Lab 5: Heap

Implement a C++ application for solving the given problem using as data structure a ***heap***. If no further specification is given, use a **binary heap.** You are not allowed to use STL vector or other container/data structure from STL or other libraries for the implementation of the heap.

The problems will be solved in two-member teams (formed during the lab). Each member of the team will receive the same grade for the assignment. **Students not knowing their partner for this assignment are required to contact their lab teacher.**

1. Merge *k* sorted lists into a single sorted list (considering a relation R over the elements). For representing the input lists, use the ***list*** class from STL.
2. Merge *k* sorted vectors into a single sorted vector (considering a relation R over the elements). For representing the input vectors, use the ***vector*** class from STL.
3. Implement **ADT Priority Queue** using a 3-heap (a heap where instead of 2 descendants, every node has three descendants) as representation.
4. Implement **ADT Priority Queue** using a 4-heap (a heap where instead of 2 descendants, every node has four descendants) as representation.
5. Implement a container, called **ADT SecondPriorityQueue** which is similar to a PriorityQueue, but returns and removes the element with the second highest priority (considering a relation R over the priorities). Use a binary heap as representation.
6. Implement a container, called **ADT SecondPriorityQueue** which is similar to a PriorityQueue, but returns and removes the element with the second highest priority (considering a relation R over the priorities). Use a 3-heap (a heap where instead of 2 descendants, every node has three descendants) as representation.
7. Implement a container, called **ADT SecondPriorityQueue** which is similar to a PriorityQueue, but returns and removes the element with the second highest priority (considering a relation R over the priorities). Use a 4-heap (a heap where instead of 2 descendants, every node has four descendants) as representation.
8. Implement a container, calle**d ADT ThirdPriorityQueue** which is similar to a PriorityQueue, but returns and removes the element with the third highest priority (considering a relation R over the priorities). Use a binary heap as representation.
9. Implement a container, called **ADT KPriorityQueue** which is similar to a PriorityQueue, but returns and removes the element with the k^{th} highest priority (considering a relation R over the priorities). Use a binary heap as representation. *Hint: use two heaps, one with a fixed size of k.*
10. Determine the sum of the largest *k* elements from a vector containing *n* distinct numbers with an algorithm having $O(n*\log_2 k)$ complexity. For representing the input vector, use ***vector*** from STL.
11. Remove the smallest *k* elements from a list containing *n* distinct numbers with an algorithm having $O(n*\log_2 k)$ complexity. For representing the input list, use ***list*** from STL.

12. Determine a vector with the first k (k > 0) elements from a vector containing n distinct numbers (considering a relation R). Use a 3-heap (a heap where instead of 2 descendant, every node has three descendants). For representing the input vector use the vector from STL. Do not sort the input vector. If R is “<=”, the first element is the minimum.
13. Remove the last *k* (k > 0) elements from a vector containing *n* distinct numbers (considering a relation R). Use a 4-heap (a heap where instead of 2 descendants, every node has four descendants). For representing the input vector use the ***vector*** from STL. Do not sort the input vector. If R is “<=”, the last element is the maximum.
14. Determine the product of the greatest *k* (k > 0) elements from a vector containing distinct numbers. Use an n-heap (a heap where instead of 2 descendants, every node has n descendants). For representing the input vector use the ***vector*** from STL. Do not sort the input vector.