

Seminar 1 → Algoritmi de Sortare

1) Selection Sort



Comparatii: $\frac{n(n-1)}{2}$

A:

$$\begin{aligned}\sum \text{Attr - best} &= 0 \\ \sum \text{Attr - avg} &= c(n-1), \quad c < 3 \\ \sum \text{Attr - worst} &= 3(n-2)\end{aligned}$$

worst CASE: \rightarrow sortit: $5, 1, 2, 3, 4$

fiecare comparație de 3

1.1. Pseudocod

```
for(i=0, m-1)
    minPos = i
    for(j=i+1, m)
        if(arr[j] < arr[minPos])
            minPos = j
        if(minPos != i)
            swap(arr[i], arr[minPos])
```

1.2. Explicatii

- sirul e împărtit în două subsecvenții: unul sortat (S) și altul neordonat (U)

• pt fiecare elem din U aleg min
• și il interschimb cu elem cu elementul de pe prima pos.
din sirul U

după care sirul S crește cu o poziție, iar din sirul U nu va exclude primul element

• Stabil? NU

e.g.: $3(1), 2(2), 1 \xrightarrow{\dots} 1, 3(2), 2(1)$

• Adaptiv? NU

• în BEST Case compl. rămâne la pt AVG/WORST.

1.3. Complexitate

	Attr.	Comp.	Total
Average :	$O(n)$	$O(n^2)$	$O(n^2)$
Worst	$O(n)$	$O(n^2)$	$O(n^2)$
Best	$O(1)$	$O(n^2)$	$O(n^2)$

II Sortare prin Înserare



1.1. Pseudocod

```
for(i=1, n)
    remember = arr[i]
    for(j=i-1, j>0 & arr[j] > remember)
        arr[j+1] = arr[j]
    if(j+1 != i)
        arr[i] = remember
```

b) Binary Sort

```
for(i=1, n)
    insertPos = BinarySearch(0, i, arr[i])
    remember = arr[i]
    for(j=i, j>insertPos, j--)
        arr[j] = arr[j-1]
    if(insertPos != j)
        arr[insertPos] = remember
```

1.2. Explicatii

$$A = S + U$$

- pentru fiecare elem. din U vom căuta POZITIA pe care ar fi deplasat (nu e definitivă) în sirul ordonat S

• shiftăm elementele \rightarrow după ce am găsit poziția, aleasă pînă la ea B.S.

1.3. Stabil: DA

Adaptiv: DA

worst case: descending
best case: ascending

1.4. Complexitate

	Classic	W/ Binary Search				
At.	Co.	T	At.	Co.	T	
Average	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n\log n)$	$O(n\log n)$
Worst	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n\log n)$	$O(n^2)$
Best	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n\log n)$	$O(n\log n)$

III Bubble Sort

1.1. Pseudocod

```
int m = size of Array
int stillUnsorted
do
    stillUnsorted = 0
    for(i=0, n-1)
        if(arr[i] > arr[i+1])
            swap(arr[i], arr[i+1])
            stillUnsorted = 1
    m--;
while(stillUnsorted)
```

1.2. Explicatii

- aduce cel mai mare element din sirul U la capătul sirului S
- se ia fiecare element în parte și se compară cu elementul din dreapta lui:
 - dacă e mai mic \rightarrow Swap
 - altfel trece la urm 2 elem din U

1.3. Stabil: DA

Adaptiv: DA

worst: descending
best: ascending

1.4. Complexitate

	Attr	Comp	Total
Average	$O(n^2)$	$O(n^2)$	$O(n^2)$
Worst	$O(n^2)$	$O(n^2)$	$O(n^2)$
Best	$O(1)$	$O(n)$	$O(n)$

Sem 2 \rightsquigarrow K-way Merge

21.25

I) Storage Sort (A, i, j)

```

    if  $A[i] > A[j]$ 
        then  $A[i] \leftrightarrow A[j]$ 

    if  $i+1 \geq j$ 
        then return;
     $k \leftarrow (j-i+1)/3$ 

```

Storage Sort ($A, i, j-k$)
 StorageSort($A, i+k, j$)
 StorageSort($A, i, j-k$)

• tracare:
 $7, 3, 1, 4, 6, 2 \Rightarrow SS(A, 0, 5) \Rightarrow$

$SS(0, 5)$ $2, 3, 1, 4, 5 \Rightarrow k = 2 \rightarrow SS(0, 3)$

$SS(0, 3)$: $1, 2, 3, 4$

$SS(0, 2)$: $1, 3, 2$; $k=1$

$SS(1, 1)$: $2, 3$

ret.

$SS(2, 5)$: $3, 4, 5, 6, 5$, $k=1$

$SS(2, 4)$: $3, 4, 6$; $SS(3, 5)$: $4, 6, 5$

$SS(2, 3)$: $3, 4$

ret.

$SS(1, 2)$: $1, 3$

ret.

$SS(0, 1)$: 1

nd.

cici se rede
de ce e ret. cu al
3-lea apel.

b) Sortează un sir de n elemente.
 Cum sortează $n+1$ elemente?



De ce nu e optim? interclasarea verifică suplimentar prea multă d. clasare



$$K = (n+1-1)/3 = n/3$$



$$\begin{cases} SS(0, n+1-n/3) \sim SS(0, n/3) \\ SS(n/3, n) \end{cases}$$

$$SS(0, n/3)$$

Complexity? $T(n) = a T\left(\frac{n}{b}\right) + n^c \Rightarrow T(n) = 3T\left(\frac{3}{2}n\right) + n^0$

$$a=3, c=0$$

$$\frac{1}{b} = \text{num init / dim dim expl rec} = n / (n - \frac{n}{3}) = \frac{n}{\frac{2}{3}n} = \frac{3}{2}$$

$$a \cdot b^c \Rightarrow 3 \cdot \left(\frac{3}{2}\right)^0 \Rightarrow 3 \cdot 1 \stackrel{c=3}{=} O(n^{\log_3 3}) = O(n^{\log_2 3})$$

II K-way Merge

```

init (L0)          f(L)
HeapSize = K
Build-Heap (A, K) O(K)
while heapSize > 0 [m steps]
    if (L0.tail != NIL)
        L0.tail.next = AT[L]
    else
        L0.head = A[1]
        A[1] = A[1].next
    // L0.tail.next = NIL
    if (A[L] = NIL)
        A[L] = A[heap.size]
    heapSize--;

```

Heapify (A, L) $O(\log K)$



Complexitate?
 $O(n \log K) + O(K)$

build heap

3) Mediana a 2 siruri ordonate

```

void divide_impar(int* a, int* b, int lo, int hi, int lb, int hb)
{
    // LUNGIMI DIFERITE
    // a > sirul de lungime mai mica
    int mid1 = (lo + hi) / 2;
    int mid2 = (lb + hb) / 2;

    int rest1 = (lo + hi) % 2;
    int rest2 = (lb + hb) % 2;
    printf("%d %d %d %d-> %d %d ", lo, hi, lb, hb, mid1, mid2);

    int new_mid1 = mid1;
    int new_mid2 = mid2;

    if (mid1 == b[mid2])
    {
        //oprire
        printf(" mijloc: %d\n", a[mid1]);
        return;
    }
    else if ((hi - lo == 0)
    {
        if (a[lo] < b[lo])
            printf("mijloc: %d", a[lo]);
        else
            printf("mijloc: %d", b[lo]);
    }
    else if (a[mid1] < b[mid2])
    {
        printf("caz mai mic\n");
        // la numar impar de elemente voi lua inclusiv mijlocul in calcul subproblemei
        // la numar par de elemente voi lua cel mai din dreapta mijloc
        if (rest1 != 0) // nr par de elem
        {
            new_mid1++;
        }
        int new_lb = new_mid2 - (hi - new_mid1);

        divide_impara(a, b, new_mid1, hi, new_lb, new_mid2);
    }
    else if (a[mid1] > b[mid2])
    {
        printf("caz mai mare\n");
        if (rest2 != 0) // nr par de elemente
        {
            new_mid2++;
        }
        int new_hb = new_mid2 + (new_mid1 - lo);
        divide_impar(a, b, lo, new_mid1, new_mid2, new_hb);
    }
}

```

```

int divide_conquer_siruri_egale(int* a, int* b, int lo, int hi, int lb, int hb)
{
    // ISUNI EGAL
    int mid1 = (lo + hi) / 2;
    int mid2 = (lb + hb) / 2;
    printf("%d %d %d %d-> %d %d ", lo, hi, lb, hb, mid1, mid2);

    int rest1 = (lo + hi) % 2;
    int rest2 = (lb + hb) % 2;
    printf("%d %d %d %d-> %d %d ", lo, hi, lb, hb, mid1, mid2);

    if (a[mid1] == b[mid2])
    {
        //oprire
        printf(" mijloc: %d\n", a[mid1]);
        return;
    }
    else if ((hi - lo == 0)
    {
        if (a[lo] < b[lo])
            printf("mijloc: %d", a[lo]);
        else
            printf("mijloc: %d", b[lo]);
    }
    else if (a[mid1] < b[mid2])
    {
        printf("caz mai mic\n");
        // la numar impar de elemente voi lua inclusiv mijlocul
        new_mid1 = mid1;
        new_mid2 = mid2;
    }
    else if (a[mid1] > b[mid2])
    {
        printf("caz mai mare\n");
        new_mid1 = mid1;
        new_mid2 = mid2;
    }
}

```

```

int* aux = (int*)malloc(4 * sizeof(int));
aux[0] = 0; aux[1] = 0; aux[2] = 0;
aux[3] = 0; aux[4] = 0;
for (int i = 0; i < 3; i++)
{
    for (int j = i + 1; j < 4; j++)
    {
        if (aux[i] > aux[j])
        {
            int t = aux[i];
            aux[i] = aux[j];
            aux[j] = t;
        }
    }
}

// mijloc: aux[1] si aux[2]
printf("mijloc: %d %d\n", aux[1], aux[2]);
return;
}

else if (a[mid1] < b[mid2])
{
    printf("caz mai mic\n");
    // la numar impar de elemente voi lua inclusiv mijlocul
    new_mid1 = mid1;
    new_mid2 = mid2;
    if (rest1 != 0) // nr par de elem
    {
        new_mid1++;
    }
    divide_conquer_siruri_egale(a, b, new_mid1, hi, new_mid2);
}
else if (a[mid1] > b[mid2])
{
    printf("caz mai mare\n");
    if (rest2 != 0) // nr par de elements
    {
        new_mid2++;
    }
    divide_conquer_siruri_egale(a, b, lo, new_mid1, new_mid2, hb);
}

```

Seminar 3

① Se dă un sir de n nr. Găsiți limita teoretică pt pb. det. simultană a min și max celor n nr.

$$\Rightarrow \frac{n}{2} + 2\left(\frac{n}{2} - 1\right) = \frac{3n}{2} - 2$$

pas 1) partitionare în cadrilateri min max

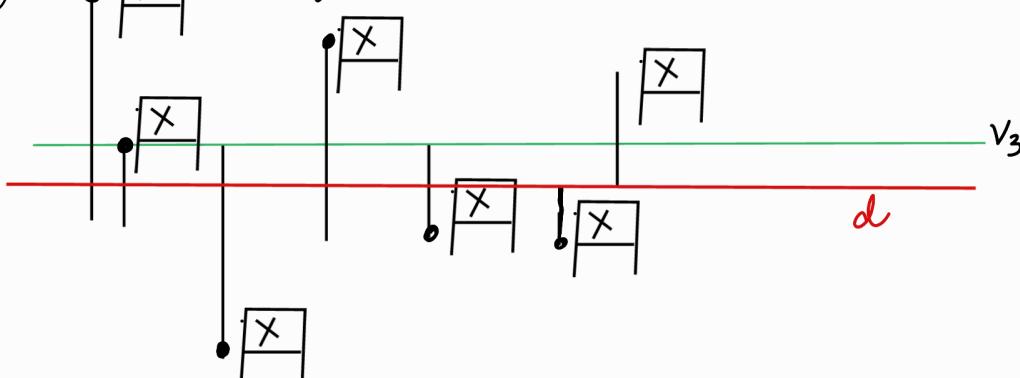
n par	m impar
$ card_{min} = card_{max} $	$= \left\lfloor \frac{n}{2} \right\rfloor + 1 =$
$= \frac{n}{2}$	$= \left\lceil \frac{n}{2} \right\rceil$

$$nr. comp = \frac{n}{2}$$

↳ cîte 2 elemente

$$nr. comp = 2\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) = 2\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) = \frac{3n}{2} - 2$$

② → algoritm de a. 1. distanța totală e



var 1) $y_d = \frac{y_1 + y_2 + \dots + y_n}{2}$

var 2) $y_d = \frac{y_{\max} + y_{\min}}{2}$

Gresit. contraexemplu: y_{\max}

Var 3) aleg pct. median

pres că numărul dr cu ϵ metri spre sud \Rightarrow
 Dist totală nouă = Dist totală veche + $\lfloor \frac{m}{2} \rfloor \cdot \epsilon + \epsilon - \lfloor \frac{m}{2} \rfloor \cdot \epsilon$

Cum găseșc mediana? Quick Select

3) ARBORI

a) hBT $O(n)$

$hBT(\text{Tree } t)$

if ($t = \text{NIL}$) then
 return -1

return $1 + hBT(t \rightarrow \text{left}) + hBT(t \rightarrow \text{right})$

$$h_T = \begin{cases} \max(h_L, h_R) + 1, & t \neq \text{NIL} \\ -1, & t = \text{NIL} \end{cases}$$

• pt arb. echilibrat
 $T(n) = 2T\left(\frac{n}{2}\right) + O(1) = O(n)$

• pt arbore carecare
 $T(n) = O(1) + T(n-1) + O(1) = \dots = n O(1) = O(n)$

b) diam BT

$diam BT(t)$

return $\max(h(t \rightarrow \text{left}) + h(t \rightarrow \text{right}) + diam(t \rightarrow \text{left}) + diam(t \rightarrow \text{right}))$

Complexitate?

average: $O(n \log n)$

worst: $O(n^2) \rightarrow h_T = n \quad \{ O(n^2)$
 $\rightarrow \text{dil. } h_T = n$

Scris

$diam BT(t)$

```

    if  $t = \text{NIL}$ 
        return  $\langle -1, -1 \rangle$ 
    L = diamBT( $t \rightarrow \text{left}$ )
    R = diamBT( $t \rightarrow \text{right}$ )
    t→h = max(L.h, R.h) + 1
    maxd = max(R.d, L.d)
    t→d = max(R.d + L.h + 1, maxd)
  
```

Complexitate?

analog hBT $\Rightarrow O(n)$

c) hMultiWay trees : $hMT = \begin{cases} \max(h_L + 1, h_R), & T = \text{NIL} \\ -1, & T \neq \text{NIL} \end{cases}$

$hMT(T)$

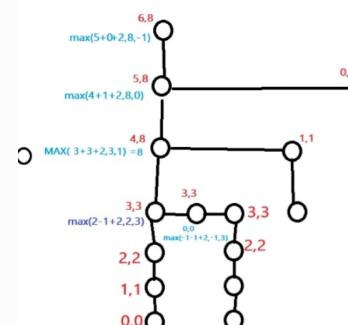
if $T = \text{NIL}$
 return -1

return $\max(hMT(T-\text{left}) + hMT(T-\text{right}))$

Complexitate: $O(n)$

d) Diametru Multi Way Trees (?)

$diam MT(t) = \{ \max(h(t \rightarrow \text{left}) + h(t \rightarrow \text{right}) + 2, \text{diam} \rightarrow \text{left}, \text{diam} \rightarrow \text{right}) \}$



4) nr. min. parcurgeri pt BST c.i. să se poată face reconstru
 rea. (R: 1 : post / pre) \rightarrow sol S4

Sem 4

- de către parcurgeri e meniu pt a da structura unui BST
- R:L (pre/post)

met 1) postorder

Structură BST (A, l, r)

```

if(l > r)
    return NIL
i <= n;
while(A[i-l] > A[r])
    i--
t = new Node(A[r])
t->left = Structură BST(A, l, i-1)
t->right = Structură BST(A, i, r)
return t;
  
```

la tabla:

Structură BST (A, s, e)

```

if(s > e)
    return NIL
t <- createNode(A[s])
s <= s+1;
i <= s;
while(i <= e)
    if(A[i] > t.key)
        break;
    i++
if(i == end)
    i++
t->left = Structură BST(A, s, i)
t->right = Structură BST(A, i+1, e)
return t;
  
```

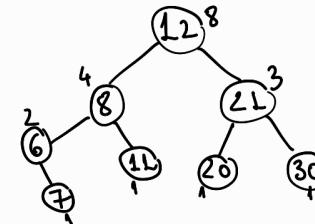
met 2) preorder

Structură BST (A, m)

```

t <- NIL
for(i <= l, m)
    t <- insertRoot(A(i))
  
```

② OS-select



obs: OS Select e eficient DOAR PT arbore echilibrat

Ce se întâmplă dacă OS select caută un nod ce nu se află în arbore?

R: os-select cauță rangul unui element și rețineaza săptă la cel rang.
dacă il nu se află în arbore \Rightarrow

$\Rightarrow i > \text{size}(\text{tree})$ sau $i < 0$

se apela OS_Select(right[X], i-r)
 \rightarrow cînd depășesc dimensiunea arborelui
și nu am $X < \text{right}[X] = \text{null}$ deci

la cîntarea dimensiunii o să am $\text{dim}[\text{left}[\text{null}]] + 1$

arbore

eritate:

if $X \neq \text{NIL}$

③ Să se construiască un PBT cu noduri numerotate de la 1 la n

build PBT (A, s, e)

if ($s > e$)

return (NIL, 0)

t <- newNode(A[(s+e)/2], size, $(s+e)/2$)

t->left = build PBT ($A, s, (s+e)/2$)

t->right = build PBT ($A, ((s+e)/2+1), e$)

// $t \rightarrow \text{size} = t \rightarrow \text{left} \rightarrow \text{size} + t \rightarrow \text{right} \rightarrow \text{size} + 1;$

if ($t \rightarrow \text{left} \neq \text{NIL}$)

$t \rightarrow \text{left} \rightarrow \text{parent} = t;$

if ($t \rightarrow \text{right} \neq \text{NIL}$)

$t \rightarrow \text{right} \rightarrow \text{parent} = t;$

④ Interclasarea a 2 BST
pt care avem h_1, h_2 a.t. $h_{\text{max}} = \max(h_1 + h_2)$

met 1)

MergeTwoBST(T₁, T₂)
if T₂ = NIL then
return T₁

NT₁ = insert(T₁, T₂ → root)

NT₂ = MergeTwoBST(NT₁, T₂ → root → left)

return MergeTwoBST(NT₁, T₂ → root → right)

Complexity?

$$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$$

$\sim n (?)$

Ce face? „destramă” T₂ în subarborele stg și s. drept (devin „arboare independențială”)

met 2)

MergeTwoBST(T₁, T₂)
Arr₁ = inorder(T₁)

Arr₂ = inorder(T₂)

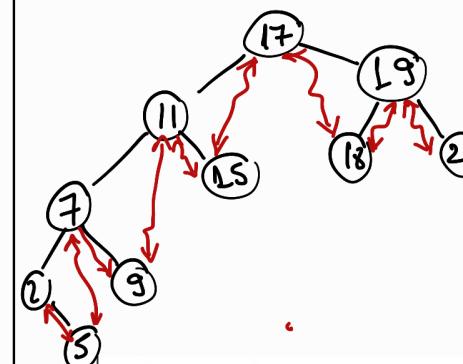
Arr_f = merge(Arr₁, Arr₂)

build - PBT (Arr_f, 1, m_{T₁} + m_{T₂})

Complexitate?
 $O(n+m)$

Seminar 5

① BST to DLL



```
Type* BSTtoDLL(Type* root)
{
    if (root == NULL)
        return NULL;

    Type* fromLeft = BSTtoDLL(root->prev_left);
    Type* fromRight = BSTtoDLL(root->next_right);

    if (fromLeft != NULL)
    {
        Type* auxLeft = fromLeft->right;
        auxLeft->next_right = root;
        root->prev_left = auxLeft;
        auxLeft->prev_left = root;
    }

    if (fromRight != NULL)
    {
        Type* aux = leftmost(fromRight);
        if (aux != NULL)
        {
            root->next_right = aux;
            aux->prev_left = root;
        }
    }

    return rightmost(fromRight);
}
else
{
    return root;
}
}
```

creează circuit
în față
modificări
înțelese în
arbore $\Rightarrow O(n)$

principiu
de la
față logaritmică
return root.

\hookrightarrow Best $O(n) \rightarrow$

Solutie bazați pe inordine $\Rightarrow O(n)$ r.t
 prev = null \rightarrow globală $\left\{ \begin{array}{l} O(h+l) \text{ mem.} \\ \end{array} \right.$

BST to DLL (root) {

```

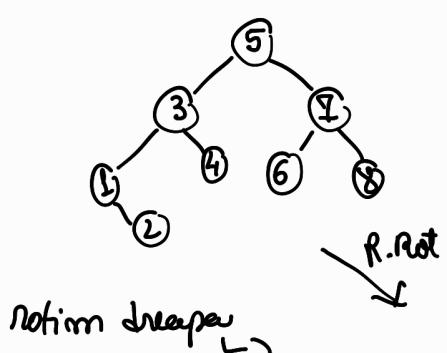
if (root = nil)
    return
BST to DLL (root->left):
    root->left = prev;
    if prev != nil
        prev->right = prev
    prev = root;
    BST to DLL (root->right);
    }
```

② Dati un alg ce transf. un BST intr-un BST degenerat pe stanga [folosind recursivitate limitată]

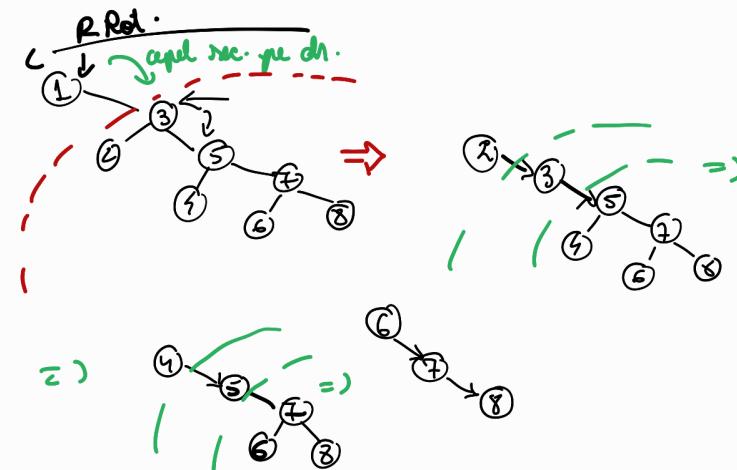
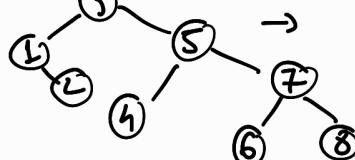
similar cu ① pe care ramura de la dreapta este o recursie maximă

Avantaj? : memorie
 \hookrightarrow ne transf. în cera iterativ fără a implementa efectiv o stivă

dvs : ne cere să dezechilibram un arbore.



rotim dreapta \hookrightarrow



Transform (T)

if T = nil then return nil
 if T.left = nil

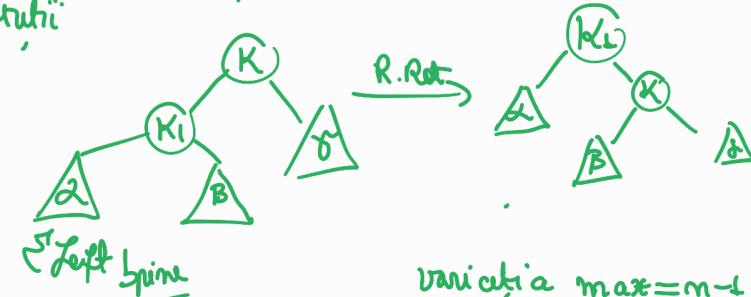
T.right = Transform(T.left)
 return T;

(else)

return Transform(R.Rot(T)) //ret ref la arborele non returnat

deac conta!

pot face rot. de mai multe ori și pe un sg. nod. DAR pt alte noeuduri nu e nes de rotitii



St Left Spine

variația max = m - 1 mod

③ Dece că pentru treef. un ABC dat într-o altă structură de ABC va fi mult cel mult $O(m)$ rot.

2) Dece' nu alăt ce treef. un ABC sănăt-un alt ABC nu poate avea multime de chei.

1) Dacă \rightsquigarrow Right degenerated ...?

din pb 2)

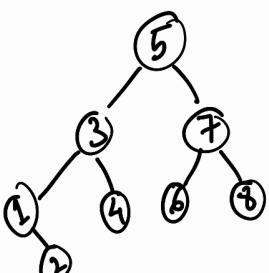
$$T_1 \rightarrow RT_1 : R\text{Rot}_{11}, R\text{Rot}_{12}, \dots, R\text{Rot}_{1K}, 1 \leq K \\ (\text{Longest Right} \dots)$$

$$T_2 \rightarrow RT_2 : R\text{Rot}_{21}, R\text{Rot}_{22}, \dots, R\text{Rot}_{2K} (T \text{ crește} \\ \text{ cu } 1 \text{ la} \\ \text{ fiecare } R\text{Rot})$$

$$\leftarrow : L\text{Rot}_{2m}, R\text{Rot}_{2m-1}, \dots, L\text{Rot}_{21}$$

$$T_L \rightarrow RT_L = RT_2 \rightarrow T_2 \\ \underbrace{\quad}_{K} \quad \quad$$

$O(n)$ rot.



ii) \rightsquigarrow degenerare T_3 , clegh. $T_2 \Rightarrow$
 $RT_L \rightarrow$ recu. II de deg.
 și RLeft.

④ Dămulse un el. și sănăt-un arbore de statistică de ordin m de dim n , dat. căt. al K-lea maxim al lui \exists și $O(\log n)$

OS-Rank $\rightarrow O(\log n)$

OS-Sel $\rightarrow O(\log n)$

OS Sel $(OS\text{-Rank}(\exists) + K)$

3

⑤ RB enumerate

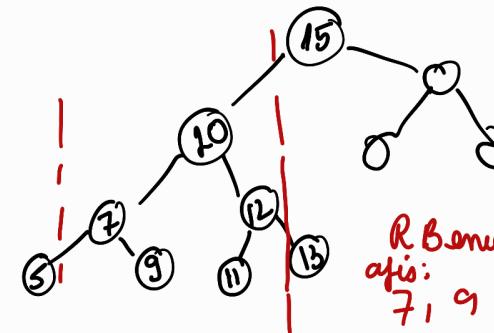
implementații alg. RB enumerate (x, a, b) ca opis

totale modulare arond cheie K sau $a \leq K \leq b$ în
 un RB

și explicatierea se ceiază la $O(m + \log n)$

$m = m$. chei afișate

$n =$ chei RB tree



\rightarrow se pot obține (apărt
 a unei și găzduie
 în arbore).

R.B enumerate($x, 7, 12$)
 apis:
 $7, 9, 10, 11, 12$

sol naivă: parcurg tot arb. și afișez dacă
 e în interval

Ce lucru să se facă în cadrul intervalelor.

cant intervale... nu de deosebit pe o ramură

RB-Enumerate(x, a, l)

if ($x = \text{nil}$)
return;

if $x.\text{key} < a$ then

RB-Enum($x.\text{right}, a, l$)

else if $x.\text{key} > a$ then

RB-Enum($x.\text{left}, a, l$)

else

RB-Enum($x.\text{right}, a, l_r$)

Show($x.\text{key}$)

RB-Enum($x.\text{left}, a, l_l$)

cáutore

parámetros

merge pl nice corbore ::