesüllyesztés; akkor is, ha az aktuális lesüllyesztés nem mozdítja el a csúcsban lévő kulcsot! Minden valódi lesüllyesztés előtt jelölje a lesüllyesztés irányát és útvonalát! Minden olyan lesüllyesztés előtt rajzolja újra a fát, ami az aktuális ábrán már módosított csúcsokat érinti! Újra rajzoláskor adja meg a fát tartalmazó tömb pillanatnyi állapotát is!

## 2.a, Egy bináris fa mikor szigorúan bináris? Mikor teljes? Mikor majdnem teljes? Ez utóbbi mikor balra tömörített, és mikor kupac?

* **Szigorú bináris fa:** Minden belső csúcsnak két gyereke van.
* **Teljes bináris fa:** Szigorú bináris fa + Ha minden levele azonos szinten van.
* **Majdnem teljes bináris fa:** Ha egy teljes bináris fa levélszínijéről 0 vagy több levelet elveszünk, de nem az összeset. (üres fa, teljes bináris fa is az)
* **Majdnem teljes bináris fa balra tömörített:** Ha az alsó szintjén egyetlen levéltől balra sem lehet új levelet beszúrni.
* Maximum-**kupacnak:** Ha minden belső csúcs kulcsa >=, mint a gyerekeié.

## 2.b, Szemléltesse a kupacrendezést a következő tömbre! ⟨5; 7; 6; 4; 2; 8; 9; 4; 3⟩ Minden lesüllyesztés elött jelölje a csúcs mellett egy kis körbe tett sorszámmal, hogy ez a rendezés során a hányadik lesüllyesztés; akkor is, ha az aktuális lesüllyesztés nem mozdítja el a csúcsban lévő kulcsot! Minden valódi lesüllyesztés elött jelölje a lesüllyesztés irányát és útvonalát! Minden olyan lesüllyesztés előtt rajzolja újra a fát, ami az aktuális ábrán már módosított csúcsokat érinti! Újra rajzoláskor adja meg a fát tartalmazó tömb pillanatnyi állapotát is!

## 3.a, Adja meg a heapSort(A : T []) és segédeljárásai struktogramjait!

A képen szöveg, Betűtípus, képernyőkép, szám látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, képernyőkép, szám látható

Automatikusan generált leírás

## 3.b, Igaz-e, hogy MT(n) ∈ Θ(n lg n)? Miért? [Vegyük észre, hogy az indokláshoz elegendő, ha a kupaccá alakítás és az utána következő rész műveletigényére is durva felső becslést adunk, továbbá használjuk az összehasonlító rendezések (alsókorlát-elemzésre vonatkozó) alaptételét!]

2.4. Gyorsrendezés

## 1.a, Írja le az előadásról ismert formában a gyorsrendezés (quicksort) struktogramjait!

A képen szöveg, Betűtípus, képernyőkép, szám látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, nyugta, képernyőkép látható

Automatikusan generált leírás

## 1.b, Szemléltesse a program „partition” függvényének működését a következő vektorra! ⟨3; 4; 8; 7; 1; 2; 6; 4⟩.

## 1.c, Mekkora a gyorsrendezés műveletigénye?

A képen Betűtípus, szöveg, fehér, tipográfia látható

Automatikusan generált leírás

## 1.d, Érdemes-e a gyorsrendezést és a beszúró rendezést egyetlen rendezésben egyesíteni? Hogyan? Miért? **(ChatGPT)**

* **Előnyei:** stabilitás, egyszerűség
* **Hogyan, miért:** Gyorsrendezést alkalmazzuk az eredeti tömbre, majd azokat a résztömböket, amelyek a gyorsrendezés során kisebbek lettek egy bizonyos mérettől, beszúró rendezéssel rendezzük.

## 2.a, Írjuk le az előadásról ismert formában a gyorsrendezés (quicksort) struktogramjait!

A képen szöveg, Betűtípus, képernyőkép, szám látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, nyugta, képernyőkép látható

Automatikusan generált leírás

## 2.b, Szemléltessük a program „partition” függvényének működését a következő vektorra! ⟨1; 2; 8; 7; 3; 2; 6; 3⟩.

## 2.c, Mekkora a gyorsrendezés műveletigénye?

A képen Betűtípus, szöveg, fehér, tipográfia látható

Automatikusan generált leírás

## 2.d, Mekkora munkatárat igényel a gyorsrendezés (quicksort) alapváltozata a legjobb és a legrosszabb esetben? Miért? 🡪idoigeny

## 3. Tervezze újra a quicksort eljárást egyszerű láncolt listákra! Adja meg a szükséges struktogramokat, ha a szétvágásnál a rendezendő lista első eleme a tengely (pivot)! Vegye figyelembe, hogy a listaelemekben a key mezőn és a következő listaelemre mutató mezőn kívül lehetnek számunkra ismeretlen, járulékos mezők is!

**Algo kérdések kidolgozva(?)**

# 3. Absztrakt adattípusok

(a pirossal bekeretezett részek a kérdés válasza)

# 3.1 Verem

1. Adja meg az előadásról ismert módon a Stack osztály - tömbös reprezentációra alapozott - leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje? A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

A képen szöveg, képernyőkép, Betűtípus, diagram látható

Automatikusan generált leírás

**Műveletigénye:**



## 2. A TransparentStack adattípus műveletei a szokásos jelentéssel és formában a push, pop, isEmpty verem műveletek, de a v.push(x) csak akkor teszi x-et v-be, ha éppen nincs benne (ha benne van, nem csinál semmit). A TransparentStack elemtípusa az 1..n egész intervallum típus, és így legfeljebb n méretű vermekre van szükségünk, ahol n pozitív egész szám**. Adja meg a fenti típust megvalósító osztályt, a metódusokkal együtt, ahol n a konstruktor paramétere!** A konstruktor műveletigénye Θ(n), a többi metódus és a destruktor műveletigénye Θ(1) legyen! {Használhat egy b[1..n] logikai tömböt ,,b[x] = true ⇐⇒ x a veremben van” jelentéssel, ahol x ∈ 1..n.}

**(NINCS KÉSZ MERT ITT STUKIT KELL IRNI!!!!!!!!!!!!!!!)**

## 3. Adja meg az előadásról ismert Stack osztály - egyszerű láncolt listás reprezentációra alapozott - leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje?

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás**A képen szöveg, diagram, Tervrajz, Párhuzamos látható

Automatikusan generált leírás**

**Műveletigénye:**



# 3.2 Sor

## 1. Adja meg az előadásról ismert módon a Queue osztály - tömbös reprezent

## ációra alapozott - leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?

**A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás**

**A képen szöveg, képernyőkép, nyugta, Betűtípus látható

Automatikusan generált leírás**

**Műveletigénye:**

****

## 2. Adja meg az előadásról ismert Queue osztály - egyirányú, nem ciklikus, láncolt listás reprezentációra (S1L) alapozott - leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! (Szüksége lesz arra, hogy a lista elejét és végét is közvetlenül elérje.) Mekkora az egyes műveletek aszimptotikus futási ideje?

**A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás**

**A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírásA képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás**

****

## 3. Adja meg az előadásról ismert Queue osztály - egyirányú, ciklikus, fejelemes láncolt listás reprezentációra alapozott - leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! A listát csak a fejelemre mutató pointeren keresztül szabad elérni. Mekkora az egyes műveletek aszimptotikus futási ideje?

A képen szöveg, képernyőkép, Betűtípus, nyugta látható

Automatikusan generált leírásA képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírásA képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírásA képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

****

# 3.3 Prioritásos sor

## 1. Tegyük fel, hogy a prioritásos sort tömbben, maximum kupaccal reprezentáljuk! Adja meg az előadásról ismert módon a PrQueue osztály leírását, a metódusok struktogramjaival együtt! (A lesüllyesztést nem kell leírni.) Mekkora az egyes műveletek aszimptotikus futási ideje?

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás



A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás



A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás



A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

## 2.a, Egy bináris fa mikor szigorúan bináris? Mikor teljes? Mikor majdnem teljes? Ez utóbbi mikor balra tömörített, és mikor kupac?

* **Szigorú bináris fa:** Minden belső csúcsnak két gyereke van.
* **Teljes bináris fa:** Szigorú bináris fa + Ha minden levele azonos szinten van.
* **Majdnem teljes bináris fa:** Ha egy teljes bináris fa levélszínijéről 0 vagy több levelet elveszünk, de nem az összeset. (üres fa, teljes bináris fa is az)
* **Majdnem teljes bináris fa balra tömörített:** Ha az alsó szintjén egyetlen levéltől balra sem lehet új levelet beszúrni.
* Maximum-**kupacnak:** Ha minden belső csúcs kulcsa >=, mint a gyerekeié.

## 2.b, Szemléltesse az alábbi kupacra a 9, majd az eredmény kupacra a 8 beszúrásának műveletét! ⟨8; 8; 6; 6; 5; 2; 3; 1; 5; 4⟩.

A képen vázlat, rajz, kör, clipart látható

Automatikusan generált leírás

A képen vázlat, rajz, Vonalas grafika, clipart látható

Automatikusan generált leírás

A képen vázlat, rajz, Vonalas grafika, rajzfilm látható

Automatikusan generált leírás

## 2.c, Szemléltesse az eredeti kupacra a maxKivesz eljárás kétszeri végrehajtását! Minden művelet után rajzolja újra a fát!

A képen bicikli, kerék, diagram látható

Automatikusan generált leírás

A képen bicikli, rajz, vázlat, kerék látható

Automatikusan generált leírás

A képen kör, vázlat, diagram, rajz látható

Automatikusan generált leírás

## 3. Tegyük fel, hogy a prioritásos sort tömbben, rendezetlen módon reprezentáljuk! Adja meg az előadásról ismert módon a PrQueue osztály leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, képernyőkép, szám látható

Automatikusan generált leírás

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

PrQueueue:: max(): T

**return** Z[maxInd]

## 4. Tegyük fel, hogy a prioritásos sort tömbben, monoton növekvően **rendezett sorozatként** tároljuk! Adja meg az előadásról ismert módon a PrQueue osztály leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, szám, képernyőkép látható

Automatikusan generált leírás

PrQueueue:: max(): T

**return** Z[n], n--

## 5. Tegyük fel, hogy a prioritásos sort fejelemes láncolt listával, rendezetlen módon reprezentáljuk! Adja meg az előadásról ismert PrQueue osztály leírását erre az esetre, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

**(NINCS KÉSZ MERT NEM TALÁLOM SEHOL EZT!!!!!!!!!!!!!!!)**

## 6. Tegyük fel, hogy a prioritásos sort fejelemes láncolt listával, monoton csökkenően rendezett sorozatként tároljuk! Adja meg az előadásról ismert PrQueue osztály leírását erre az esetre, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

**(NINCS KÉSZ MERT NEM TALÁLOM SEHOL EZT!!!!!!!!!!!!!!!)**

# 4. Láncolt listák

# 4.1. Egyszerű listák (S1L)

## 1. Az L1 és L2 pointerek két egyszerű láncolt listát azonosítanak. Írja meg az append(L1, L2) eljárást, ami MT(n) ∈ Θ(n) és mT(n) ∈ Θ(1) (n = |L1|) műveletigénnyel az L1 lista után fűzi az L2 listát!

## 2. Írja meg a reverse(L) eljárást, ami megfordítja az L egyszerű láncolt lista elemeinek sorrendjét! T(n) ∈ Θ(n), ahol n a lista hossza.

A képen szöveg, nyugta, Betűtípus, szám látható

Automatikusan generált leírás

4.2. Fejelemes listák (H1L)

## 1. Az L1, L2 pointerek egy-egy szigorúan monoton növekvő H1L (fejelemes, egyirányú, nem ciklikus, láncolt lista) fejelemére mutatnak. Írjuk meg az **intersection(L1, L2)** eljárást, ami az L1 lista elemei közül törli azokat az elemeket, amelyek kulcsa nem szerepel az L2 listán! Így az L1 listán a két input lista metszete jön létre, míg az L2 lista változatlan marad. Mindkét listán legfeljebb egyszer menjünk végig! Listaelemeket ne hozzunk létre! MT(n1, n2) ∈ O(n1 + n2) és mT(n1, n2) ∈ O(n1) legyen, ahol n1 az L1, n2 az L2 lista hossza.

## 2. Az L pointer egy nemüres, fejelemes láncolt lista fejelemére mutat. Írjuk meg a **minElejére(L)** eljárást, ami az L lista legkisebb kulcsú elemét átfűzi az L lista elejére! A program a listán csak egyszer menjen végig! T(n) ∈ Θ(n), ahol n az L lista hossza. A listaelemeknek a key és a next mezőkön kívül más részei is lehetnek, de ezeket nem ismerjük.

4.3. Egyszerű kétirányú listák (S2L)

## 1. Az L pointer egy rendezetlen, kétirányú, fejelem nélküli, nem ciklikus, láncolt listát azonosít. Írjuk meg a **del(L, k)** eljárást, ami az L listából törli a k kulcsú listaelemeket! A listán legfeljebb egyszer menjünk végig! Listaelemet ne hozzunk létre, a feleslegessé váló listaelemeket viszont deallokáljuk! Mekkora a fenti eljárás műveletigénye? Miért?

## 2. Az L pointer egy monoton növekvö kétirányú, fejelem nélküli, nem ciklikus, láncolt listát azonosít. Írjuk meg a **sortedInsert(L, k)** eljárást, ami az L listába rendezetten beszúr egy k kulcsú listaelemet! A listán legfeljebb egyszer menjünk végig! Pontosan egy listaelemet hozzunk létre! MT(n) ∈ Θ(n) és mT(n) ∈ O(1) legyen, ahol n az L lista hossza.

4.4. Fejelemes, kétirányú, ciklikus listák (C2L)

## 1. Az L1, L2 pointerek egy-egy szigorúan monoton növekvő C2L (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listákat kizárólag az unlink(q), precede(q, r) és follow(p, q) eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk. **Írjuk meg** a **difference(L1, L2)** eljárást, ami az L1 lista elemei közül törli az L2 listán is szereplő elemeket! Az L2 lista változatlan, de az L1 is szigorúan monoton növekvő® marad. Mindkét listán legfeljebb egyszer menjünk végig! A felszabaduló listaelemeket adjuk vissza a szabad területnek! MT(n1, n2) ∈ O(n1 + n2) és mT(n1, n2) ∈ O(min(n1, n2)) legyen, ahol n1 az L1, n2 az L2 lista hossza.

## 2. Az L1, L2 pointerek egy-egy szigorúan monoton növekvő® C2L (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listákat kizárólag az unlink(q), precede(q, r) és follow(p, q) eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk. **Írjuk meg** a **unionIntersection(L1, L2)** eljárást, ami az L1 lista elemei közé átfűzi az L2 listáról az L1 listán eredetileg nem szereplő elemeket! Így az L1 listán a két input lista uniója, míg az L2 listán a metszetük jön létre, és mindkét lista szigorúan monoton növekvő marad. Mindkét listán legfeljebb egyszer menjünk végig! Listaelemeket ne hozzunk létre és ne is töröljünk! MT(n1, n2) ∈ O(n1 + n2) és mT(n1, n2) ∈ O(n2) legyen, ahol n1 az L1, n2 az L2 lista hossza.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

5. Bináris fák

## 1.a A t : Node\* típusú pointer egy láncoltan ábrázolt bináris fát azonosít. A fa csúcsaiban nincsenek „parent” pointerek. Írjuk meg a **töröl(t)** rekurzív eljárást, ami törli a t fa csúcsait, Θ(|t|) műveletigénnyel, a posztorder bejárás szerint! Rendelkezésre áll ehhez a delete p utasítás, ami a p pointer által mutatott csúcsot törli. A t fa végül legyen üres!

## 1.b A t1, t2 : Node\* típusú pointerek egy-egy láncoltan ábrázolt bináris fát azonosítanak. A fák csúcsaiban nincsenek parent pointerek. A t2 fa akkor fedi le a t1 fát, ha t1 üres, vagy ha egyikük sem üres, és a t1 bal és jobb részfáját is lefedik a t2 megfelel® oldali részfái. Egy fa megmetszése alatt bizonyos részfái törlését értjük. Írjuk meg az **alámetsz(t1, t2)** rekurzív eljárást, ami úgy metszi meg a t1 fát, hogy a t2 fa lefedje, de a t1 fa a lehet® legnagyobb maradjon! A t1 feleslegessé váló részfáit töröljük a töröl(t) eljárás segítségével! T(n) ∈ Θ(n) legyen, ahol n a t1 fa mérete.

## 2. A t : Node\* típusú pointer egy láncoltan ábrázolt bináris fát azonosít, ami majdnem teljes és balra tömörített. A fa csúcsaiban nincsenek parent pointerek. Írja meg az **szkiír(t, A, n)** eljárást, ami Θ(|t|) m¶veletigénnyel a fa kulcsait szintfolytonosan az A tömbbe írja, és n-ben visszaadja a fa méretét! Feltehet®, hogy a tömb elég nagy. Az eljárás a fát ne változtassa meg!

## 3. A t : Node\* típusú pointer egy láncoltan ábrázolt bináris fát azonosít. A fa csúcsaiban nincsenek parent pointerek. Írja meg az **szkiír(t)** eljárást, ami Θ(|t|) m¶veletigénnyel szintenként írja ki a fa kulcsait úgy, hogy minden szint külön sorba kerül, azaz az els® sorban csak a fa gyökércsúcsának kulcsa jelenik meg, a második sorban a gyerekei kulcsai, a harmadikban az unokáié stb. Mindegyik szintet balról jobbra írjuk ki! Az eljárás a fát ne változtassa meg!

## 

## 4. Milyen bináris fa bejárásokat ismer?

inorder, preorder, postorder

## 4.a, Adja meg a struktogramjaikat!

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

## 4.b, Számolja ki a struktogramokhoz tartozó műveletigényeket!

A képen szöveg, Betűtípus, fehér, kalligráfia látható

Automatikusan generált leírás

5.1. Bináris keresőfák (és rendezőfák)

## 1.a, A bináris fa fogalmát ismertnek feltételezve, mondjuk ki a bináris keresőfa definícióját!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 1.b, A t egy láncoltan ábrázolt bináris keresőfa gyökérpointere. A csúcsokban nincsenek parent pointerek. (A fa üres is lehet.) Írja meg az előadásról ismert módon az **insert(t, k)** eljárás rekurzív struktogramját, ami megkeresi a t fában a k kulcs helyét, és ha ott egy üres részfát talál, akkor az üres részfa helyére tesz egy új levélcsúcsot, k kulccsal.

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás

## 1.c, Mekkora az insert(t, k) eljárás maximális, illetve minimális műveletigénye? Miért?

## 2.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 2.b, Írja meg az **insert(t, k, s)** ciklust nem tartalmazó MT(h) ∈ Θ(h) hatékonyságú rekurzív eljárást (h = h(t)), ami megpróbál beszúrni a t bináris keresőfába egy k kulcsú csúcsot (akkor tudja beszúrni, ha a fában nem talál ilyet), és az s, logikai típusú paraméterben visszaadja, hogy sikeres volt-e a beszúrás! A fa csúcsai Node típusúak; szülő pointert nem tartalmaznak.

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás

## 2.c, Igaz-e, hogy a fenti insert(t, k, s) eljárásra mT(h) ∈ Θ(1)? Miért?

## 3.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, de- niálja a bináris keres®fa fogalmát!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 3.b, Szemléltesse a remMin(t, minp) eljárás hatását az alábbi bináris keresőfán! { [ (1 {2}) 3 (4) ] 5 [ (6) 7 (8) ] }

## 3.c, Írja meg a remMin(t, minp) és a remMax(t, maxp) , MT(h(t)) ∈ O(h(t)) hatékonyságú rekurzív eljárásokat, ahol t ̸= bináris keres®fa! (A remMax(t, maxp) eljáráshívás a maxp pointert a t fa a legnagyobb kulcsú csúcsára állítja, és ezt a csúcsot el is távolítja a fából, ami továbbra is bináris keres®fa marad.) A fa csúcsai parentpointert nem tartalmaznak, számunkra ismeretlen, járulékos adatmez®ket viszont tartalmazhatnak.

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás

## 3.d, Igaz-e a fenti remMin(t, minp) és remMax(t, maxp) eljárásokra, hogy mT(h(t)) ∈ Θ(1)? Miért?

A képen szöveg, Betűtípus, fehér, tipográfia látható

Automatikusan generált leírás

## 4.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, de- niálja a bináris keres®fa fogalmát!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 4.b, Szemléltesse a delRoot(t) eljárás hatását az alábbi bináris keres®fán! (Törléskor, indeterminisztikus esetben a jobb részfa minimumát használjuk!) { [ (1) 2 (3) ] 4 [ (5 {7}) 8 (9) ] } A remMin(t, minp) eljárás struktogramját nem kell megadni.

## 4.c, Írja meg a **delRoot(t)**, MT(h(t)) ∈ O(h(t)) hatékonyságú eljárást (ami a nemüres t bináris keresőfából törli a gyökércsúcsot)! A fa csúcsai „parent” pointert nem tartalmaznak, számunkra ismeretlen, járulékos adatmezőket viszont tartalmazhatnak.

A képen szöveg, Betűtípus, képernyőkép, szám látható

Automatikusan generált leírás

## 4.d, Igaz-e a fenti delRoot(t) eljárásra, hogy mT(h(t)) ∈ Θ(1)? Miért?

A képen szöveg, Betűtípus, fehér, tipográfia látható

Automatikusan generált leírás

## 5.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 5.b, Szemléltesse a delRoot(t) eljárás hatását az alábbi bináris keresőfán! (Törléskor, indeterminisztikus esetben a jobb részfa minimumát használjuk!) { [ (2) 3 ] 4 [ (5) 6 ({7} 8 ) ] } A delRoot(t) eljárás struktogramját nem kell megadni.

## 5.c, Írja meg a del(t, k), MT(h(t)) ∈ O(h(t)) hatékonyságú rekurzív logikai függvényt, ami megpróbálja törölni a láncoltan ábrázolt t bináris keres®fából a k kulcsú csúcsot! A del(t, k) függvény adja vissza, hogy sikeres volt-e a törlés! A fa csúcsai parentpointert nem tartalmaznak, számunkra ismeretlen, járulékos adatmez®ket viszont tartalmazhatnak.

A képen szöveg, Betűtípus, sor, képernyőkép látható

Automatikusan generált leírás

## 5.d, Igaz-e a fenti del(t, k) függvényre, hogy mT(h(t)) ∈ Θ(1)? Miért?

## 6.a, A bináris fa fogalmát ismertnek feltételezve, deniálja a bináris keres®fa fogalmát!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 6.b, Adott a t bináris fa. A csúcsok kulcsai pozitív egész számok. Írja meg a bst(t) logikai függvényt; ami a t egyszeri (Inorder) bejárásával eldönti, hogy keres®fa-e! MT(n) ∈ O(n), ahol n = |t|. MS(h) ∈ O(h), ahol h = h(t); MS pedig az algoritmus maximális munkatár-igényét jelöli. (A bejárást és eldöntést a megfelel®en inicializált, rekurzív, bst(t, k) logikai segédfüggvény végezze, ami híváskor k-ban a t kulcsainál kisebb értéket vár, visszatéréskor pedig, amennyiben t nemüres keres®fa, a t-beli legnagyobb kulcsot tartalmazza! Ha t üres, akkor k-ban maradjon a függvényhívásnál kapott érték!)

## 6.c, Igaz-e az Ön által megfogalmazott bst(t) logikai függvényre, hogy mT(h) ∈ O(h)? Miért?

## 7.a, Bizonyítsa be, hogy tetszőleges, n csúcsú és h magasságú bináris fára az n − 1 ≥ h egyenlőtlenség teljesül!

## 7.b, Mikor lesz h = n − 1, és miért?

## 7.c, Bizonyítsa be, hogy h ≥ ⌊lg n⌋, ha a bináris fa nemüres!

## 7.d, Bizonyítsa be, hogy h = ⌊lg n⌋, ha az előbbi fa majdnem teljes!

## 7.e, Lehetséges-e, hogy h = ⌊lg n⌋, holott a nemüres bináris fára a majdnem teljesség kritériuma nem teljesül? Miért?

## 8.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, de- niálja a bináris keresőfa fogalmát!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 8.b, Milyen kapcsolat van a bináris keres®fák és az inorder bejárás között? (Indokolja is az állítást!)

## 8.c, A t : Node\* típusú pointer egy láncoltan ábrázolt bináris keres®fát azonosít. Írja meg a printIncreasingly(t) f®eljárást és a pI(t, level) rekurzív segédeljárást, amiknek a segítségével Θ(|t|) m¶veletigénnyel ki tudja írni a fa kulcsainak szigorúan monoton növekv® sorozatát, és mindegyik kulcs mellett megjeleníti azt is, hogy az a fa hányadik szintjén található, mégpedig (kulcs/szint)alakban! Pl. a { [ 2 (3) ] 4 [ (5) 6 ({7} 8 ) ] } fára az eredmény: (2/1) (3/2) (4/0) (5/2) (6/1) (7/3) (8/2) Az eljárások a fát ne változtassák meg, és ne tartalmazzanak ciklust!

## 9.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, de- niálja a bináris keres®fa fogalmát!

Bináris keresőfa: (Minden belső csúcsára és annak y kulcsára igazak a követelmények)

* A csúcs bal részfájában tetszőleges csúcs x kulcsára: x < y
* A csúcs jobb részfájában tetszőleges csúcs z kulcsára: z > y

## 9.b, A t:Node\* egy láncoltan ábrázolt bináris keres®fát azonosít. Írja meg az el®adásról ismert módon az insert(t, k) eljárás rekurzív struktogramját!

## 9.c, Mekkora aszimptotikusan az mT(n) és az MT(n)? Miért?

## 10.a, Mondja ki a bináris rendezőfa definícióját!

Egy bináris fát rendezőfának nevezünk, ha minden belső csúcsára és annak y kulcsára igazak az alábbi követelmények:

- A csúcs bal részfájában tetszőleges csúcs x kulcsára x ≤ y.

- A csúcs jobb részfájában tetszőleges csúcs z kulcsára z ≥ y.

A keresőfában tehát minden kulcs egyedi, míg a rendezőfában lehetnek duplikált és többszörös kulcsok is.

## 10.b, A t:Node\* egy láncoltan ábrázolt bináris rendez®fát azonosít. Írja meg az insert(t,k) eljárás rekurzív, ciklust nem tartalmazó struktogramját, MT(h) ∈ Θ(h) m¶veletigénnyel!

## 10.c, Hogyan tudna a fenti eljárásra stabil rendezést építeni?

6. Rendezés lineáris id®ben

# 6.1. Edényrendezés a [0;1) intervallumon (bucket sort)

## 1.a, A ⟨0,4; 0,82; 0,0; 0,53; 0,73; 0,023; 0,64; 0,7; 0,39; 0,203⟩, tíz elemű listán mutassa be a bucket sort algoritmusát [0; 1)-beli kulcsokra!

## 1.b, Adja meg a bucket sort struktogramját!

A képen szöveg, képernyőkép, Betűtípus, nyugta látható

Automatikusan generált leírás

## 1.c, Mekkora a minimális műveletigénye? Mekkora az átlagos műveletigénye, és milyen feltétellel? Hogyan tudnánk biztosítani, hogy a maximális műveletigénye Θ(n lg n) legyen?

Nyilván mT(n) ∈ Θ(n), és a fenti egyenletes eloszlást feltételezve AT(n) ∈ Θ(n) is teljesül. MT(n) pedig attól függ, hogy a B[j] listákat milyen módszerrel rendezzük. Pl. egyszerű beszúró rendezést használva MT(n) ∈ Θ(n^2 ), összefésülő rendezéssel viszont MT(n) ∈ Θ(n log n).

## 2.a, A ⟨0,42; 0,16; 0,64; 0,39; 0,20; 0,89; 0,13; 0,79; 0,53; 0,71⟩ listán mutassa be és magyarázza el az bucket sort algoritmusát [0; 1)-beli kulcsokra!

## 2.b, Mekkora a (szokásos, beszúró rendezéses változatának) minimális és maximális műveletigénye? Miért? Mekkora az átlagos műveletigénye?





## 2.c, Hogyan tudná a maximális műveletigényt aszimptotikusan csökkenteni?

Beszúró rendezés helyett összefésülő rendezéssel viszont MT(n) ∈ Θ(n log n)

## 2.d, Mit értünk stabil rendezés alatt? Hogyan tudná a bucket sort-ot stabil rendezéssé alakítani?

## 3. Adott az L egyszer¶ láncolt lista, aminek n ≥ 0 eleme van. Minden elemének kulcsa a [0; 1) intervalumon egyenletes eloszlás szerint választott érték. Írja meg a bucketSort(L, n) utasítással meghívható egyszer¶ edényrendezés struktogramját, AT(n) ∈ Θ(n), MT(n) ∈ Θ(n lg n) m¶veletigénnyel és S(n) ∈ O(n) tárigénnyel! Segédrendezésként felhasználható a megfelel®, ebben a félévben tanult, egyszer¶ láncolt listákat kulcsösszehasonlításokkal rendez® eljárás. Ezt nem kell megírni, a kód többi részét viszont teljes részletességgel kérjük.

6.2. Leszámláló rendezés (counting sort)

## 1.a Adja meg a leszámláló rendezés előfeltételit, struktogramját és aszimptotikus műveletigényét!

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás



## 1.b, Szemléltesse a ⟨30; 20; 11; 22; 23; 13⟩ négyes számrendszerbeli számok tömbjén, ha a kulcsfüggvény a baloldali számjegyet választja ki!

## 1.c, Minek kellett teljesülnie a bemenetre, és minek a rendezésre, hogy a fenti példában a végeredmény, mint számsor is rendezett lett? Hogyan biztosítottuk a rendezés e tulajdonságát?

6.3. Radix rendezés leszámláló rendezéssel

## 1.a, Mutassa be a számjegypozíciós (Radix) rendezés működését a következő, négyes számrendszerbeli számok tömbjén: ⟨20; 02; 21; 01; 31; 20⟩! Az egyes menetekben leszámláló rendezést alkalmazzon!

## 1.b, Mekkora a fenti rendezés aszimptotikus műveletigénye, és miért?

Ha a stabil rendezés a leszámláló rendezés, akkor a műveletigény Θ(d(n+r)), mivel a teljes rendezés d leszámláló rendezésből áll. Feltételezve, hogy d konstans és r ∈ O(n), Θ(d(n + r)) = Θ(n), azaz T(n) ∈ Θ(n).

1.c, A leszámláló rendezés mint segédprogram mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

## 2.a, Mutassa be a számjegypozíciós (Radix) rendezés m¶ködését a ⟨11; 20; 10; 23; 21; 30⟩ négyes számrendszerbeli számok tömbjén! Az egyes menetekben leszámláló rendezést alkalmazzon!

## 2.b, Mekkora a Radix rendezés m¶veletigénye, és miért?

Ha a stabil rendezés a leszámláló rendezés, akkor a műveletigény Θ(d(n+r)), mivel a teljes rendezés d leszámláló rendezésből áll. Feltételezve, hogy d konstans és r ∈ O(n), Θ(d(n + r)) = Θ(n), azaz T(n) ∈ Θ(n).

## 2.c, A leszámláló rendezés mint segédprogram mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

6.4. Radix rendezés szétválogatással

## 1.a, Mutassa be a számjegypozíciós (Radix) rendezés m¶ködését a ⟨31; 20; 11; 23; 21; 10⟩ négyes számrendszerbeli számok listáján! Az egyes menetekben a megfelel® számjegy szerinti szétválogató rendezést alkalmazzon!

## 1.b, Mekkora a Radix rendezés m¶veletigénye, és miért?

Ha a stabil rendezés a leszámláló rendezés, akkor a műveletigény Θ(d(n+r)), mivel a teljes rendezés d leszámláló rendezésből áll. Feltételezve, hogy d konstans és r ∈ O(n), Θ(d(n + r)) = Θ(n), azaz T(n) ∈ Θ(n).

## 1.c, A felhasznált rendezés mint segédprogram mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

## 2. Oldja meg az el®z® feladatot a ⟨11; 20; 10; 23; 21; 30⟩ input listával!

7. Hasító táblák

7.1. Ütközés feloldása láncolással

1. A T[0..(m−1)] hasító tábla rései kétirányú, nemciklikus, fejelem nélküli, 14 rendezetlen láncolt listák pointerei. Adott a k mod m hasító függvény. A kulcsütközéseket láncolással oldjuk fel. Mindegyik kulcs csak egyszer szerepelhet T-ben.

1.a, Írja meg az ins(T, k, a):0..2 érték¶ függvényt, ami beszúrja a hasító táblába a (k, a) kulcs-adat párt! Ha a táblában már volt k kulcsú elem, a beszúrás meghiúsul, és a 2 hibakódot adja vissza. Különben, ha nem tudja már a szükséges listaelemet allokálni, az 1 hibakódot adja vissza. (Feltesszük, hogy a new m¶velet, ha sikertelen, akkor pointert ad vissza.) Az ins() m¶velet akkor ad vissza 0 kódot, ha sikeres volt a beszúrás. Ilyenkor az új listaelemet a megfelel® lista elejére szúrja be.

1.b, Mi a kitöltöttségi hányados? Milyen aszimptotikus becslést tud adni a fenti m¶velet minimális, átlagos és maximális futási idejére? Miért?

2. A T[0..(m−1)] hasító tábla rései kétirányú, nemciklikus, fejelem nélküli, rendezetlen láncolt listák pointerei. Adott a k mod m hasító függvény. A kulcsütközéseket láncolással oldjuk fel. Mindegyik kulcs csak egyszer szerepelhet T-ben.

(2.a) Írja meg a search(T, k) függvényt, ami visszaadja a T-beli, k kulcsú listaelem címét, vagy a pointert, ha ilyen nincs!

(2.b) Írja meg a del(T, p) eljárást, ami törli a T hasító táblából (és deallokálja is) a p pointer által mutatott listaelemet!

(2.c) Mi a kitöltöttségi hányados? Milyen aszimptotikus becslést tudunk adni a fenti m¶veletek minimális, átlagos és maximális futási idejére? Miért?

7.2. Nyílt címzés

1. A T[0..(m−1)] hasító táblában a kulcsütközést nyílt címzéssel oldjuk fel.

1.a, Mit értünk kitöltöttségi hányados, próbasorozat és egyenletes hasítás alatt? Mit tudunk a keresés és a beszúrás során a próbák várható számáról?

1.b, Mekkora egy sikertelen keresés várható hossza 80%-os kitöltöttség esetén, ha nincs törölt rés? Egy sikeres keresésé ennél több vagy kevesebb? Miért?

1.c, Legyen most m = 11, h1(k) = k mod m, és alkalmazzon lineáris próbát! Az alábbi m¶veletekre szemléltesse a hasító tábla változásait az el®adásról ismert módon! Szúrja be a táblába sorban a következ® kulcsokat: 10; 22; 31; 4; 15; 28; 16; 26; 62; ezután törölje a 16-ot, majd próbálja megkeresni a 27-et és a 62-t, végül pedig szúrja be a 27-et! Szemléltesse a hasító tábla változásait az el®adásról ismert módon!

2. A T[0..(m−1)] hasító táblában a kulcsütközést nyílt címzéssel oldjuk fel.

2.a, Mit értünk kitöltöttségi hányados és egyenletes hasítás alatt? Mit tudunk a keresés és a beszúrás során a próbák várható számáról? 15

2.b, Mekkora egy sikertelen keresés várható hossza 90%-os kitöltöttség esetén, ha nincs törölt rés? Egy sikeres keresésé ennél több vagy kevesebb? Miért?

2.c, Legyen most m = 7. Nyílt címzést és kett®s hasítást alkalmazunk a h1(k) = k mod m, h2(k) = 1 + (k mod (m − 2)) hasító függvények segítségével. Az alábbi m¶veletek mindegyikére adja meg a próbasorozatát ⟨. . .⟩ alakban! Szemléltesse a hasító tábla változásait az el®adásról ismert módon! Szúrja be a táblába sorban a következ® kulcsokat: 22; 31; 4; 28; 15; 14; 30; ezután törölje a 14-et, majd próbálja megkeresni a 38-at és a 30-at, végül pedig szúrja be a 27-et!

3.a, Adott egy m = 7 méret¶, üres hasító tábla. Nyílt címzést és kett®s hasítást alkalmazunk a h1(k) = k mod m, h2(k) = 1 + (k mod (m − 2)) hasító függvények segítségével. Az alábbi m¶veletekre szemléltesse a hasító tábla változásait az el®adásról ismert módon (i-beszúrás, s-keresés, d-törlés)! i37, i45, i19, i72, i33, d19, s12, i33, d33, i33.

3.b, Magyarázza meg, mi a szerepe nyílt címzés esetén a foglalt, az üres és a törölt réseknek, a beszúrás, a keresés és a törlés során!

3.c, Mi a kitöltöttségi hányados? Milyen becslést tudunk adni az egyes m¶veletigényekre, és milyen feltétellel?

4. Adott egy m = 11 méret¶ üres hasító tábla. Nyílt címzést és kett®s hasítást alkalmazunk a h1(k) = k mod m és a h2(k) = 1 + (k mod (m − 1)) hasító függvények segítségével.

4.a, Az alábbi m¶veletekre szemléltesse a hasító tábla változásait az el®adásról ismert módon! Szúrja be a táblába sorban a következ® kulcsokat: 12; 17; 23; 105. Ezután törölje a 23-at, majd próbálja megkeresni a 133-at, végül pedig szúrja be a 133-at!

4.b, Magyarázza meg, mi a különbség nyílt címzés esetén a foglalt, az üres és a törölt rések között, az alkalmazható m¶veletek szempontjából!