1. Függvények aszimptotikus viselkedése

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

**1.a, Adjuk meg az O(g) és az Ω(g) függvényhalmazok defnícióját!**

O(g) = { f | létezik c > 0 ÉS N eleme N(természetes számok halmaza), hogy minden n>=N esetén f(n) <= c\*g(n)} f-nek g aszimptotikus függvénye ha f eleme O(g)

Ω(g) = { f | létezik c > 0 ÉS N eleme N(természetes számok halmaza), hogy minden n>=N esetén f(n) >= c\*g(n)} f-nek g aszimptotikus függvénye ha f eleme Ω (g)

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

Θ(g) = O(g) metszve Ω(g)

**1.c, Igaz-e, hogy (3n + 4)2 ∈ Θ(n2) ? Miért?**

9n2 + 24n + 16 eleme Θ(n2) => igaz

elég nagy n-re az a\* n2 + b\*n + c polinomban elhanyagolható a b\*n és a c

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

2n felülről becsülhető nn**-**el => igaz

**1.e, Igaz-e, hogy 1000n2 (ln n)2 ∈ O(n3) ? Miért?**

1000n2 (ln n)2 nem becsülhető felül n3-al => hamis

**2.c, Igaz-e, hogy (2n + 1)(3n − 4) ∈ Θ(n2) ? Miért?**

6n2 - 5n – 4 eleme Θ(n2) => igaz

elég nagy n-re az a\* n2 + b\*n + c polinomban elhanyagolható a b\*n és a c

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

2n felülről becsülhető n!-el => igaz

**2.e, Igaz-e, hogy (n ln n)2 ∈ Θ(n3) ? Miért?**

n2 log2(n) nem becsülhető alulról n3-al => hamis

2. Összehasonlító rendezések

**1.a, Osztályozza az előadásról ismert négy összehasonlító rendezést műveletigény (MT(n), AT(n), mT(n)) szempontjából!**

(Θ == THETA)

BeszúróRendezés: mT(n) ∈ Θ(n), MT(n) ∈ Θ(n2), AT(n) ∈ Θ(n2)

ÖsszefésülőRendezés: mT(n) ∈ Θ(n log n), MT(n) ∈ Θ(n log n), AT(n) ∈ Θ(n log n)

GyorsRendezés: mT(n) ∈ Θ(n log n), MT(n) ∈ Θ(n2), AT(n) ∈ Θ(n log n)

KupacRendezés: mT(n) ∈ Θ (log m) (m ∈ {n−1, n−2, . . . , 1}), MT(n) ∈ Θ (log n)

**1.b, Mondja ki az összehasonlító rendezések maximális műveletigényének alsó korlátjára vonatkozó alaptételt!**

(Ω == OMEGA)

Tetszőleges rendező algoritmusra MT(n) ∈ Ω(n log n).

⇒ mind az n rendezendő elem értékét ellenőriznie kell, valamint minden alprogramhívás és ciklusiteráció csak korlátos számú elemet ellenőriz. Ha a korlátok maximuma k

⇒ MT(n) ∗ k ≥ MC(n) ⇒ MT(n) ≥ 1/k MC(n) ⇒ MT(n) ∈ Ω(MC(n)) ⇒ MT(n) ∈ Ω(n log n)

**1.c, Bizonyítsa be a kulcsösszehasonlítások számának maximumára vonatkozó lemmát!**

(MC == maximális kulcsősszehasonlítás)

Bármely összehasonító rendezés végrehejásához a legrosszabb esetben MC(n) ∈ Ω(n log n) kulcsösszehasonlítás szükséges.

Graphical user interface, text

Description automatically generated with medium confidence

**1.d, Mi a jelentősége ennek a tételnek?**

Az összefésülő és a kupacrendezés aszimptotikusan optimális. A műveletigényük O(n log n). Mindkettőre igaz, hogy MT(n) ∈ Ω(n log n).

**2.b, Mit tud mondani a rendezések minimális műveletigényének alsó korlátjáról? Miért?**

(Ω == OMEGA)

Tetszőleges rendező algoritmusra mT(n) ∈ Ω(n).

⇒ mind az n rendezendő elem értékét ellenőriznie kell, valamint minden alprogramhívás és ciklusiteráció csak korlátos számú elemet ellenőriz. Ha a korlátok maximuma k

⇒ mT(n) ∗ k ≥ n ⇒ mT(n) ≥ 1/k n ⇒ mT(n) ∈ Ω(n)

**2.c, Osztályozza az előadásról ismert négy összehasonlító tömbrendezést (maximális és minimális) tárigény szempontjából! (A tárigény = az algoritmus végrehajtásához használt temporális adatok számára + az alprogram hívások számára szükséges tárterület.)**

BeszúróRendezés: M(n) ∈ Θ(1)

ÖsszefésülőRendezés: M(n) ∈ Θ(1)

GyorsRendezés: M(n) ∈ Θ(log n)

KupacRendezés: M(n) ∈ Θ(1)

**2.d, A fenti négy rendezés közül melyeket javasolná egyirányú láncolt listák rendezésére is? Mit tud mondani ezen listarendezések tárigényéről?**

Összefésülő rendzés: kisebb allistákra szedi szét a listát

**2.e, A fenti négy rendezés közül melyiket javasolná előrendezett, kétirányú láncolt listák rendezésére? Miért?**

Beszúró rendezés

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!**

5 7 6484

57 6 484

567 4 84

4567 8 4

45678 4

445678

⇒ stabil

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

Table

Description automatically generated

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

mT(n) = n = 6

MT(n) = ½ n2 + ½ n + 1 = 16

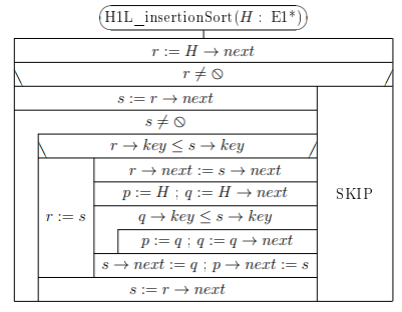
**1.d, Mit jelentenek ezek az eredmények aszimptotikusan?**

aszimptotikusan a legrosszabb esethez esik közel

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

Stabil, ami a helyén van nem mozgatja, ha két egyforma elem van a tömbben a rendezett listában bal oldalt lesz ami az eredeti listában bal oldalt volt és jobb oldalt lesz ami az eredeti listában jobb oldalt volt.

**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.)**



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

s felveszi r->next értékét, ha r->key <= s->key akkor r továbbmegy

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

mT(n) ∈ Θ(n), MT(n) ∈ Θ(n2), AT(n) ∈ Θ(n2)

**3.b, Adja meg fejelemes, kétirányú, ciklikus listára (C2L) a beszúró rendezés struktogramját! A listát a key mez®k 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 out(q), precede(q, r) és follow(p, q) eljárásokkal szabad módosítani, ahogy azt az el®adáson tanultuk.)**

**Diagram, table

Description automatically generated**

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

A beszúró rendezés jobbról balra rendez, valamint nincs megengedve egyenlőség. Ez biztosítja a stabilitást.

2.2. Összefésülő rendezés

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

4 35 21 83

4 35 12 38

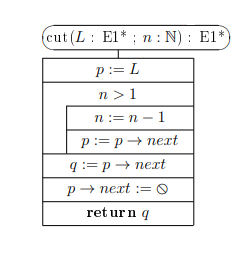
345 1238

1233458

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

Diagram

Description automatically generated



**1.c, Mekkora a rendezés műveletigénye? Röviden indokoljuk állításunkat! (Csak a bizonyítás vázlatát kell leírni.)**

mTcut(n) ∈ Θ(n), MTcut(n) ∈ Θ(n log n)

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

Table

Description automatically generated

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

mT(n) ∈ Θ(n log n), MT(n) ∈ Θ(n log n), M(n) ∈ Θ(1)

**3.a, Szemléltessük az összefésülő rendezés (merge sort) működését az előadásról ismert módon az <5; 3; 2; 1; 4; 9; 2> sorozatra!**

5 32 14 92

5 23 14 29

235 1249

1223459

**3.c, Mekkora a merge(L, L2) eljárás műveletigénye? Miért?**

mT(n) ∈ Θ(n log n), MT(n) ∈ Θ(n log n)

**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.)**

1 92 53 24

1 29 35 24

129 2345

1223459

**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!)**

45 23 5 73 94

45 23 5 37 49

23b4b5b 3j4j5j79

23 b3 j4 b4 j 5 b5 j 79

2.3. Kupacrendezés

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

Szigorúan bináris fa: bin fa, ahol minden belső csúcsnak két gyereke van

Teljes bináris fa: szigorúan bin fa, ahol minden levél azonos szinten van

Majdnem teljes bináris fa: ha egy teljes fa levélszintjéről nulla, egy vagy több levelet elveszünk (de nem az összeset)

Balra tömörített bináris fa = Szintfolytonos bin fa = kupac: olyan majdnem teljes bin fa ahol nem lehet balra már beszúrni új levelet

A picture containing scissors, watch, tool

Description automatically generated

**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! Újrarajzoláskor adja meg a fát tartalmazó tömb pillanatnyi állapotát is! Elég a kupaccá alakítást és még utána a fő ciklus első 3 iterációját (a 3. lesüllyesztés végéig) szemléltetni.**

3

9 8

2 4 6 7

5

9

3 8

2 4 6 7

5

9

3 8

5 4 6 7

2

9

5 8

3 4 6 7

2

az új tömb <9,5,8,3,4,6,7,2> ⇒ <2,5,8,3,4,6,7,**9**>

2

5 8

3 4 6 7

8

5 2

3 4 6 7

8

5 7

3 4 6 2

az új tömb <8,5,7,3,4,6,2,**9**> ⇒ <2,5,7,3,4,6,**8**,**9**>

2

5 7

3 4 6

7

5 6

3 4 2

az új tömb <7,5,6,3,4,2, **8**,**9**> ⇒ <2,5,6,3,4,**7**,**8**,**9**>

2

5 6

3 4

6

5 2

3 4

az új tömb <6,5,2,3,4,**7**, **8**,**9**> ⇒ <2,5,3,4,**6**,**7**,**8**,**9**>

2

5

3 4

5

2

3 4

5

4

3 2

az új tömb <5,4,3,2,**6**,**7**, **8**,**9**> ⇒ <2,4,3,**5**,**6**,**7**,**8**,**9**>

2

4

3

4

2

3

4

3

2

az új tömb <4,3,2,**5**,**6**,**7**, **8**,**9**> ⇒ <2,3,**4**,**5**,**6**,**7**,**8**,**9**>

2

3

3

2

az új tömb <3,2,**4**,**5**,**6**,**7**, **8**,**9**> ⇒ <2,**3**,**4**,**5**,**6**,**7**,**8**,**9**>

**2.b, Szemléltesse a kupacrendezést a következő tömbre! <5; 7; 6; 4; 2; 8; 9; 4; 3>**

5

7 6

4 2 2 8

9 4 3

9

7 8

4 2 2 6

5 4 3

9

7 8

5 2 2 6

4 4 3

az új tömb <9,7,6,5,2,2,8,4,4,3> ⇒ <3,7,8,5,2,2,6,4,4,**9**>

3

7 8

5 2 2 6

4 4

8

7 3

5 2 2 6

4 4

8

7 6

5 2 2 3

4 4

az új tömb <8,7,6,5,2,2,3,4,4,**9**> ⇒ <4,7,6,5,2,2,3,4,**8**,**9**>

4

7 6

5 2 2 3

4

7

4 6

5 2 2 3

4

7

5 6

4 2 2 3

4

az új tömb <7,5,6,4,2,2,3,4,**8,9**> ⇒ <4,5,6,4,2,2,3,**7,8,9**>

4

5 6

4 2 2 3

6

5 4

4 2 2 3

az új tömb <6,5,4,4,2,2,3,**7,8,9**> ⇒ <3,5,4,4,2,2,**6,7,8,9**>

3

5 4

4 2 2

5

3 4

4 2 2

5

4 4

3 2 2

az új tömb <5,4,4,3,2,2,**6,7,8,9**> ⇒ <2,4,4,3,2,**5,6,7,8,9**>

2

4 4

3 2

4

2 4

3 2

4

3 4

2 2

az új tömb <4,3,4,2,2,**5,6,7,8,9**> ⇒ <2,3,4,2,**4,5,6,7,8,9**>

2

3 4

2

4

3 2

2

az új tömb <4,3,2,2,**4,5,6,7,8,9**> ⇒ <2,3,2,**4**,**4,5,6,7,8,9**>

2

3 2

3

2 2

az új tömb <3,2,2,**4**,**4,5,6,7,8,9**> ⇒ <2,2,**3**,**4**,**4,5,6,7,8,9**>

2

2

<2,**2**,**3**,**4**,**4,5,6,7,8,9**>

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

Diagram, table

Description automatically generated

**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!]**

a süllyesztés műveletigénye O(log n)

kupaccá alakítás műveletigénye Θ(n)

a teljes futási idő O(n log n)

2.4. Gyorsrendezés

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

Diagram, table

Description automatically generated with medium confidence

**1.b, Szemléltesse a program „partition” függvényének működését a következő vektorra! <3; 4; 8; 7;** **1; 2; 6; 4>.**

legyen p=3

p helyet cserél az utolsó elemmel ⇒ <4,4,8,7,1,2,6,3>

elindulunk balról keresünk egy elemet ami nagyobb vagy egyenlő mint a p ⇒ i=4

elindulunk jobbról keresünk egy elemet ami kisebb mint p ⇒ j=2

kicseréljük i-t és j-t ⇒ <2,4,8,7,1,4,6,3>

i=4 j=1 ⇒ <2,1,8,7,4,4,6,3>

⇒ <2,1,3,7,4,4,6,8> <- a 3 a helyén van so quicksort mindkét oldalára

<2,1> ahol p=2 ⇒ <1,2>

<7,4,4,6,8> ahol p=7 ⇒ <8,4,4,6,7>

i=8 j=6 ⇒ <6,4,4,8,7>

⇒ <6,4,4,7,8> < a 7 és a 8 a helyén quicksort a bal oldalra

<6,4,4> ahol p=6 ⇒ <4,4,6>

⇒ <1,2,3,4,4,6,7,8>

**1.c, Mekkora a gyorsrendezés műveletigénye?**

mT(n) ∈ Θ(n log n), MT(n) ∈ Θ(n2), AT(n) ∈ Θ(n log n)

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

A beszúró rendezés egyes esetekben hatékonyabb, mint a gyorsrendezés. A gyorsrendezés eljárása jelentsősen gyorsítható, ha a kis méretű résztömbökre beszúró rendezére térnénk át.

**2.b, Szemléltessük a program „partition” függvényének működését a következő vektorra! <1; 2; 8; 7;** **3; 2; 6; 3>.**

p=1 ⇒ <3,2,8,7,3,2,6,1>

⇒ <1,2,8,7,3,2,6,3>

i=8 j=2 ⇒ <1,2,2,7,3,8,6,3>

i=7 j=3 ⇒ <1,2,2,3,7,8,6,3>

⇒ <1,2,2,3,3,8,6,7> <- a 3 és a jobb oldal fix

<8,6,7> ahol p=8 ⇒ <7,6,8> <- a 8 fix

<7,6> ahol p=7 ⇒ <6,7>

⇒ <1,2,2,3,3,6,7,8>

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

mT(n) ∈ Θ(n log n), MT(n) ∈ Θ(n2), AT(n) ∈ Θ(n log n)

3. Absztrakt adattípusok

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 vermek műveleteinek műveletigénye Θ(1) kivéve a push mert az mT(n) ∈ Θ(1), MT(n) ∈ Θ(n)

Text

Description automatically generated

Diagram

Description automatically generated

3.2. Sor

**1. Adja meg az előadásról ismert módon a Queue osztály tömbös reprezentációra alapozott leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?**

a sorok műveleteinek műveletigénye Θ(1) kivéve az add mert az mT(n) ∈ Θ(1), MT(n) ∈ Θ(n)

Text

Description automatically generated

Diagram, table, letter

Description automatically generated

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?**

mTadd(n) ∈ Θ(1), MTadd(n) ∈ Θ(n)

mTremMax(n) ∈ Θ(1), MTremMax(n) ∈ Θ(n)

mmax(n) ∈ Θ(1)

**Text

Description automatically generated**

**Text

Description automatically generated with low confidence**

Table

Description automatically generated

**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>.**

8

8 6

6 5 2 3

1 5 4 **9**

8

8 6

6 **9** 2 3

1 5 4 5

8

**9** 6

6 8 2 3

1 5 4 5

**9**

8 6

6 8 2 3

1 5 4 5

az új tömb: <9,8,6,6,8,2,3,1,5,4,5>

9

8 6

6 8 2 3

1 5 4 5 **8**

9

8 6

6 8 **8** 3

1 5 4 5 2

9

8 **8**

6 8 6 3

1 5 4 5 2

az új tömb: <9,8,8,6,8,6,3,1,5,4,5,2>

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

8

8 6

6 5 2 3

1 5 4

4

8 6

6 5 2 3

1 5 (8)

8

4 6

6 5 2 3

1 5

8

6 6

4 5 2 3

1 5

8

6 6

5 5 2 3

1. 4

az új tömb: <8,6,6,5,5,2,3,1,4>

4

6 6

5 5 2 3

1 (8)

6

4 6

5 5 2 3

1

6

5 6

4 5 2 3

1

az új tömb: <6,5,6,4,5,2,3,1>

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!**

Text, letter

Description automatically generated

**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.**

**Table

Description automatically generated**

(Fej == H)

4.4. Fejelemes, kétirányú, ciklikus listák (C2L)

**2. Az L1, L2 pointerek egy-egy szigorúan monoton növekvő C2L (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listákat kizárólag az out(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.**

Table

Description automatically generated

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!**

Text, letter

Description automatically generated with medium confidence

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

Diagram, table

Description automatically generated i=0 -> A[i]=process(s) -> i++ -> return i

**4. Milyen bináris fa bejárásokat ismer?**

**4.a, Adja meg a struktogramjaikat!**

Diagram

Description automatically generated

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

Tpreorder(n), Tinorder(n), Tpostorder(n), TlevelOrder(n) ∈ Θ(n), ahol n a fa mérete

5.1. Bináris keresőfák (és rendezőfák)

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

keresőfa = a bal oldali kulcsok mind kisebbek a jobb részfa kulcsai mind nagyobbak

**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.**

Table

Description automatically generated

**3.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát! Adott a remMin(t, minp) eljárás az előadásról ismert jelentéssel. (Ezt nem kell megírni.)**

Table

Description automatically generated

**3.b, Írja meg a del(t, k, s) ciklust nem tartalmazó MT(h(t)) ∈ O(h(t)) hatékonyságú rekurzív eljárást, ami megpróbálja törölni a láncoltan ábrázolt t bináris keresőfából a k kulcsú csúcsot! (Akkor tudja törölni, ha talál ilyet a fában.) Az s, logikai típusú paraméterben visszaadjuk, hogy sikeres volt-e a törlés. A fa csúcsai „parent” pointert nem tartalmaznak, számunkra ismeretlen, járulékos adatmezőket viszont tartalmazhatnak.**

**Table

Description automatically generated**

**4.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 keresőfa-e! MT(n) ∈ O(n), ahol n = |t|. MS(h) ∈ O(h), ahol h = h(t); MS pedig az algoritmus maximális tárigényét jelöli. (A bejárást és eldöntést a megfelelően inicializált, rekurzív, bst(t, k) logikai segédfüggvény végezze, ami híváskor k-ban a t kulcsainál kisebb értéket vár, visszatéréskor pedig, amennyiben t nemüres keresőfa, a t-beli legnagyobb kulcsot tartalmazza! Ha t üres, akkor k-ban maradjon a függvényhívásnál kapott érték!)**

**Table

Description automatically generated** l=true, for i=1; i<A.length; i++ {if(A[i-1]>A[i]) {l=false, break}} return l

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

a gyökér a 0.szinten van

tegyük fel a fánk így néz ki:

5

2

7

ekkor n=3 és h=2 => 3-1 >= 2 <- IGAZ

**5.b, Mikor lesz h = n − 1, és miért?**

ha minden gyereknek további egy gyereke van csak

**5.c, Bizonyítsa be, hogy h ≥ [lg n], ha a bináris fa nemüres!**

tetszőleges h magasságú teljes bináris fa csúcsainak száma tehát 1 + 2 + 4 + ... + 2h = 2h+1 − 1.

**5.d, Bizonyítsa be, hogy h = [lg n], ha az előbbi fa majdnem teljes!**

tetszőleges h mélységű, majdnem teljes bináris fa csúcsainak száma n ∈ [2h..2h+1−1), és így h = [log n]

**6.b, Milyen kapcsolat van a bináris keresőfák és az inorder bejárás között? (Indokolja is az állítást!)**

ha a bináris keresőfát inorder járjuk be akkor egy szigorúan monoton növő sorozatot kapunk

A picture containing text, clock

Description automatically generated

6. Rendezés lineáris időben

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!**

**0** 0.0 => 0.023, 0.0 => 0.0, 0.023

**1** -

**2** 0.203

**3** 0.39

**4** 0.4

**5** 0.53

**6** 0.64

**7** 0.73 => 0.7, 0.73 => 0.7, 0.73

**8** 0.82

**9** –

=> <0.0, 0.023, 0.203, 0.39, 0.4, 0.53, 0.64, 0.7, 0.73, 0.82>

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

Table

Description automatically generated

**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?**

O(n logn), Ω(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!**

0 -

1 0.16 => 0.13, 0.16

2 0.20

3 0.39

4 0.42

5 0.53

6 0.64

7 0.79 => 0.71, 0.79

8 0.89

9 -

=> <0.13, 0.16, 0.20, 0.39, 0.42, 0.53, 0.64, 0.71, 0.79, 0.89>

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

ami jobb oldalt volt eredetileg az a rendezett tömbben is jobb oldalt marad

ha a rendezésnél az egyenlő elemeket megcserélnénk

**3. Adott az L egyszerű 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 egyszerű edényrendezés struktogramját, AT(n) ∈ Θ(n), MT(n) ∈ Θ(n lg n) műveletigénnyel és M(n) ∈ O(n) tárigénnyel! Segédrendezésként felhasználható a megfelelő, ebben a félévben tanult, egyszerű láncolt listákat kulcsösszehasonlításokkal rendező eljárás. Ezt nem kell megírni, a kód többi részét viszont teljes részletességgel kérjük.**

Table

Description automatically generated

6.2. Leszámláló rendezés (counting sort)

**1.a Adja meg a leszámláló rendezés előfeltéteit, struktogramját és aszimptotikus műveletigényét!**

a radix rendezés segédfüggvénye, szükség lesz két tömbre: A = {rendezendő elemek} és B = {eredmény}, az A elemeinek muszáj egész számnak lennie, valamint meg kell adni számrendszert

Table

Description automatically generated

a műveletigény Θ(n + r), de mivel az r konstans => Θ(n)

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

C 30 20 11 22 23 13 13 23 22 11 20 30

0 0 |0 0|

1 0 1 2 |2 2| 1 0

2 0 1 2 3 |3 5| 4 3 2

3 0 1 |1 6| 5

=> B = <11,13,20,22,23,30>

**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?**

már rendezve voltak a második számjegy szerint

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!**

<20; 02; 21; 01; 31; 20>

**0** 20 20 2 2

**1** 21 01 31 3 5

**2** 02 1 6

**3** - 0 6

<20,31,01,21,02,20> => <20,20,21,01,31,02>

**0** 01 02 2 2

**1** - 0 2

**2** 20 20 21 3 5

**3** 31 1 6

<02,31,01,21,20,20> => <01,02,20,20,21,31>

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

a műveletigény Θ(n + r), de mivel az r konstans => Θ(n)

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

stabilitás

**2.a, Mutassa be a számjegypozíciós („Radix”) rendezés működését a <11; 20; 10; 23; 21; 30> négyes számrendszerbeli számok tömbjén! Az egyes menetekben leszámláló rendezést alkalmazzon!**

<11,20,10,23,21,30>

**0** 20 10 30 3 3

**1** 11 21 2 5

**2** - 0 5

**3** 23 1 6

<30,21,23,10,20,11> => <20,10,30,11,21,23>

**0** - 0 0

**1** 10 11 2 2

**2** 20 21 23 3 5

**3** 30 1 6

<23,21,11,30,10,20> => <10,11,20,21,23,30>

6.4. Radix rendezés szétválogatással

**1.a, Mutassa be a számjegypozíciós („Radix”) rendezés működését a <31; 20; 11; 23; 21; 10> négyes számrendszerbeli számok listáján! Az egyes menetekben a megfelelő számjegy szerinti szétválogatást alkalmazzon!**

<31; 20; 11; 23; 21; 10>

0 20 10 2 2

1 31 11 21 3 5

2 - 0 5

3 23 1 6

=> <20,10,31,11,21,23>

0 - 0 0

1 10 11 2 2

2 20 21 23 3 5

3 31 1 6

=> <10,11,20,21,23,31>

**2. Oldja meg az előző feladatot a <11; 20; 10; 23; 21; 30> input listával!**

<11; 20; 10; 23; 21; 30>

0 20 10 30 3 3

1 11 21 2 5

2 - 0 5

3 23 1 6

=> <20,10,30,11,21,23>

0 - 0 0

1 10 11 2 2

2 20 21 23 3 5

3 30 1 6

=> <10,11,20,21,23,30>

7. Hasító táblák

7.1. Ütközés feloldása láncolással

**1. A Z[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 Z-ben.**

**1.a, Írja meg az ins(Z, k, a):0..2 értékű 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 művelet, ha sikertelen, akkor pointert ad vissza.) Az ins() művelet akkor ad vissza 0 kódot, ha sikeres volt a beszúrás. Ilyenkor az új listaelemet a megfelelő lista elejére szúrja be.**

Text, letter

Description automatically generated

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

AT ∈ Θ (1), MT ∈ Θ (n)

**2. A Z[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 Z-ben.**

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

Text, letter

Description automatically generated

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

Text, letter

Description automatically generated

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

AT ∈ Θ (1), MT ∈ Θ (n)

7.2. Nyílt címzés

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

**1.a, Mit értünk kitöltöttségi hányados, próbasorozat és egyenletes hasítás alatt?**

kitöltöttségi hányados: alfa = n/m (n==hasító táblán tárolt adatok száma, m==tábla mérete)

próbasorozat: van potenciális és akutális, <h(k, 0), h(k, 1), . . . , h(k, m−1)>

egyenletes hasítás: a kulcsokat a rések között egyenletesen szórja szét

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

ha a keresés a h(k, i−1) próbánál áll meg => a keresés hossza i

kevesebb

**1.c, Legyen most m = 11, h1(k) = k mod m, és alkalmazzon lineáris próbát! Az alábbi műveletek mindegyikére adja meg a próbasorozatot h. . .i alakban! Szúrja be a táblába sorban a következő 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! Rajzolja újra, valahányszor egy művelet egy nemüres rés állapotát változtatja meg!**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| művelet | kulcs | h1(k) | ps | ? | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| i | 10 | 10 | 10 | I |  |  |  |  |  |  |  |  |  |  | 10 |
| i | 22 | 0 | 0 | I | 22 |  |  |  |  |  |  |  |  |  | 10 |
| i | 31 | 9 | 9 | I | 22 |  |  |  |  |  |  |  |  | 31 | 10 |
| i | 4 | 4 | 4 | I | 22 |  |  |  | 4 |  |  |  |  | 31 | 10 |
| i | 15 | 4 | 4,5 | I | 22 |  |  |  | 4 | 15 |  |  |  | 31 | 10 |
| i | 28 | 6 | 6 | I | 22 |  |  |  | 4 | 15 | 28 |  |  | 31 | 10 |
| i | 16 | 5 | 5,6,7 | I | 22 |  |  |  | 4 | 15 | 28 | 16 |  | 31 | 10 |
| i | 26 | 4 | 4,5,6,7,8 | I | 22 |  |  |  | 4 | 15 | 28 | 16 | 26 | 31 | 10 |
| i | 62 | 7 | 7,8,9,10,0,1 | I | 22 | 62 |  |  | 4 | 15 | 28 | 16 | 26 | 31 | 10 |
| d | 16 | 5 | 5,6,7 | I | 22 | 62 |  |  | 4 | 15 | 28 | D | 26 | 31 | 10 |
| s | 27 | 5 | 5,6,7,8,9,10,0,1,2 | X | 22 | 62 |  |  | 4 | 15 | 28 | D | 26 | 31 | 10 |
| s | 62 | 7 | 7,8,9,10,0,1 | I | 22 | 62 |  |  | 4 | 15 | 28 | D | 26 | 31 | 10 |
| i | 27 | 5 | 5,6,7,8,9,10,0,1,2 | I | 22 | 62 | 27 |  | 4 | 15 | 28 | D | 26 | 31 | 10 |

**2.c, Legyen most m = 8, (az egyszerűség kedvéért) h1(k) = k mod m, és alkalmazzunk négyzetes próbát a szokásos c1 = c2 = 1/2 konstansokkal! Az alábbi műveletek mindegyikére adja meg a próbasorozatát h. . .i alakban! Szemléltesse a hasító tábla változásait! Rajzolja újra, valahányszor egy művelet egy nemüres rés állapotát változtatja meg! Szúrja be a táblába sorban a következő 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!**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| művelet | kulcs | h1(k) | ps | ? | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| i | 22 | 6 | 6 | I |  |  |  |  |  |  | 22 |  |
| i | 31 | 7 | 7 | I |  |  |  |  |  |  | 22 | 31 |
| i | 4 | 4 | 4 | I |  |  |  |  | 4 |  | 22 | 31 |
| i | 28 | 4 | 4,5 | I |  |  |  |  | 4 | 28 | 22 | 31 |
| i | 15 | 7 | 7,0 | I | 15 |  |  |  | 4 | 28 | 22 | 31 |
| i | 14 | 6 | 6,7,1 | I | 15 | 14 |  |  | 4 | 28 | 22 | 31 |
| i | 30 | 6 | 6,7,1,4,0,5,3 | I | 15 | 14 |  | 30 | 4 | 28 | 22 | 31 |
| d | 14 | 6 | 6,7,1 | I | 15 | D |  | 30 | 4 | 28 | 22 | 31 |
| s | 38 | 6 | 6,7,1,4,0,5,3,2 | X | 15 | D |  | 30 | 4 | 28 | 22 | 31 |
| s | 30 | 6 | 6,7,1,4,0,5,3 | I | 15 | D |  | 30 | 4 | 28 | 22 | 31 |
| i | 27 | 3 | 3,4,6,1,5,2 | I | 15 | D | 27 | 30 | 4 | 28 | 22 | 31 |

**3.a, Adott egy m = 7 méretű, üres hasító tábla. Nyílt címzést és kettős hasítást alkalmazunk a h1(k) = k mod m, h2(k) = 1 + (k mod (m−2)) tördelő függvények segítségével. Az alábbi műveletek mindegyikére adja meg a próbasorozatot h. . .i alakban, és a hasító táblát is rajzolja újra, valahányszor egy művelet egy nemüres rés státuszát változtatja meg (i-beszúrás, s-keresés, d-törlés): i37, i45, i19, i72, i33, d19, s12, i33, d33, i33.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| művelet | kulcs | h1(k) | h2(k) | ps | ? | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| i | 37 | 2 | 3 | 2 | I |  |  | 37 |  |  |  |  |  |
| i | 45 | 3 | 1 | 3 | I |  |  | 37 | 45 |  |  |  |  |
| i | 19 | 5 | 5 | 5 | I |  |  | 37 | 45 |  | 19 |  |  |
| i | 72 | 2 | 3 | 2,5,0 | I | 72 |  | 37 | 45 |  | 19 |  |  |
| i | 33 | 5 | 4 | 5,1 | I | 72 | 33 | 37 | 45 |  | 19 |  |  |
| d | 19 | 5 | 5 | 5 | I | 72 | 33 | 37 | 45 |  | D |  |  |
| s | 12 | 5 | 3 | 5,0,3,6 | X | 72 | 33 | 37 | 45 |  | D |  |  |
| i | 33 | 5 | 4 | 5,1 | X | 72 | 33 | 37 | 45 |  | D |  |  |
| d | 33 | 5 | 4 | 5,1 | I | 72 | D | 37 | 45 |  | D |  |  |
| i | 33 | 5 | 4 | 5,1 | I | 72 | 33 | 37 | 45 |  | D |  |  |

**4. Adott egy m = 11 méretű üres hasító tábla. Nyílt címzést és kettős hasítást alkalmazunk a h1(k) = k** **mod m és a h2(k) = 1 + (k mod (m − 1)) tördelő függvények segítségével.**

**4.a, Az alábbi műveletek mindegyikére adja meg a próbasorozatát h. . .i alakban! Szemléltesse a hasító tábla változásait! Rajzolja újra, valahányszor egy művelet egy nemüres rés állapotát változtatja meg! Szúrja be a táblába sorban a következő kulcsokat: 12; 17; 23; 105. Ezután törölje a 23-at, majd próbálja megkeresni a 133-at, végül pedig szúrja be a 133-at!**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| művelet | kulcs | h1(k) | h2(k) | ps | ? | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| i | 12 | 1 | 3 | 1 | I |  | 12 |  |  |  |  |  |  |  |  |  |
| i | 17 | 6 | 8 | 6 | I |  | 12 |  |  |  |  | 17 |  |  |  |  |
| i | 23 | 1 | 4 | 1,5 | I |  | 12 |  |  |  | 23 | 17 |  |  |  |  |
| i | 105 | 6 | 6 | 6,1,7 | I |  | 12 |  |  |  | 23 | 17 | 105 |  |  |  |
| d | 23 | 1 | 4 | 1,5 | I |  | 12 |  |  |  | D | 17 | 105 |  |  |  |
| s | 133 | 1 | 4 | 1,**4**,8 | X |  | 12 |  |  |  | D | 17 | 105 |  |  |  |
| i | 133 | 1 | 4 | 1,**4**,8 | I |  | 12 |  |  |  | 133 | 17 | 105 |  |  |  |