1 (2). Implement stack via two queues (regardless complexity).

2 (2). The efficiency of Heap is based on the approach of storing an almost-complete binary tree in the array. Provide an approach to storing an almost-complete ternary tree in the array. How one can compute the index of the parent provided the index of the child is given? How to compute the indices of the children provided the index of the parent is given?

3(2). Construct an algorithm that finds the second maximum in a max-heap of size n (stored in the array) with the minimum number of comparisons between the elements. Proof your lower bound.

4 (3). A dynamical array data structure (`std::vector` in C++) can be effectively used for the implementation of Queue (if the static-array approach is not an option). Recall that the deletion of k-th element from the array costs $O(n-k)$ since the shift of the last $n-k$ elements is required after the deletion. So, the naive implementation of Queue via a dynamic array (delete the first element on the extraction) results in $O(n^2)$ algorithm

So, we describe another approach that requires at most $2n$ memory if n elements are in the queue. Denote the array by A. Let b be an integer variable that stores an index of the queue's head. Firstly $b = 0$ points to $A[0]$, but then it increases by one after each extraction (the extracted elements are still stored in A). If after the increment of b the inequality $b \geqslant \lfloor \frac{n}{2} \rfloor$ holds, the algorithm deletes b first elements from the array, and b is set back to 0.

Describe the details of the deletion so that your approach leads to $O(n)$ algorithm, i.e. the amortized cost of each operation is $O(1)$.

5 (2+2). A data structure PriorityQueue stores key-priority pairs and has the following operations:

- `insert`(k, p) — adds the element with the key k and the priority p;
- `extract_max`$()$ — returns a pair (k, p) with maximal p;
- `set_priority`(k, p) sets the priority p to the key k.

Implement PriorityQueue with the following operations' costs:

- `insert` — $O(1)$,
- `extract_max` — $O(n)$,
- `set_priority` — $O(1)$.

1. It is known that all keys are integers from 1 to n.
2. Solve the problem in the general case (you can solve only this part of the problem).

6 (2+2). An array of size n stores keys with repetitions. Assume that a key is (an arbitrarily large) number.

1. Design an algorithm that outputs pairs (k, r) ordered by decreasing of r, where r is the number of occurrences of the key k in the array.
2. Construct an $O(n)$ algorithm for this problem (you can solve only this part of the problem).