Attendance

Capra Paul Ovidiu	Darida Razvan
Curbat Alexandra	Cheran Bianca Paula
Colta Paul Stefan	Demian Ana-Maria
Clapou Alexandru	Cirtorosan Dragos
Chis Sergiu	Craciun Ioan-Flaviu
Deiac David-Mihai	Ciorba Rares-Nicolaie
Cioroga Rares	Craiu Constantin-Tiberiu
Comănac Dragoş-Mihail	
Caravia Andrei	Custura Stefan - Octavian
Copindean Alexandru	
Cosma Eduard	
Comsa Filip-Emanuel	
Carare Claudiu	
Cimpean Andreea	
Ciupe Sergiu-Calin	Neghina Dragos(915)
Condrea Adrian	
	Chis Matei

1. We have an array with n distinct integer numbers. We want to determine the sum of the largest k (k <= n) elements from the array.
   For example: 33, 11, 22, 52, 41, 97, 71,51 and k = 3 => 97 + 71 + 52.

1. sort the array (mergeSort), descendingly and then add the first k elements
   Complexity: Theta(n*log_2n) + Theta(k) => Theta(n*log_2n)

2. using a MAX-heap: add all the elements to the heap and then remove k elements and add them up
   - assume that we have the heap already implemented with the following operations: init, add, remove, getFirst

```
function sumOfK(elems, n, k) is:
        init(h, ">=")
        for i ← 1, n execute
                add(h, elems[i]) // O(log_2n)
        end-for
        sum ← 0
        for i ← 1, k execute
                sum ← sum + remove(h) // O(log_2 n)
        end-for
        sumOfK ← sum
end-function
```

   - Complexity: O(n*log_2n) +O(k*log_2n) => O(n*log_2n)

3. use a Min-heap with only the k largest elements found

```
function sumOfK2(elems, n, k) is:
        inti(h, "<=")
        for i ←  1, k execute
                add(h, elems[i])
        end-for
        for i ← k+1, n execute
                if elems[i] > getFirst(h) then
                        remove(h)
                        add(h, elems[i])
                end-if
        end-for
        sum ← 0
        for i ← 1, k execute
                sum ← sum + remove(h)
        end-for
```

```
        sumOfK2 ← sum
    end-function
```

Complexity: O(k * log_2 k) + O((n-k) * log_2 k) + O(k*log_2 k) => O(n*log_2 k)

4. If we have access to the representation of the heap we can reduce the number of operations a little

   Heap:

```
        elements: TElem[]
        cap: Integer
        size: Integer
        R: relation

    function sumOfK2(elems, n, k) is:
        inti(h, "<=")
        for i ← 1, k execute
            add(h, elems[i])
        end-for
        for i ← k+1, n execute
            if elems[i] > h.elements[1] then
                h.elements[1] ← elems[i]
                bubble-down(h, 1)
            end-if
        end-for
        sum ← 0
        for i ← 1, k execute
            sum ← sum + h.elements[k]
        end-for
        sumOfK2 ← sum
    end-function
```

5. Use a Max-heap, but build it with heapify. And then remove and add the k elements
   Complexity: O(n) + O(k*log_2 n) => O(n+k*log_2n)

2. ADT SortedMap represented on a hash table, collision resolution with separate chaining. How to implement the iterator?

Assume:

- we memorize only the keys from the map
- keys are integer numbers

Ex:

- Map with the following keys: 5, 28, 19,15, 20, 33, 12, 17, 10
- Hash table with m = 9 positions, and a hash function with the division method: h(k) = k mod m

k	5	28	19	15	20	33	12	17	10
h(k)	5	1	1	6	2	6	3	8	1

How to implement the iterator?

- to give the elements in a sorted order
- to be efficient

1. Iterate step-by-step
    - keep a copy of the hash table and the position of the minimum element

    - init Theta(m)
        - create a copy of the hash table (only the first node of each list) and find the position of the minimum element
    - getCurrent Theta(1)
        - return the information from the node from the minimum position
    - next Theta(m)
        - replace the minimum node with its next (if it exists) and find the new minimum position
    - valid Theta(1)
        - check if minimum position has a special value (ex. -1)

```
subalgorithm print(sm) is:
        iterator(sm, itsm) //Theta(m)
        while valid(itsm) execute: //Theta(1)
                e ← getCurrent(itsm)  // Theta(1)
                print e
                next(itsm) //Theta(m)
        end-while
```

end-subalgorithm
Theta(n*m)

2. Build a single sorted singly linked list from the separate chains and iterate over that list

    - getCurrent, next and valid are going to be Theta(1)
    - init ?

merge the linked lists:
2.1 merge list1 with list2, then the result with list3…

SortedMap with n elements
Hash table with m positions
=> the average length of a list is n/m => alpha (load factor)

The merge process:
list1 + list2 => list12 alpha + alpha => 2 alpha
list12 + list3 => list123 2alpha + alpha => 3 alpha
…
list123…(m-1) + listm => list12...m (m-1)* alpha + alpha => m* alpha

Total complexity:
2alpha + 3alpha + …. + m alpha => alpha (2 + 3+...+m) ~alpha * m (m+1)/2
n/m * m * (m+1) /2 => O(n*m)

2.2 merge all lists using a heap with at most m elements.
- initially add the first node of every list in a min-heap
- remove the minimum an if it has a next, add the next to the heap

Total complexity: O(n*log_2m)