ALGORITMI I STRUKTURE PODATAKA 2

Pripremila: Lena Joković

K3-24

① - Хеш табела која сваку реч смешта на место у табели које представља збир вредности свих њених карактера, при чему је вредност карактера с с-'a'. Тиме те две речи које у међусобном поређењу дају палиндром бити хеширане на исту адресу.

- За разрешавање колизија се користи метод одвојеног улančавања

```
FIND-PALINDROME-PAIRS(words, n)
int result[][2]
for i=0 to n-1 do
    INSERT(H, i)
end_for
for j=0 to H.size-1 do
    w1 = H[j].next
    while (w1 ≠ nullptr) do
        w2=w1.next
        while (w2≠nullptr) do
            if (IS_PALINDROME(words[w1.i]+words[w2.i])) then
                result.append({w1.i, w2.i})
            end_if
            w2=w2.next
        end_while
        w1=w1.next
    end_while
end_for
return result
```

K3-24

② → 13, 18, 8, 11, 22, 72, 61

$h(k) = k \bmod 9$, $h_s(k) = 7 + k \bmod 2$

0	18
1	
2	11
3	
4	13
5	72
6	61
7	22
8	8

$13 \bmod 9 = 4$ ✓

$18 \bmod 9 = 0$ ✓

$8 \bmod 9 = 8$ ✓

$11 \bmod 9 = 2$ ✓

$22 \bmod 9 = 4$ ×

$(4 + 7 + 22 \bmod 2) \bmod 9 = 2$ ×

$(2 + 7 + 22 \bmod 2) \bmod 9 = 0$ ×

$(0 + 7 + 22 \bmod 2) \bmod 9 = 7$ ✓

$72 \bmod 9 = 0$ ×

$(0 + 7 + 72 \bmod 2) \bmod 9 = 7$ ×

$(7 + 7 + 72 \bmod 2) \bmod 9 = 5$ ✓

$61 \bmod 9 = 7$ ×

$(7 + 7 + 61 \bmod 2) \bmod 9 = 6$ ✓

- Просечан број корака приликом успешног претраживања: $\frac{13}{7}$
- Веројатноће појављивања слободних локација:

$k \bmod 2 = 0$	$k \bmod 2 = 1$
0	0
7	8
5	7
3	6
1	5
8	4
6	3
4	2
2	1
0	0

1: $\frac{1}{2}\left(\frac{1}{9} + \frac{2}{9}\right) = \frac{1}{6}$

3: $\frac{1}{2} \cdot \left(\frac{8}{9} + \frac{7}{9}\right) = \frac{5}{6}$

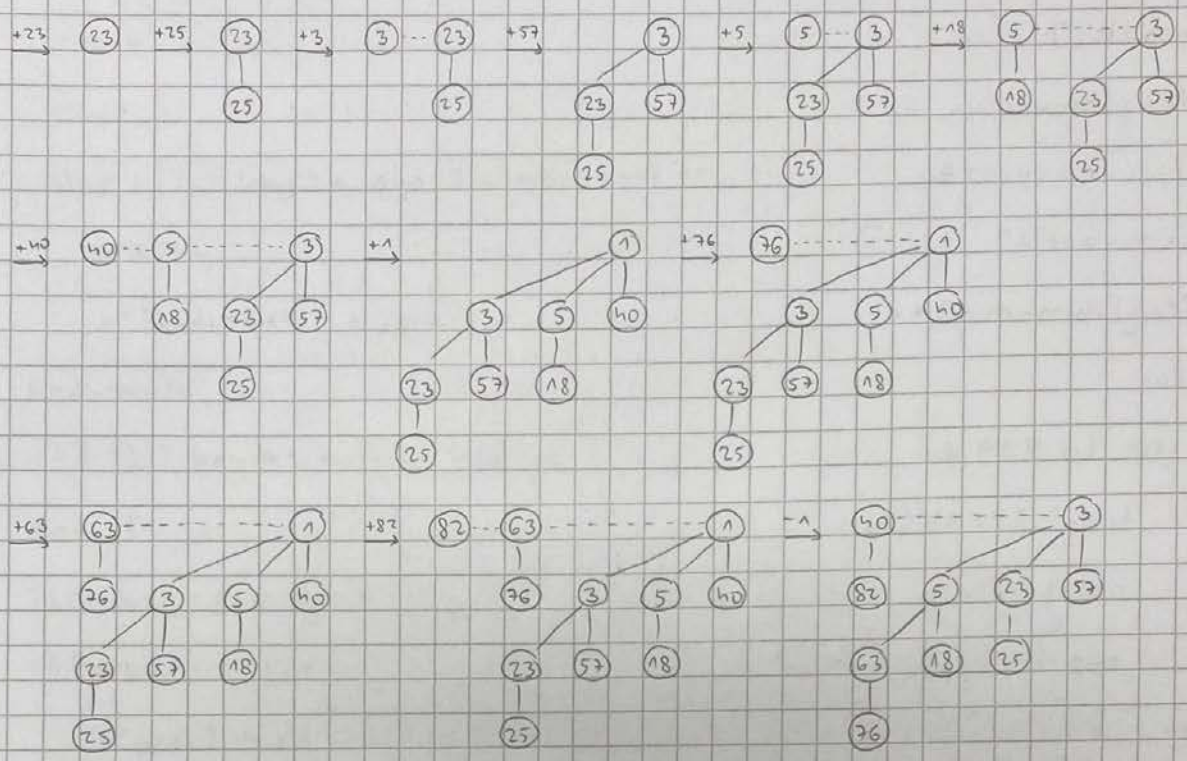
③ a) - Оптимизован bubble sort алгоритам се може завршити за мање од n итерација уколико се детектује прелаз без икакве замене, тј. уколико је низ сортиран

б)

	55	66	60	19	79	27	56	34	42	72
	55	60	19	66	27	56	34	42	72	79
	55	19	60	27	56	34	42	66	72	79
	19	55	27	56	34	42	60	66	72	79
	19	27	55	34	42	56	60	66	72	79
	19	27	34	42	55	56	60	66	72	79
	19	27	34	42	55	56	60	66	72	79

К3-24

(4) → 23, 25, 3, 57, 5, 18, 40, 1, 76, 63, 82 =>



(5) $h(k) = k \bmod 7$, линеарно претраживање

0	70
1	15
2	23
3	63
4	49
5	
6	76

- брисање 70 са адресе 0

адреса 1: $15 \bmod 7 = 1$ ✓

адреса 2: $23 \bmod 7 = 2$ ✓

адреса 3: $63 \bmod 7 = 0$ => 63 на 0

адреса 4: $49 \bmod 7 = 0$ => 49 на 3

0	63
1	15
2	23
3	49
4	
5	
6	76

- Селективно померање при брисању је претпостављање кључева после брисања са нулом да се добуду обрисана места, а не прекине истинити низ
- брисање на адреси i, прва слободна адреса j
- истраживање наредних адреса кључева од $l = i+1$ до $j-1$, $r = h(k)$
- премештање кључа k на i ако је r не налази између i и j
- понављање поступка надаље, $i = l$

K3-24

⑥ FIND_UNIQUE(arr, n, min, max)

```
int result[]
int freq[1000]
for i=0 to n-1 do
    num = arr[i]
    freq[num-min]++
end_for
for i=0 to 999 do
    if (freq[i]==1) then
        num = i + min
        result.append(num)
    end_if
end_for
return result
```

K3-20

① DELETE_ENTRY(table a, key, free)

```
i = h(k)
prev = -1
while (T[i].key ≠ k and T[i].next ≠ -1) do
    prev = i
    i = T[i].next
end_while
if (T[i].key ≠ k) then return
end_if
T[i].key = empty
if (prev ≠ -1) then
    T[prev].next = T[i].next
end_if
if (i > free) then
    free = i
end_if
return
```

КЗ-20

4. $h(k) = k \bmod 7$, квадратично претраниване

→ 38, 31, 10, 56, 21

0	10
1	56
2	21
3	38
4	31
5	
6	

$38 \bmod 7 = 3$ ✓

$31 \bmod 7 = 3$ ✗

$(3+1^2) \bmod 7 = 4$ ✓

$10 \bmod 7 = 3$ ✗

$(3+1^2) \bmod 7 = 4$ ✗

$(3+2^2) \bmod 7 = 0$ ✓

$56 \bmod 7 = 0$ ✗

$(0+1^2) \bmod 7 = 1$ ✓

$21 \bmod 7 = 0$ ✗

$(0+1^2) \bmod 7 = 1$ ✗

$(0+2^2) \bmod 7 = 4$ ✗

$(0+3^2) \bmod 7 = 2$ ✓

- просечан број пристапа приликом успешното пребарување: $\frac{12}{5}$

- просечан број пристапа приликом неуспешното пребарување:

$\frac{1}{7}(7+3+3+4+2+1+1) =$

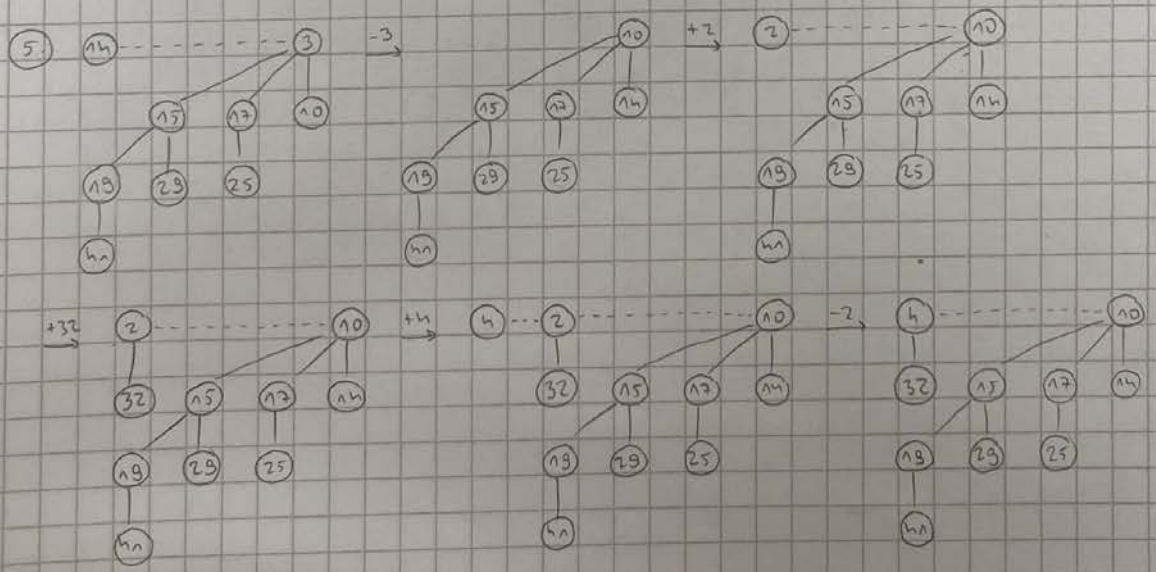
$= \frac{1}{7} \cdot 21 = 3$

- веројатноста полнување празните улаза:

k mod 7	0	1	2	3	4	5	6
i=0	0	1	2	3	4	5	6
i=1	1	2	3	4	5	6	0
i=2	4	5	6	0	1	2	3
i=3	2	3	4	5	6	0	1
i=4	2	3	4	5	6	0	1
i=5	4	5	6	0	1	2	3
i=6	1	2	3	4	5	6	0

5: $\frac{1}{7}(0+1+0+1+1+1+0) = \frac{4}{7}$

6: $\frac{1}{7}(0+0+1+0+0+0+1) = \frac{2}{7}$



K3-20

(2.) a)

7	18	23	4	10	16	9	15
7	15	9	4	10	16	23	18
7	4	9	15	10	16	23	18
7	4	9	10	15	16	18	23

7: 00111 radix exchange
18: 10010
23: 10111
4: 00100
10: 01010
16: 10000
9: 01001
15: 01111

b) - Algoritam nije stabilan, jer pri zamenama ne vodi računa o poretku ključa u međukoraku u međukoraku

(3.) HEAP_CHANGE(heap, n, key, new-key)

```
for i=1 to n do
    if (heap[i] == key) then break
end_for
heap[i] = new_key
while (i>1 and heap[i/2] > heap[i]) do
    swap(heap[i], heap[i/2])
    i = i/2
end_while
while (2·i ≤ n and (heap[2·i] < heap[i] or heap[2·i+1] < heap[i])) do
    if (heap[2·i] < heap[2·i+1]) then
        swap(heap[i], heap[2·i])
        i = 2·i
    else
        swap(heap[i], heap[2·i+1])
        i = 2·i+1
    end_if
end_while
```

K3-20

⑥ - shell sort, секвенца инкремената 1, 3, 5

159	57	103	7	24	95	8	101	179	45	303	42	219

95	8	101	7	45	159	42	103	179	24	303	57	219

7	8	57	42	45	101	24	103	159	95	303	179	219

7	8	42	45	57	24	95	101	103	159	179	219	303

- Бројеви у секвенци инкремената су узајамно прости, последњи је 1

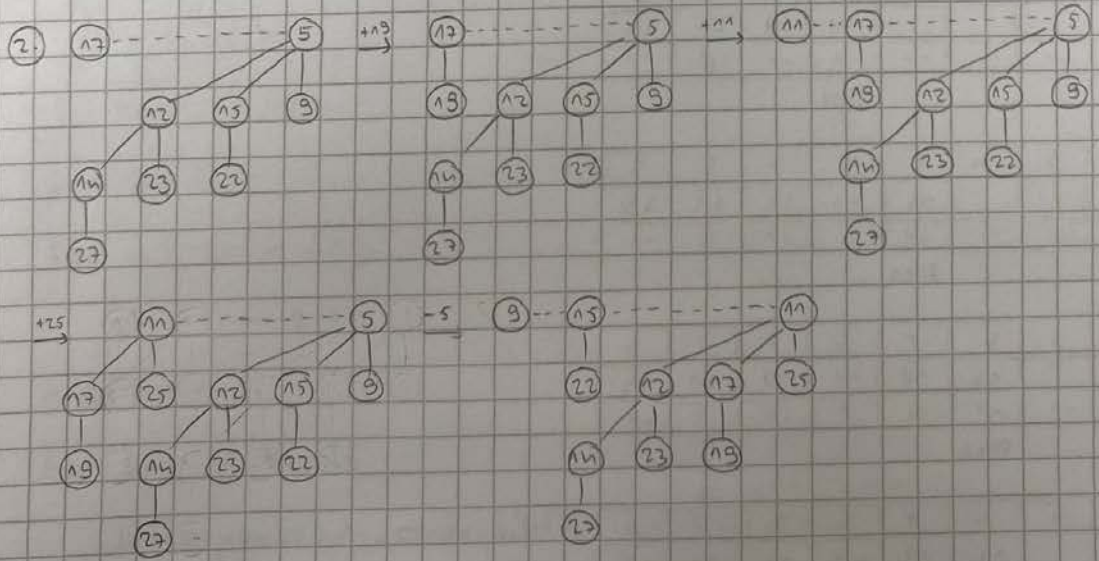
⑧ FIND_MAX_REP(arr, n, k)

```
int ans
int max_freq=0
int freq[k]
for i=0 to n-1 do
    num= arr[i]
    freq[num]++
    if (freq[num] > max_freq) then
        max_freq=freq[num]
        ans=num
    end_if
end_for
return ans
```

K3-19

① FIND_NUM_OF_PAIRS(arr, n, k)

```
int freq[n], ans=0
for i=0 to arr.size-1 do
    num = arr[i]
    freq[num mod n]++
end_for
for i=0 to n-1 do
    for j=i to n-1 do
        if ((i+j) mod n = k) then
            if (i≠j) then ans = ans + freq[i]·freq[j]
            else ans = ans + freq[i]·(freq[i]-1)/2
            end_if
        end_if
    end_for
end_for
return ans
```

② 

КЗ-19

③ radix sort 27, 25, 14, 34, 7, 1, 5, 193, 33, 41, 73, 124

- цифре единици:

0	1	2	3	4	5	6	7	8	9
	1		193	14	25		27		
	41		33	34	5		7		
			73	124					

- цифре десетици:

0	1	2	3	4	5	6	7	8	9
1	14	124	33	41			73		193
5		25	34						
7		27							

- цифре сложено: 1, 5, 7, 14, 25, 27, 33, 34, 41, 73, 124, 193

④ $h(K) = K \bmod 10$, обединено уланчавање

→ 37, 41, 18, 16, 12

- табела:

0	1	2	3	4	5	6	7	8	9
	11							24	29
-1	8	-1	-1	-1	-1	-1	-1	-1	-1

free

0	1	2	3	4	5	6	7	8	9
	11						37	21	29
-1	8	-1	-1	-1	-1	-1	-1	-1	-1

free

0	1	2	3	4	5	6	7	8	9
	11					41	37	21	29
-1	8	-1	-1	-1	-1	-1	-1	6	-1

free

0	1	2	3	4	5	6	7	8	9
	11				18	41	37	21	29
-1	8	-1	-1	-1	-1	5	-1	6	-1

free

0	1	2	3	4	5	6	7	8	9
	11			16	18	41	37	21	29
-1	8	-1	-1	-1	4	5	-1	6	-1

free

0	1	2	3	4	5	6	7	8	9
	11	12		16	18	41	37	21	29
-1	8	-1	-1	-1	4	5	-1	6	-1

free

- веројатноста пребарување празних улаза

$0: \frac{1}{10}$, $3: \frac{9}{10}$

K3-19

(7) h(k) = k mod 9, линеарно претраживање са кораком 2

- примарно труписање - делимично поклапање испитних низова два кључа са различитом матичном адресом
- секундарно труписање - поклапање испитних низова два кључа са истом матичном адресом

0	27
1	
2	11
3	
4	
5	23
6	
7	
8	8

- примарно труписање при уметању 9 и 20
- секундарно труписање при уметању 18 и 27

(5) SORT (cards)

```
for i=2 to 32 do
    k=cards[i] , j=i-1
    while(j>0 and cards[j].val > k.val) do
        cards[j+1]=cards[j]
        j=j-1
    end-while
    cards[j+1]=k
end-for
for i=1 to n do
    c[i]=(i-1)8 +1
end-for
for i=1 to 32 do
    k=cards[i] , result[c[k.sign]]=k
    c[k.sign] = c[k.sign]+1
end-for
return result
```

K3-19

⑥ QUICK_SELECT(arr,k)

elem = FIND(arr, 1, n, k)

return elem

FIND(arr, low, high, k)

j = PARTITION(arr, low, high)

i = low + k - 1

if(i=j) then return arr[i]

end_if

if(j>i) then return FIND(arr, low, j-1, k)

else return FIND(arr, j+1, high, k-j)

end_if

PARTITION(arr, low, high)

i = low, j = high

if (arr[i] < arr[$\frac{i+j}{2}$] XOR arr[i] < arr[j]) then

pivot = arr[i]

else if (arr[j] < arr[i] XOR arr[j] < arr[$\frac{i+j}{2}$]) then

pivot = arr[j], arr[i] ↔ arr[j]

else

pivot = arr[$\frac{i+j}{2}$], arr[i] ↔ arr[$\frac{i+j}{2}$]

end_if

while(i<j) do

while(arr[i] ≤ pivot and i<j) do i = i+1 end_while

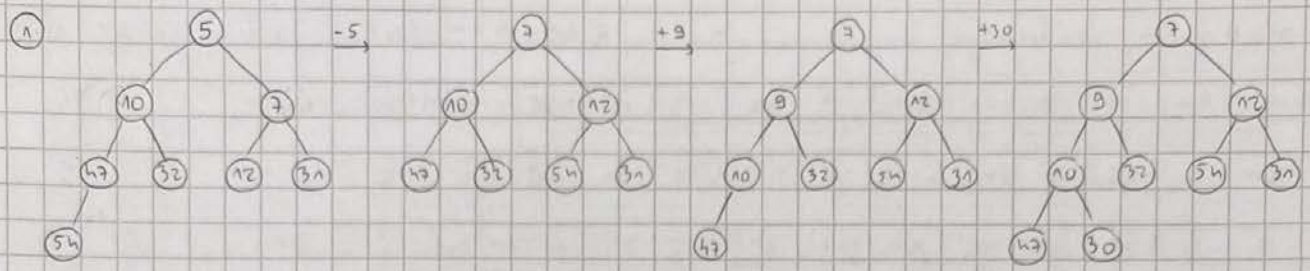
while(arr[j] > pivot) do j = j-1 end_while

if(i<j) then arr[i] ↔ arr[j] end_if

end_while

arr[low] ↔ arr[j], return j

K3-18

① 

② INSERT (table, k)

int $i = h_1(k)$, $j = h_2(k)$

while (table[i] ≠ empty and table[j] ≠ empty) do

$i = (i + g(k)) \bmod n$

$j = (j + g(k)) \bmod n$

end-while

if (table[i] = empty) then table[i] = k

else table[j] = k

end-if

SEARCH (table, k)

int $i = h_1(k)$, $j = h_2(k)$, num = 0

while (table[i] ≠ k and table[j] ≠ k and num < n) do

$i = (i + g(k)) \bmod n$

$j = (j + g(k)) \bmod n$

num++

end-while

if (num = n) return -1

end-if

if (table[i] = k) return i

else return j

end-if

К3-18

(3) - shell sort, секвенца инкремената 1, 3, 5

169	84	99	6	61	84	9	96	158	42	435	46	223
84	9	96	6	42	169	46	99	158	61	435	84	223
6	9	84	46	42	96	61	99	158	84	435	169	223
6	9	42	46	61	84	84	96	99	158	169	223	435

- Бројеви у секвенци инкремената су узајамно прости.

Последњи је 1.

(4) BUBBLE_SORT_REVISITED(arr, n)

```
for i = n-1 downto 1 do
    for j = 1 to i do
        if (arr[j] > arr[j+1]) then
            t = arr[j]
            arr[j] = arr[j+1]
            arr[j+1] = t
        end_if
    end_for
end_for
```

K3-18

(5) $h(k) = k \bmod 7$, $h_s(k) = 4 + k \bmod 3$

0	14
1	
2	11
3	24
4	18
5	
6	13

- просечан број приступа приликом неуспешног тражења

$$1\cdot\frac{2}{7} + 2\cdot\frac{5}{7}\cdot\frac{2}{6} + 3\cdot\frac{5}{7}\cdot\frac{4}{6}\cdot\frac{2}{5} + 4\cdot\frac{5}{7}\cdot\frac{4}{6}\cdot\frac{3}{5}\cdot\frac{2}{4} + 5\cdot\frac{5}{7}\cdot\frac{4}{6}\cdot\frac{3}{5}\cdot\frac{2}{4}\cdot\frac{2}{3} +$$

$$+ 6\cdot\frac{5}{7}\cdot\frac{4}{6}\cdot\frac{3}{5}\cdot\frac{2}{4}\cdot\frac{1}{3}\cdot\frac{2}{2} = \frac{2}{7} + \frac{10}{21} + \frac{12}{21} + \frac{4}{7} + \frac{10}{21} + \frac{6}{21} = \frac{6}{7} + \frac{38}{21} = \frac{56}{21} = \frac{8}{3}$$

- вероватноћа попуњавања празних улаза

k mod 3 = 0	k mod 3 = 1	k mod 3 = 2
0	0	0
4	5	6
1	3	5
5	1	4
2	6	3
6	4	2
3	2	1
0	0	0

$$1: \frac{1}{3}\left(\frac{6}{7} + \frac{2}{7} + \frac{4}{7}\right) = \frac{4}{7}$$

$$5: \frac{1}{3}\left(\frac{1}{7} + \frac{5}{7} + \frac{3}{7}\right) = \frac{3}{7}$$

(7) - хеш табела са $n=8$ улаза, метод дељења

- метода случајног претраживања $c=3$

	$p_1 =$	011	= 3
·2	$p_2 =$	110	= 6
·2		1100	
-8 ⊕ 3	$p_3 =$	111	= 7
·2		1110	
-8 ⊕ 3	$p_4 =$	101	= 5
·2		1010	
-8 ⊕ 3	$p_5 =$	001	= 1
·2	$p_6 =$	010	= 2
·2	$p_7 =$	100	= 4
·2		1000	
-8 ⊕ 3	$p_8 =$	000	= 0

$h_0(k) = k \bmod 8$

$h_i(k) = (h_0(k) + p_i) \bmod 8$, $i = 1, 2, \ldots, n-1$

- кључ 21:
  испитни низ 5, 0, 3, 4, 2, 6, 7, 1
- кључ 18:
  испитни низ 2, 5, 0, 1, 7, 3, 4, 6
- не долази до примарног груписања, јер следећа адреса у испитном низу зависи и од броја колизија до тад
- долази до секундарног груписања, јер следећа адреса у испитном низу не зависи од кључа

KB-18

⑥ REMOVE_MIN_KEY(binheap)

```
BinTree tree = binheap.findMin()
binheap.remove(tree)
int num = tree.getNumOfChildren()
for i = n to num do
    BinTree newtree = new BinTree()
    newtree.setRoot(tree.getChildAtPos(i))
    newtree.setOrder(newtree.getRoot.getNumOfChildren())
    binheap.add(newtree)
end_for
```

⑧ a) нерационална сложност $O(n\sqrt{n})$

б) средно сложност $O(n \log n)$

в) сложност $O(n \log n)$

K3 - 17

1. - shakersort алгоритам

12	3	34	83	21	95	34	1	18	20	83	17	44	39
3	12	34	21	83	34	1	18	20	83	17	44	39	95
1	3	12	34	21	83	34	17	18	20	83	39	44	95
1	3	12	21	34	34	17	18	20	83	39	44	83	95
1	3	12	17	21	34	34	18	20	39	83	44	83	95
1	3	12	17	21	34	18	20	34	39	44	83	83	95

2. $h_1(k) = (1 + len(k)) \mod 10$, $h_2(k) = 3 + up\text{-}case\text{-}cnt(k)$

а) - примарном разрешавању колизије, можемо постојеће кључеве двоструким хеширањем померити да буду дубље у испитном низу за једно место, а нови кључ ставити на његов почетак

б)

0	TORtA
1	
2	RoLaT
3	čokoLaDa
4	
5	kaFA
6	koLAČ
7	PalAČINka
8	
9	SLedoled

h_1(čoKoLaDa) = 9

h_1(PalAČINka) = 0

h_1(kaFA) = 5

h_1(TORtA) = 6

h_1(RoLaT) = 6

TORtA → (6 + h_2(TORtA)) mod 10 = (6 + 3 + 4) mod 10 = 3

h_1(SLedoled) = 9

čoKoLaDa → (9 + h_2(čoKoLaDa)) mod 10 = (9 + 3 + 4) mod 10 = 6

RoLaT → (6 + h_2(RoLaT)) mod 10 = (6 + 3 + 3) mod 10 = 2

h_1(koLAČ) = 6

čoKoLaDa → (6 + h_2(čoKoLaDa)) mod 10 = (6 + 3 + 4) mod 10 = 3

TORtA → (3 + h_2(TORtA)) mod 10 = (3 + 3 + 4) mod 10 = 0

PalAČINka → (0 + h_2(PalAČINka)) mod 10 = (0 + 3 + 4) mod 10 = 7

k3-17

(3.) QUICKSORT(arr, n)

```
STACK_INIT(S)
PUSH(S, {0, n-1})
while (!EMPTY(S)) do
    down, up = TOP(S), POP(S)
    if (down < up) then
        pi = PARTITION(arr, down, up)
        PUSH(S, {down, pi-1}), PUSH(S, {pi+1, up})
    end_if
end_while
```

PARTITION(arr, down, up)

```
if (arr[down] < arr[down+1] XOR arr[down] < arr[down+2]) then
    pivot = arr[down]
else if (arr[down+1] < arr[down] XOR arr[down+1] < arr[down+2]) then
    pivot = arr[down+1], arr[down] ↔ arr[down+1]
else
    pivot = arr[down+2], arr[down] ↔ arr[down+2]
end_if
i = down, j = up
while (i < j) do
    while((arr[i] ≤ pivot) and (i < j)) do i = i+1 end_while
    while((arr[j] > pivot) do j = j-1 end_while
    if(i < j) then arr[i] ↔ arr[j] end_if
end_while
arr[down] ↔ arr[j]
return j
```

K3 - 17

4) HEAP_KEY_DEC(i)

```
heap[i] = heap[i] - 1
int n = heap size
while ((2·i+1 < n and heap[2·i+1] < heap[i])
        or (2·i+2 < n and heap[2·i+2] < heap[i]))
    if (2·i+2 = n or heap[2·i+1] ≤ heap[2·i+2]) then
        temp = heap[i], heap[i] = heap[2·i+1], heap[2·i+1] = temp
        i = 2·i + 1
    else
        temp = heap[i], heap[i] = heap[2·i+2], heap[2·i+2] = temp
        i = 2·i+2
    end_if
end_while
```

5) $h(k) = k \bmod 7$, $h_s(k) = 4 + k \bmod 2$

0	
1	
2	2
3	16
4	
5	
6	30

- до секундарног трупкања долази при уметању кључева који имају исти остатак и по модулу 7 и по модулу 2, тј. исти низови ће се потпуно поклапају

- нпр. уметање 2, 30, 16

$h(2) = 2 \bmod 7 = 2$ ✓

$h(30) = 30 \bmod 7 = 2$ ×

$(2 + 4 + 30 \bmod 2) \bmod 7 = 6$ ✓

$h(16) = 16 \bmod 7 = 2$ ×

$(2 + 4 + 16 \bmod 2) \bmod 7 = 6$ ×

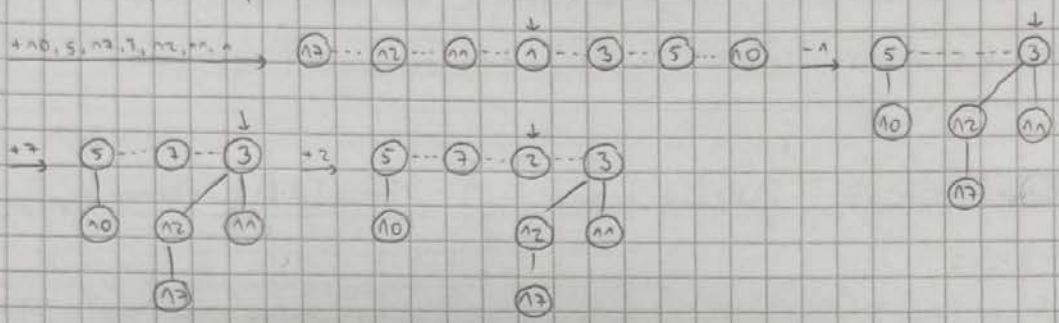
$(6 + 4 + 16 \bmod 2) \bmod 7 = 3$ ✓

K3-17

⑥ a) - При додавању у бинарни хип, слабо лева о се догађе на почетак хипа и одмах се врше два потребна спајања слабих левих

- При додавању у биномијални хип, слабо лева о се догађе неко од чворова на слабом са најмањим/највећим елементом и евентуално се анулира тај чворан

б) $\downarrow$ 10, 5, 17, 3, 12, 11, 1 $\Rightarrow$ 1 $\Rightarrow$ 3, 2



⑧ - Понашање алгоритама сортирања поређењем се може моделирати стаблом одлучивања. Свако одлучивање се моделује унутрашњим чвором, а један исход листом, тј је сложеност у најгорем случају висина стабла h

- Број исхода $n!$, највећи број листова 2^h

$n! = 2^h$

$h = \log_2(n!) > \log_2\left(\left(\frac{n}{e}\right)^n\right) = n\log_2\left(\frac{n}{e}\right) = n\log_2 n - n\cdot\log_2 e \sim \Theta(n\log n)$

КЗ-17

7. $h(k) = k \bmod n$, квадратично претърсване

DELETE(H, key)

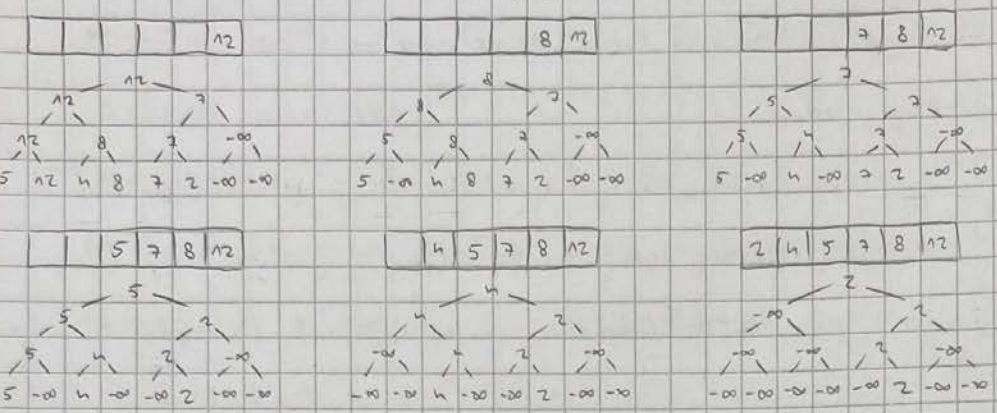
```
int n = H.size , a = key mod n
d = 0 , i = 0
while (H[a] != key and H[a] != empty and i < n) do
    d = d + i + 1 , i = i + 1 , a = (a + d) mod n
end-while
if (H[a] = key) then H[a] = deleted
else ERROR (No key)
end-if
```

INSERT(H, key)

```
int n = H.size , a = key mod n
d = 0 , i = 0
while ((H[a] != empty or H[a] != deleted) and i < n) do
    i = i + 1 , d = d + i - 1 , a = (a + d) mod n
end-while
if (i = n) then ERROR (Full table)
else H[a] = key
end-if
```

КЗ-16

① 5, 12, 4, 8, 7, 2, слабо селективно



- ефикасност не зависи од иницијалне уређености низа

② двоструко хеширање, $h_p(k)$ и $h_s(k)$

а) - При брисању, на адресу обрисаног кључа се уписује константа deleted, која означава да је са ње избрисан кључ а не да је само празна.

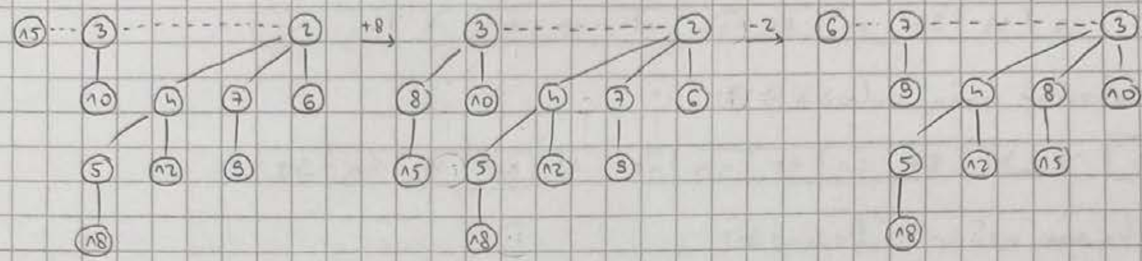
б) OPTIMIZE_HASH_TABLE(H)

```
int n=H.size
for i=0 to n do
    if (H[i] ≠ empty and H[i] ≠ deleted) then
        key = H[i]
        addr = h_s(key)
        if (addr ≠ i and (H[addr] = empty or H[addr] = deleted)) then
            H[addr]=key
            H[i] = deleted
        end_if
    end_if
end_for
```

КЗ-16

3. а) - Биномно стабло реда 0 је један чвор, а биномно стабло реда b има корен чија подстабла су корене биномних стабала редом реда b-1, b-2, ..., 1, 0

б)



4. $h(k) = k \bmod 9$, $hs(k) = 7 + k \bmod 2$

+ 23, 29, 17

0	18
1	
2	11
3	
4	13
5	23
6	17
7	29
8	8

$23 \bmod 9 = 5$ ✓

$29 \bmod 9 = 2$ ×

$(2 + 7 + 29 \bmod 2) \bmod 9 = 0$ ×

$(0 + 7 + 29 \bmod 2) \bmod 9 = 7$ ✓

$17 \bmod 9 = 8$ ×

$(8 + 7 + 17 \bmod 2) \bmod 9 = 7$ ×

$(7 + 7 + 17 \bmod 2) \bmod 9 = 6$ ✓

- вероватноћа попуњавања празних позиција

k mod 2 = 0	k mod 2 = 1
0	0
7	8
5	7
3	6
1	5
8	4
6	3
4	2
2	1
0	0

$1: \frac{1}{2}\left(\frac{1}{9} + \frac{2}{9}\right) = \frac{1}{6}$

$3: \frac{1}{2}\left(\frac{8}{9} + \frac{7}{9}\right) = \frac{5}{6}$

К3-16

(5.) - Избор пивота при извршавању quick sort алгоритма утиче на величине партиција на које се низ дели, тј. касније на број поређења, премештања и целокупно време извршавања

3	12	33	88	21	95	34	15	18	20	8	44	39

- неповољан избор пивота (3):

3	12	33	88	21	95	34	15	18	20	8	44	39

- повољан избор пивота (34):

3	12	33	8	21	20	18	15	34	95	88	44	35

(8.) а) квадратна сложеност $O(n\sqrt{n})$

б) стабло бинарног претраживања $O(n^2)$

КЗ-15

① - shellsort, сегменца инкрементата 1, 2, 5

12	3	33	88	21	95	34	1	18	20	8	44	39

8	3	1	18	20	12	34	33	88	21	95	44	39

② COUNTING_SELECT(arr, k)

```
int n=arr.size , c[16]
for i=0 to n-1 do
    c[arr[i]] = c[arr[i]] +1
end_for
for i=1 to 15 do
    C[i] = C[i] + C[i-1]
end_for
int elem=0
while (c[elem]<k) do elem=elem+1
end_while
return elem
```

③ $h(k) = k \bmod 7$, линейно претраснување со чекор 2

→ 18, 13, 11, 8, 23

0		0		0		0		0	
1		1		1	11	1	11	1	11
2		2		2		2		2	23
3		3		3		3	8	3	8
4	18	4	18	4	18	4	18	4	18
5		5		5		5		5	
6		6	13	6	13	6	13	6	13

[K3-15]

(4) MAKE_HEAP (arr)

```
int n = arr.size
for i = 2 to n do
    nhe = arr[i]
    s = i , f = s/2
    while ((s>1) and (arr[f] < nhe)) do
        arr[s] = arr[f]
        s = f , f = s/2
    end-while
    arr[s] = nhe
end-for
```

(7) 5 клуча, Brent-ов метод

→ 57, 62, 10, 20, 32

Knuth-ова препорака: $h(k) = k \bmod 5$, $g(k) = 1 + k \bmod 3$

	1	2	3	4	5
0		62	10	10	10
1				20	20
2	57	57	57	57	57
3			62	62	62
4					32

57 mod 5 = 2 ✓

62 mod 5 = 2 ✗

(2 + 1 + 62 mod 3) mod 5 = 0 ✓

10 mod 5 = 0 ✗

(0 + 1 + 10 mod 3) mod 5 = 2 ✗

(2 + 1 + 10 mod 3) mod 5 = 4 ✓

? преместување 62

(0 + 1 + 62 mod 3) mod 5 = 3 ✓

20 mod 5 = 0 ✗

(0 + 1 + 20 mod 3) mod 5 = 3 ✗

(3 + 1 + 20 mod 3) mod 5 = 1 ✓

преместување 10

(0 + 1 + 10 mod 3) mod 5 = 2 ✗

32 mod 5 = 2 ✗

(2 + 1 + 32 mod 3) mod 5 = 0 ✗

(0 + 1 + 32 mod 3) mod 5 = 3 ✗

32 mod 5 = 2 ✗

(3 + 1 + 32 mod 3) mod 5 = 1 ✗

(1 + 1 + 32 mod 3) mod 5 = 4 ✓

преместуваме 57

(2 + 1 + 57 mod 3) mod 5 = 3 ✗

KB-n5

(5.) $h(k) = k \bmod 8$, обједињено улančавање

→ 18, 10, 7, 22, 23, 28

0	1	2	3	4	5	6	7
		18					
-	-	-	-	-	-	-	-

free

0	1	2	3	4	5	6	7
		18					10
-	-	7	-	-	-	-	-

free

0	1	2	3	4	5	6	7
		18				7	10
-	-	7	-	-	-	-	6

free

0	1	2	3	4	5	6	7
		18			22	7	10
-	-	7	-	-	-	5	6

free

0	1	2	3	4	5	6	7
		18		23	22	7	10
-	-	7	-	-	4	5	6

free

0	1	2	3	4	5	6	7
		18	28	23	22	7	10
-	-	7	-	3	4	5	6

free

(8.) - Поједначење алгоритама сортирања поређењем се може моделирати стаблом одлучивања. Свако одлучивање се моделује унутрашњим чвором, а један исход листом, па је сложеност у најгорем случају једнака висини стабла h.

- Број исхода $n!$, највећи број листова 2^h

$n! = 2^h$

$h = \log_2(n!) \geq \log_2\left(\left(\frac{n}{e}\right)^n\right) = n \log_2\left(\frac{n}{e}\right) = n \cdot \log_2 n - n \log_2 e \sim O(n \log n)$

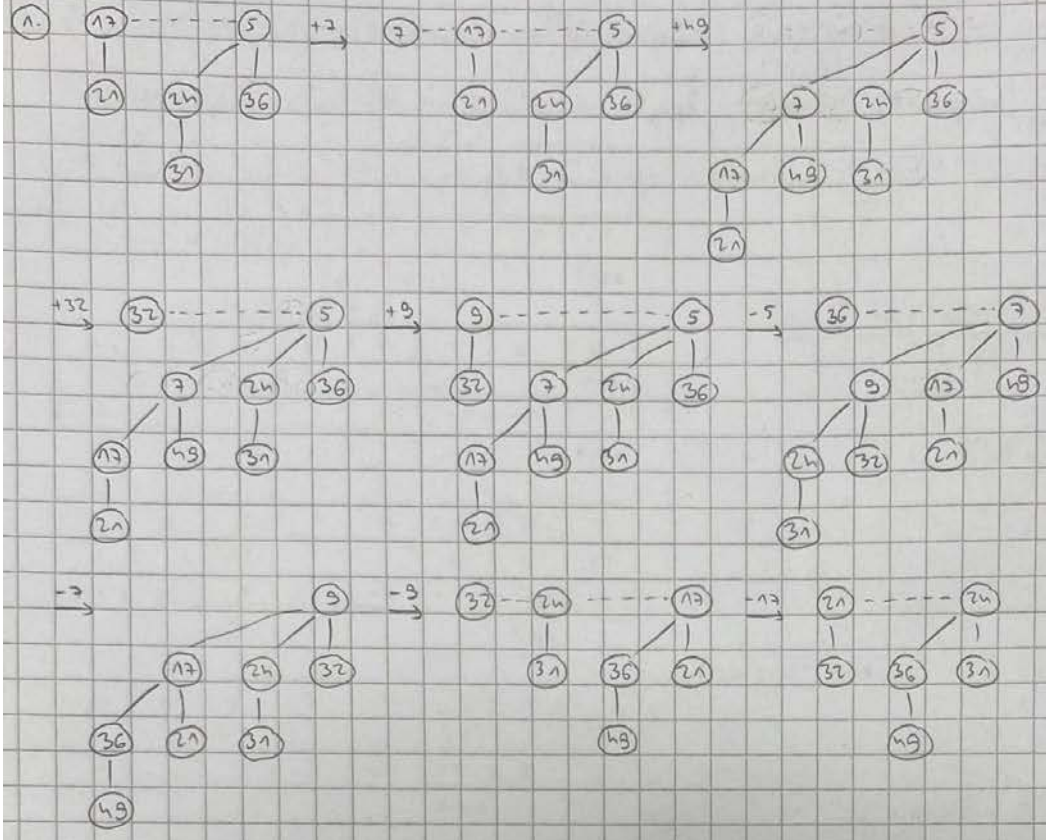
KB-15

(6) а) - Населба се могу прво сортирати лексикографски, а затим по удалености од задатот населба. Други алгоритам за сортирање по удалености мора бити стабилан иначе би одржао лексикографски поредак

б)
```
RADIX_SORT_CITIES(names, n)
QUEUE oldQueues[26]
for i=0 to n-1 do
    name = names[i]
    pos = name[9] - 'a'
    oldQueues[pos].INSERT(name)
end_for
for i=8 down to 0 do
    QUEUE newQueues[26]
    for j = 0 to 25 do
        while (not oldQueues[j].EMPTY) do
            name = oldQueues[j].DELETE
            pos = name[i] - 'a'
            newQueues[pos].INSERT(name)
        end_while
    end_for
    oldQueues = newQueues
end_for
i=0
for j=0 to 25 do
    while (not oldQueues[j].EMPTY) do
        names[i] = oldQueues[j].DELETE, i=i+1
    end_while
```

JAH-21

1.



4. HASHMAP_SEARCH_INSERT(H, n, c, key)

```
i = key mod n
while (H[i] != key and H[i] != empty) do
    i = (i+c) mod n
end_while
if (H[i] = empty) then
    H[i] = key
end_if
return i
```

ЈАН-21

(3.) а) insertion sort - 9 чекори за добивање четвртиот најмален елемент

9	3	75	85	11	81	75	11	20	51
3	9	75	85	11	81	75	11	20	51
3	9	75	85	11	81	75	11	20	51
3	9	75	85	11	81	75	11	20	51
3	9	11	75	85	81	75	11	20	51
3	9	11	75	81	85	75	11	20	51
3	9	11	75	75	81	85	11	20	51
3	9	11	11	75	75	81	85	20	51
3	9	11	11	20	75	75	81	85	51
3	9	11	11	20	51	75	75	81	85

б) quicksort - 6 чекори за добивање четвртиот најмален елемент

9	3	75	85	11	81	75	11	20	51
3	9	75	85	11	81	75	11	20	51
3	9	11	51	11	20	75	75	81	85
3	9	11	51	11	20	75	75	81	85
3	9	11	20	11	51	75	75	81	85
3	9	11	11	20	51	75	75	81	85
3	9	11	11	20	51	75	75	81	85

в) - Методи директне селекције, квадратне селекције и стабла селекције сигурно налазе k-ти најмањи елемент у k чекора

ΔΑΗ-21

(2) PERMUTATION-ARRAY(A,B,n)

RADIX_EXCHANGE(A,n)

RADIX_EXCHANGE(B,n)

int sum=0, C[n]

for i=0 to n-1 do

C[i]=A[i]·B[i], sum=sum+C[i]

end_for

return sum

RADIX_EXCHANGE(A,n)

STACK_INIT(S)

S.PUSH(0,n,0)

while(not S.STACK_EMPTY) do

low, high, bit=S.TOP, S.POP

if(bit≤16 and low<high) then

int i=low, j=high

while(i<j) do

while(i<j and A[i]=0) do i=i+1

while(A[i]=1) do j=j-1

if(i<j) then A[i]↔A[j] end_if

end_while

S.PUSH(low,j,bit+1)

S.PUSH(i,high,bit+1)

end_if

end_while

ФЕБ-24

① HASH_DELETE(key, g, n)

```
int i=key mod n , j=0
while (T[i]≠key and T[i]≠-1 and j<n) do
    i=(i+g(key)) mod n , j=j+1
end_while
if (T[i]=key) then T[i]=0
end_if
```

HASH_SEARCH(key, g, n)

```
int i=key mod n, j=0 , pos=-1
while (T[i]≠key and T[i]≠-1 and j<n) do
    if (pos=-1 and T[i]=0) then pos=i
    end_if
    i=(i+g(key)) mod n , j=j+1
end_while
if (T[i]≠key) then return -1
end_if
if (pos≠-1) then
    T[pos]=key, T[i]=0 , i=pos
end_if
return i
```

DES-24

(3) - insertion sort

5	28	71	32	6	1	41	26	98	24	3	36
5	28	32	71	6	1	41	26	98	24	3	36
5	6	28	32	71	1	41	26	98	24	3	36
1	5	6	28	32	71	41	26	98	24	3	36
1	5	6	41	28	32	71	26	98	24	3	36
1	5	6	41	26	28	32	71	98	24	3	36
1	5	6	41	24	26	28	32	71	98	3	36
1	3	5	6	41	24	26	28	32	71	98	36
1	3	5	6	41	24	26	28	32	36	71	98

(5) DELETE_FROM_INDEX(heap, n, i)

```
nhe = heap[n]
n = n - 1
s = i , f = s/2
while ( s > 1 and nhe < heap[f]) do
    heap[s] = heap[f] , s = f, f = s/2
end_while
heap[s] = nhe
f = s , s = 2·f
if ( s+1 ≤ n and heap[s+1] < heap[s] ) then s = s+1
end_if
while ( s ≤ n and nhe > heap[s]) do
    heap[f] = heap[s] , f = s , s = 2f
    if ( s+1 ≤ n and heap[s+1] < heap[s]) then s = s+1
    end_if
end_while
heap[f] = nhe
```

ФЕБ - 24

(8) h(k) = k mod 11

→ 44, 34, 26, 29, 23, 56, 54, 66

а) - линеарно претраживање c=2

0	44
1	34
2	66
3	23
4	26
5	56
6	
7	29
8	
9	
10	54

44 mod 11 = 0 ✓
34 mod 11 = 1 ✓
26 mod 11 = 4 ✓
29 mod 11 = 7 ✓
23 mod 11 = 1 ×
(1+2) mod 11 = 3 ✓

56 mod 11 = 1 ×
(1+2) mod 11 = 3 ×
(3+2) mod 11 = 5 ✓
54 mod 11 = 10 ✓
66 mod 11 = 0 ×
(0+2) mod 11 = 2 ✓

- просечан број корака приликом успешног претраживања: $\frac{12}{8} = 1,5$
- неуспешно претраживање на 11: 0, 2, 4, 6 => 4 корака
- неуспешно претраживање на 12: 1, 3, 5, 7, 9 => 5 корака

б) - метод уређене табеле при уметању одржава свих кључева који се мапирају на исту матичну адресу уређеним
- неуспешна претрага се раније завршава

0	44
1	23
2	66
3	34
4	26
5	54
6	
7	29
8	
9	
10	54

44 mod 11 = 0 ✓
34 mod 11 = 1 ✓
26 mod 11 = 4 ✓
29 mod 11 = 7 ✓
23 mod 11 = 1 ×
(1+2) mod 11 = 3 ✓

56 mod 11 = 1 ×
(1+2) mod 11 = 3 ×
(3+2) mod 11 = 5 ✓
54 mod 11 = 10 ✓
66 mod 11 = 0 ×
(0+2) mod 11 = 2 ✓

- неуспешно претраживање на 11: 0 (23 > 11) => 1 корак
- неуспешно претраживање на 12: 1 (23 > 12) => 1 корак