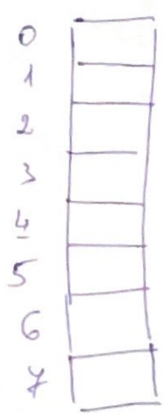
Honors Data Structures

Theoretical homework 3

Mark Kirichev, UNI: mmk 2243

Problem 1.

a) Our table is of size 8, so it's reasonable that our keys would be the integers from 0 to 7 included.



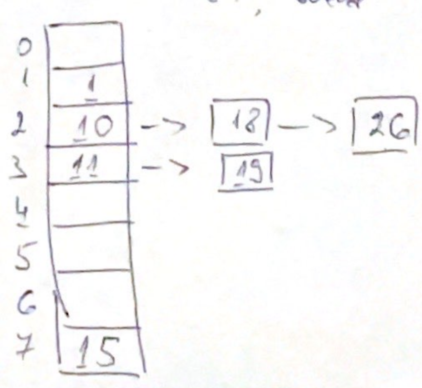
On the left we can see the initial, empty table. Now, if we apply the hash function $h(x)$ on the given set, we'd get:

$$\{10,\ 1,\ 18,\ 15,\ 26,\ 11,\ 19\}$$

$$\Big\downarrow \text{ hash: } h(x) = x \bmod 8$$

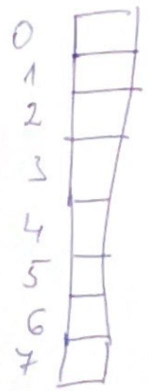
$$\{2,\ 1,\ 2,\ 7,\ 2,\ 3,\ 3\}$$

If we decide to go with chaining to resolve the collisions, then after inserting the elements in the order that they're given to us in the set, ~~which~~ we'd get the following table:



* The arrows show the pointer of the current node in the linked lists.

6) We start with the same initially empty hash table:



When we start filling it we must make sure that if we get a collision we resolve it by looking at the next possible open slot; or essentially using the formula: (hash(x) + f(i)) % Table Size

Let's start filling the table one by one element:

1) $10 \equiv 2 \pmod 8$ => We put 10 in slot 2

2) $1 \equiv 1 \pmod 8$ => We put 1 in slot 1

3) $18 \equiv 2 \pmod 8$ => We have a collision!

=> We try $(18+1) \bmod 8$

$18+1 = 19 \equiv 3 \pmod 8$ => We put 18 in slot 3

4) $15 \equiv 7 \pmod 8$ => We put 15 in slot 7

5) $26 \equiv 2 \pmod 8$ => Collision!

=> We try $(26+1) \bmod 8$

$26+1 = 27 \equiv 3 \pmod 8$ => Collision!

=> We try $(26+2) \bmod 8$

$26+2 = 28 \equiv 4 \pmod 8$ => We put 26 in slot 4

6) $11 \equiv 3 \pmod 8$ => Collision!

=> We try $(11+1) \bmod 8$

$11+1 = 12 \equiv 4 \pmod 8$ => Collision!

=> We try $(11+2) \bmod 8$

$11+2 = 13 \equiv 5 \pmod 8$ => We put 11 in slot 5.

Problem 1 continue:

7) $19 \equiv 3 \pmod 8$ => Collision!

=> We try $(19+1) \bmod 8$

$19+1 = 20 \equiv 4 \pmod 8$ => Collision!

=> We try $(19+2) \bmod 8$

$19+2 = 21 \equiv 5 \pmod 8$ => Collision!

=> We try $(19+3) \bmod 8$

$19+3 = 22 \equiv 6 \pmod 8$ => We put 19 in slot 6.

=> Our final table will look like this:

0	
1	1
2	10
3	18
4	26
5	11
6	19
7	15

c) Here we use the same formula as we did for part b) but we need to change our function $f(i) = i^2$.

=> Our formula here becomes: $(hash(x) + i^2) \%$ Table Size for the i^{th} probe

1) $10 \equiv 2 \pmod 8$ => We put 10 in slot 2

2) $1 \equiv 1 \pmod 8$ => We put 1 in slot 1

3) $18 \equiv 2 \pmod 8$ => Collision! => We try $(18+1^2) \bmod 8$

$18 + 1^2 = 19 \equiv 3 \pmod 8$ => We put 18 in slot 3.

4) $15 \equiv 7 \pmod 8$ $\Rightarrow$ We put 15 in slot 7

5) $26 \equiv 2 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(26+1^2) \bmod 8$

$26+1^2 = 27 \equiv 3 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(26+2^2) \bmod 8$

$26+2^2 = 30 \equiv 6 \pmod 8$ $\Rightarrow$ We put 26 in slot 6

6) $11 \equiv 3 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(11+1^2) \bmod 8$

$11+1^2 = 12 \equiv 4 \pmod 8$ $\Rightarrow$ We put 11 in slot 4

7) $19 \equiv 3 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(19+1^2) \bmod 8$

$19+1^2 = 20 \equiv 4 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(19+2^2) \bmod 8$

$19+2^2 = 23 \equiv 7 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(19+3^2) \bmod 8$

$19+3^2 = 28 \equiv 4 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(19+4^2) \bmod 8$

$19+4^2 = 35 \equiv 3 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(19+5^2) \bmod 8$

$19+5^2 = 44 \equiv 4 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(19+6^2) \bmod 8$

$19+6^2 = 55 \equiv 7 \pmod 8$ $\Rightarrow$ Collision!

$\Rightarrow$ We try $(19+7^2) \bmod 8$

$19+7^2 = 68 \equiv 4 \pmod 8$ $\Rightarrow$ Collision!

d) $f(i) = i * g(x)$ for $g(x) = 5 - x \pmod 5$

1) $10 \equiv 2 \pmod 8$ => We put 10 in slot 2

2) $1 \equiv 1 \pmod 8$ => We put 1 in slot 1

3) $18 \equiv 2 \pmod 8$ => Collision => We try $(h(x) + g(x)) \% TS$

$\Rightarrow [(18 \pmod 8) + (5 - 18 \pmod 5)] \% 8 =$

$= (2 + 2) \% 8 = 4$ => We put 18 in slot 4

4) $15 \equiv 7 \pmod 8$ => We put 15 in slot 7

5) $26 \equiv 2 \pmod 8$ => Collision!

=> We try $(h(x) + g(x)) \% TS$

$\Rightarrow [(26 \pmod 8) + (5 - 26 \pmod 5)] \% 8 =$

$= (2 + 4) \% 8 = 6$ => We put 26 in slot 6

6) $11 \equiv 3 \pmod 8$ => We put 11 in slot 3

7) $19 \equiv 3 \pmod 8$ => Collision!

=> We try $(h(x) + g(x)) \% TS$

$\Rightarrow [(19 \pmod 8) + (5 - 19 \pmod 5)] \% 8 =$

$= (3 + 1) \% 8 = 4$ => Collision!

=> We try $(h(x) + 2g(x)) \% TS$

$\Rightarrow [19 \pmod 8 + 2(5 - 19 \pmod 5)] \% 8 =$

$= (3 + 2 \times 1) \% 8 = 5$ => We put 19 in slot 5

Problem 1 continue:

$\Rightarrow$ We try $(19+8^2) \bmod 8$

$$19 + 8^2 = 83 \equiv 3 \pmod 8) \Rightarrow \text{Collision!}$$

However, since we got through every number from 0 to 8, we can actually make the observation that:

$$(8k)^2 \equiv 0 \pmod 8 \Rightarrow (8k)^2 + 19 \equiv 3 \pmod 8$$
$$(8k+1)^2 \equiv 1 \pmod 8 \Rightarrow (8k+1)^2 + 19 \equiv 4 \pmod 8$$
$$(8k+2)^2 \equiv 4 \pmod 8 \Rightarrow (8k+2)^2 + 19 \equiv 7 \pmod 8$$
$$(8k+3)^2 \equiv 1 \pmod 8 \Rightarrow (8k+3)^2 + 19 \equiv 4 \pmod 8$$
$$(8k+4)^2 \equiv 0 \pmod 8 \Rightarrow (8k+4)^2 + 19 \equiv 3 \pmod 8$$
$$(8k+5)^2 \equiv 1 \pmod 8 \Rightarrow (8k+5)^2 + 19 \equiv 4 \pmod 8$$
$$(8k+6)^2 \equiv 4 \pmod 8 \Rightarrow (8k+6)^2 + 19 \equiv 7 \pmod 8$$
$$(8k+7)^2 \equiv 1 \pmod 8 \Rightarrow (8k+7)^2 + 19 \equiv 4 \pmod 8$$

$\Rightarrow$ The only possible outputs of our formula are the slots 3, 4, and 7 since the quadratic remainders by mod 8 are always 0, 1, and 4 for every $k \in \mathbb{N}$

$\Rightarrow$ The slot will always be full and we <u>cannot</u> include 19.

However, without including 19, the table would be:

0	
1	1
2	10
3	18
4	11
5	
6	26
7	15

Problem 1 continue:

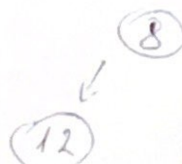
The final hash table is:

0	
1	1
2	10
3	11
4	18
5	19
6	26
7	15

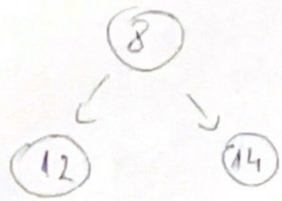
Problem 2:

a) I: Insert 8:

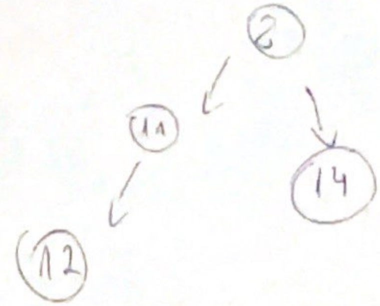
II: Insert 12:



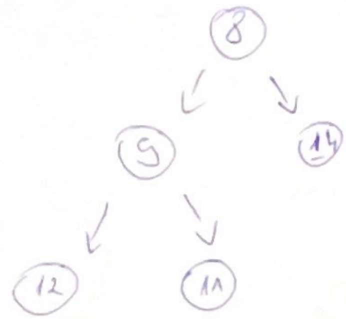
III: Insert 14:



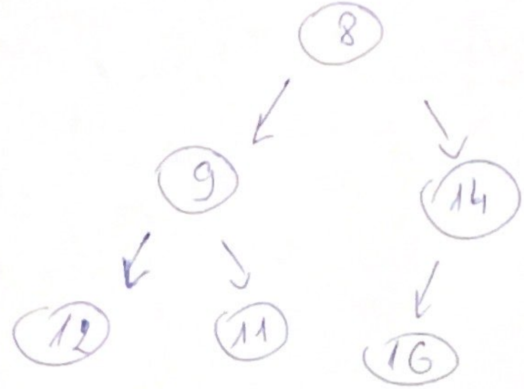
IV: Insert 11:



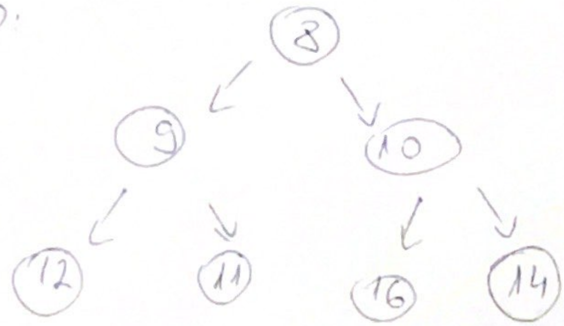
V: Insert 9:



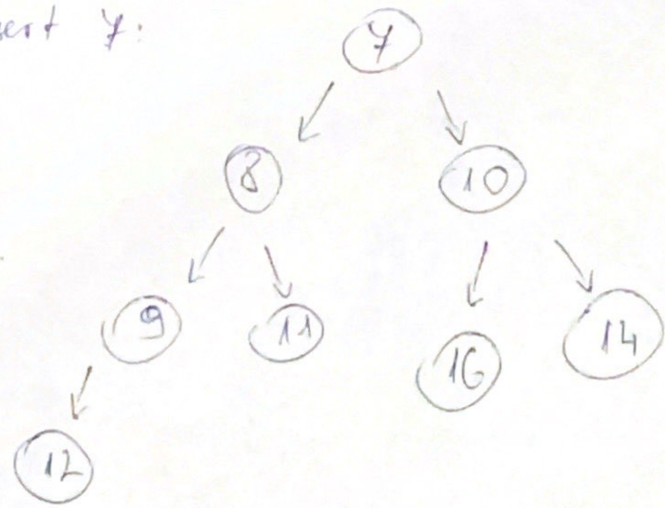
VI: Insert 16:



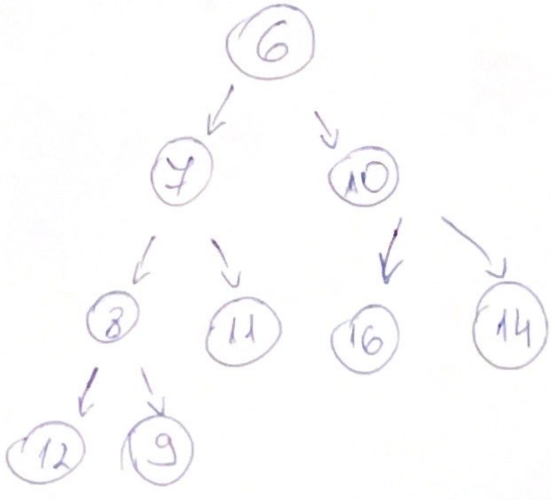
VII: Insert 10:



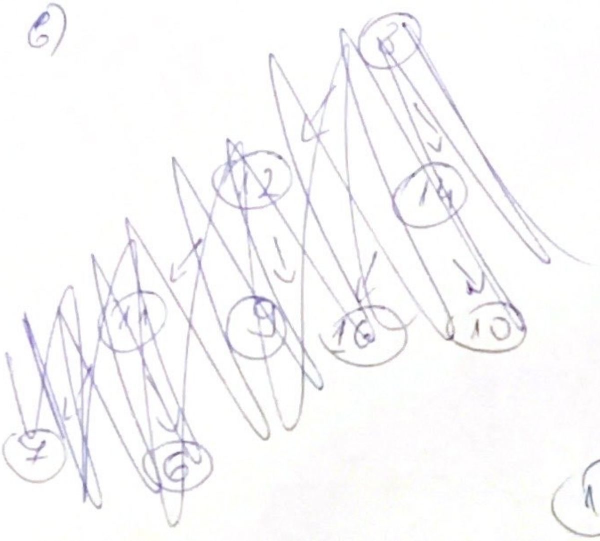
VIII: Insert 7:

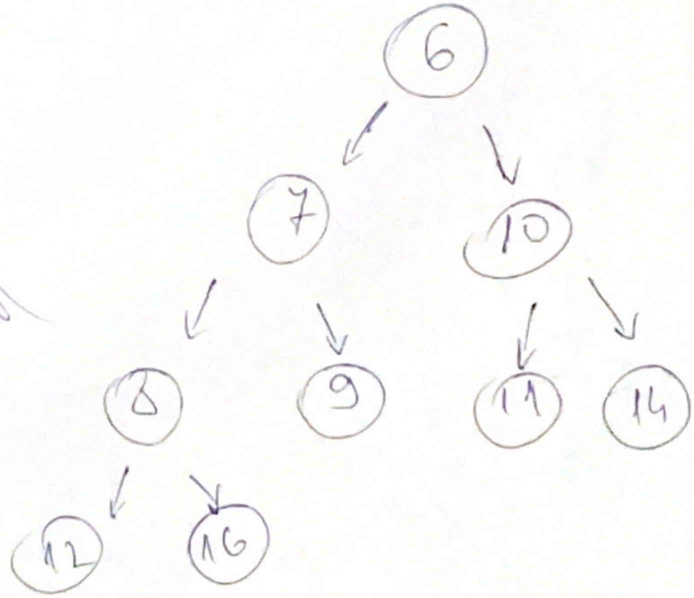


IX: Insert 6:



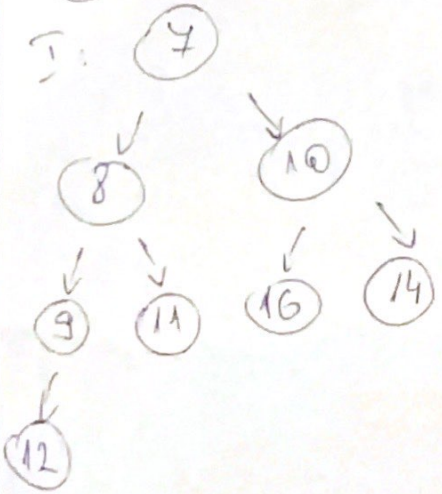
b)



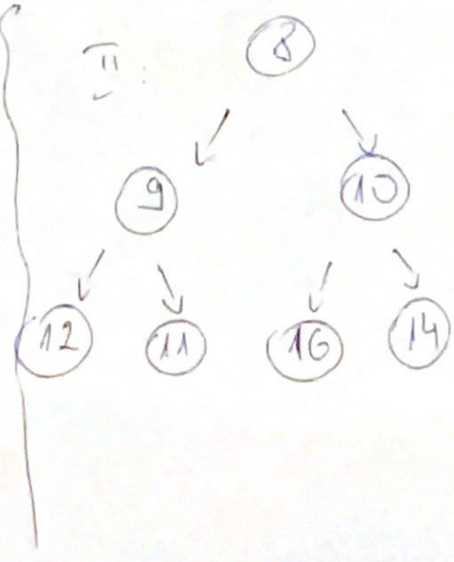


c)

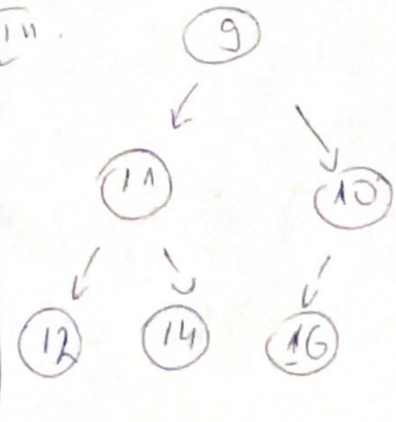
I:



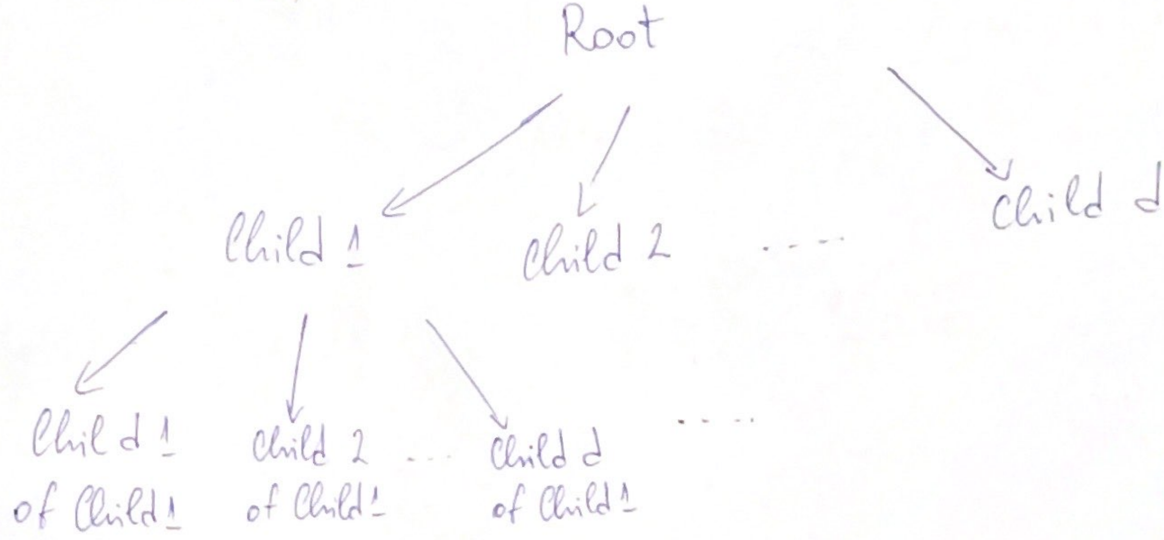
II:



III:



Problem 4:



Parent: $\left\lfloor \frac{i}{d} \right\rfloor$

Children: "Biggest" (a.k.a. the most right) child:

$$id+1$$

"Smallest" (a.k.a. the most left) child:

$$id+1-(d-1) =$$

$$= id-d+2 = d(i-1)+2$$

* Note: Those indicies of children are possible but it's not required for all of them to exist since a d-ary heap may not consist of nodes that all have either 0 or d children.

Problem 3.

a) By the description we can find that the minimal number is always stored at the root of the tree. That's because of the description we're given:

1) If we're at an even depth $(2k)$ for $k \in \mathbb{N}$, then we know that the grandparent's location always holds a lower value. Thus, we know that element$[k]$ < element$[2k]$

However we can continue this relation since we started with an arbitrary k => We know that element$[\lfloor k/2 \rfloor]$ < element$[k]$ < element$[2k]$

If we extend this to the root, we'll see that:

element$[1]$ < element$[2]^*$ < ... < element$[k]$ < element$[2k]$

For the odd positions we know that they are always larger than the parents and since they are odd, then they will always have a parent node, i.e. they will always be larger than something.

=> We can get the min. number by doing:

```
return (this.isEmpty() == false) ? this.array.get(1)
                                  : null
```

Analogously, we can see that for the max number it has to be one of the two children (if the ~~tree~~ array has at least 3 elements) of the root element. That's because the largest element couldn't be the root since we proved that the root is the min number (unless the array has only 1 element) and it couldn't be any other even depth element since all elems stored at an even depth are smaller then their parents. => It has to be an odd depth ~~pos~~ element. However, for odd depth elements, we have ~~has~~ a similar recurrence as we showed for the ~~ove~~ min elem: elements[k] > elements[2k];
elements[k] > elements[2k+1]

⇓ Extending this recurrence, it's easy to see that the largest element should be one of the root's children => In code:

```
if (this.size == 1) {
return this.min; // the root
}
```

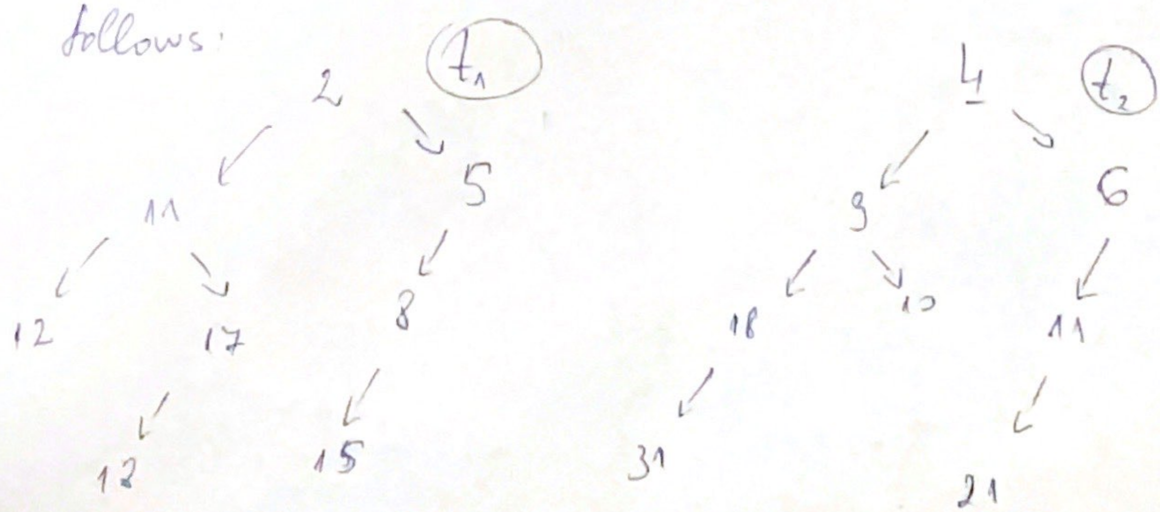
Problem 3 continue:

```
} else if (this. size == 2) {
    return this. array. get (2); //the left child
} else {
    return this. array. get(2). compareTO(this. array. get(3))
        < 0 ? this.array.get (3): this.array.get(2);
// returns the larger value from the root's children
}
```

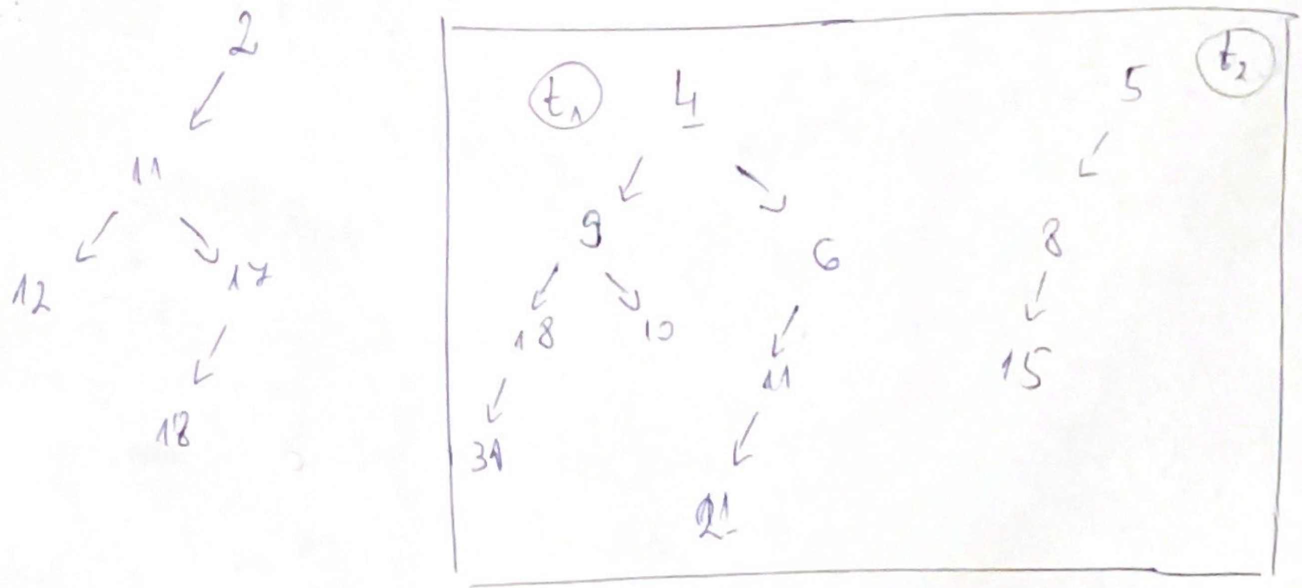
* We could've also chosen elements[3] here and it would've been as equally valid since both children satisfy the desired inequality.

6) Implemented on the last page

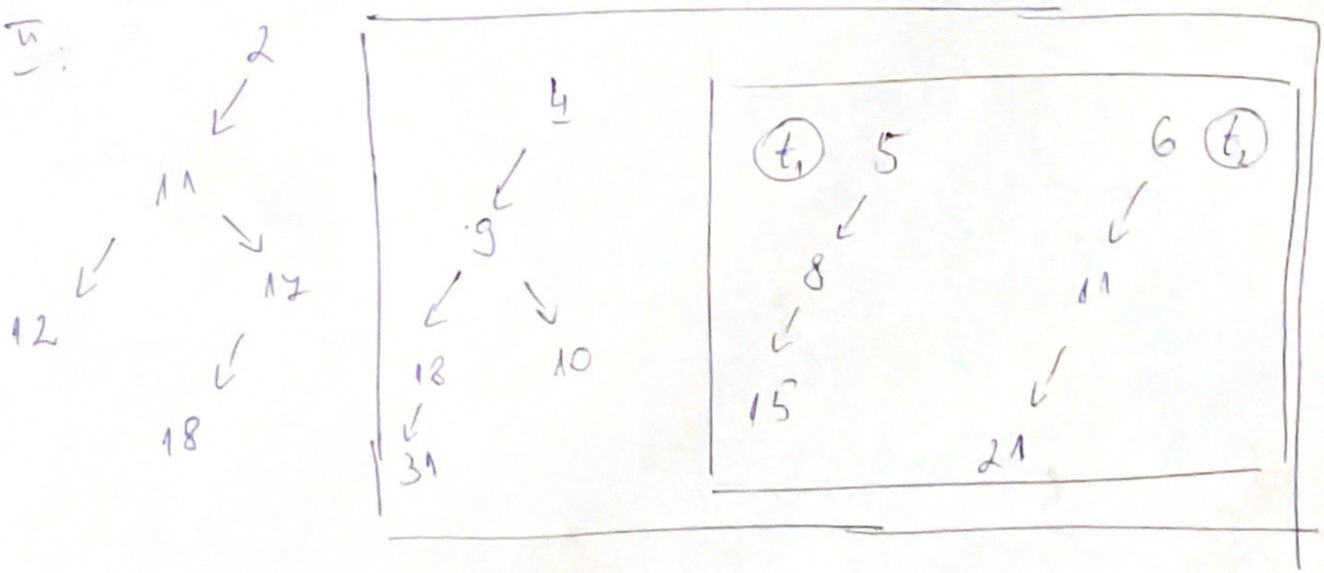
Problem 5. Let's call our trees t_1 and t_2 as follows:



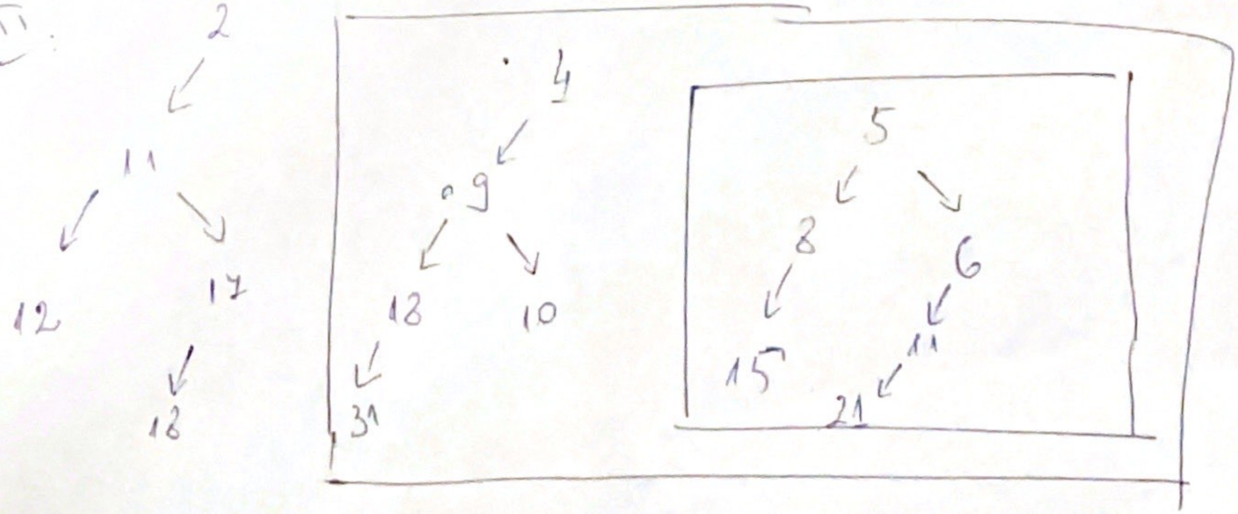
I:


2
11
12
17
18
t_1
4
9
6
18
10
11
31
21
5
t_2
8
15

II:

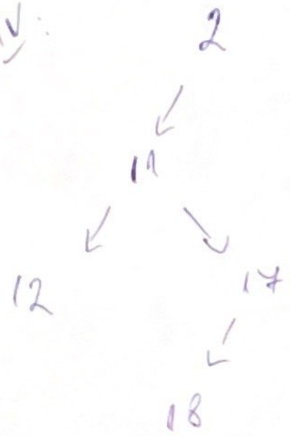

2
11
12
17
18
4
9
18
10
31
t_1
5
8
15
6
t_2
11
21

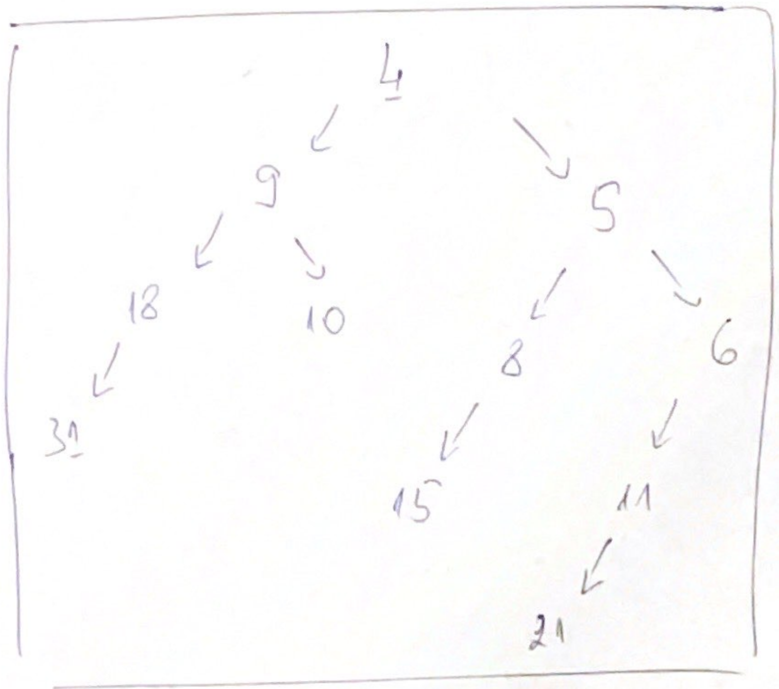
III:


2
11
12
17
18
4
9
18
10
31
5
8
6
15
11
21

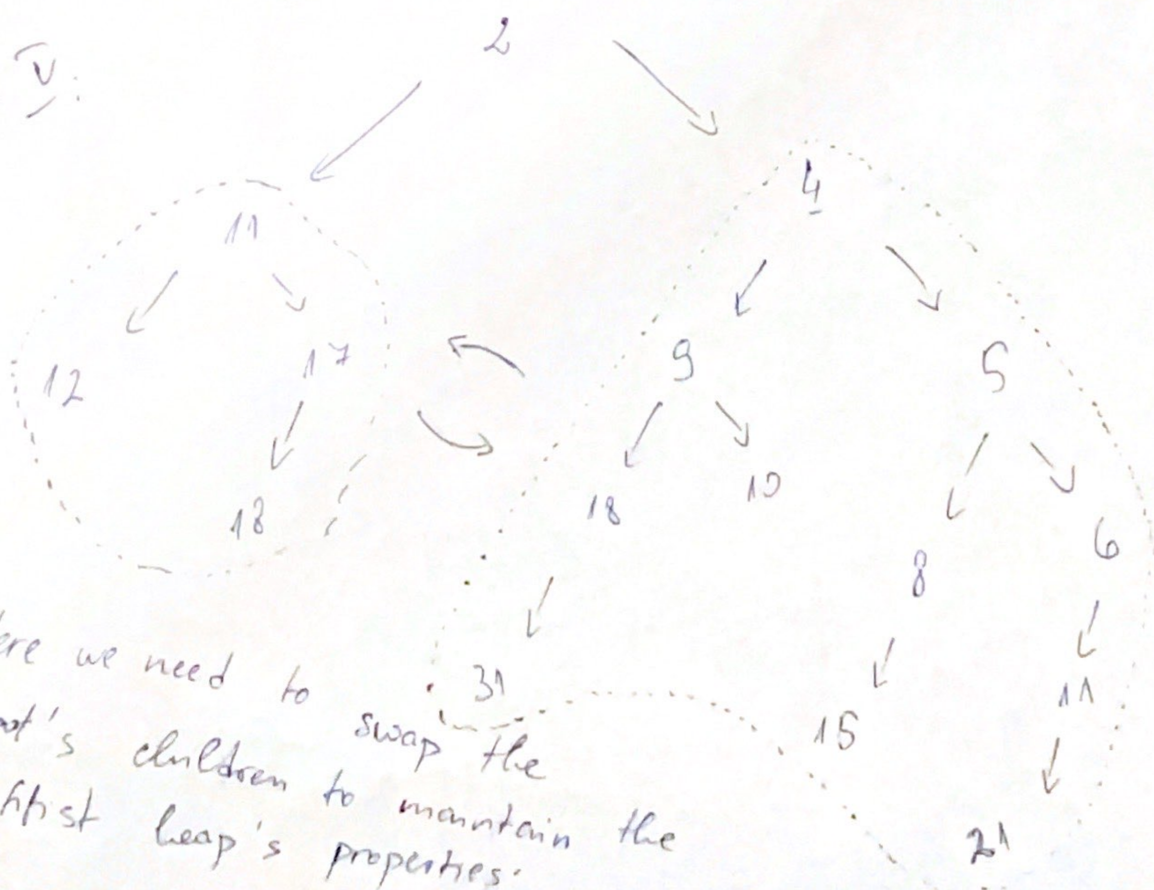
Problem 5 continue:

IV:



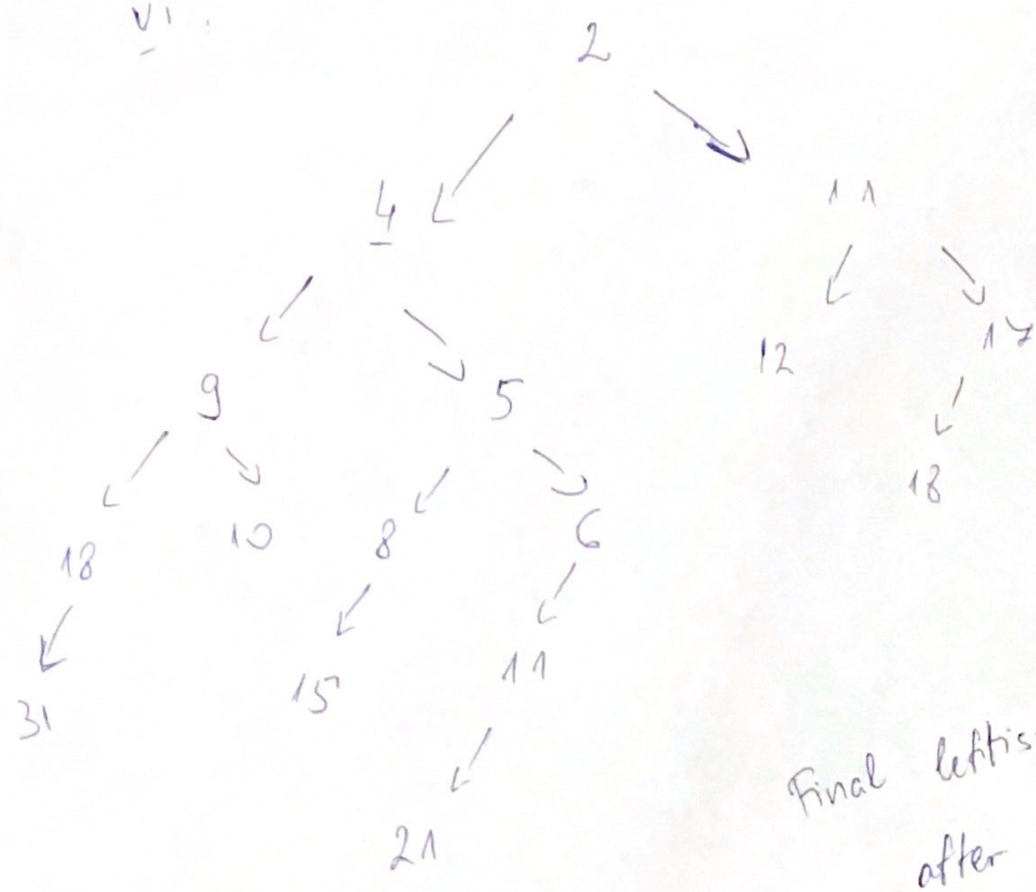


V:



Here we need to swap the root's children to maintain the leftist heap's properties.

VI.



Final leftist heap after merge

```java
    private int getParent(int i) {
        return i >> 1; // getting the floor of division by 2
    }

    private int getGrandparent(int i) {
        return i >> 2; // getting the floor of division by 4
    }

    private boolean hasGrandparent(int i) {
        return i != 1 && i != 2 && i != 3;
    }

    private void pushUpMin(List<T> heap , int i) {
        while(hasGrandparent(i) &&
                heap.get(i).compareTo(heap.get(getGrandparent(i))) < 0) {
            swap(i, getGrandparent(i), heap);
            i = getGrandparent(i);
        }
    }

    private void pushUpMax(List<T> heap , int i) {
        while(hasGrandparent(i) &&
                heap.get(i).compareTo(heap.get(getGrandparent(i))) > 0) {
            swap(i, getGrandparent(i), heap);
            i = getGrandparent(i);
        }
    }

    private void maintainMinMaxHeapProp(List<T> heap, int i) {
        if (!(i == 1)) {
            if (isEvenLevel(i)) {
                if (heap.get(i).compareTo(heap.get(getParent(i))) < 0) {
                    pushUpMin(heap, i);
                } else {
                    swap(i, getParent(i), heap);
                    i = getParent(i);
                    pushUpMax(heap, i);
                }
            } else if (heap.get(i).compareTo(heap.get(getParent(i))) > 0) {
                pushUpMax(heap, i);
            } else {
                swap(i, getParent(i), heap);
                i = getParent(i);
                pushUpMin(heap, i);
            }
        }
    }

    public void insert(T item) {
        if (this.isEmpty()) {
            this.array.add(item);
            ++this.size;
        } else if (!this.isFull()) {
            this.array.add(item);
            this.maintainMinMaxHeapProp(this.array, this.size);
```

```
            ++this.size;
        } else {
            throw new IllegalStateException("Invalid Operation!");
        }
    }
```