Data structures with a direct access

1. Construct an algorithm that verifies whether the sequence of parenthesis is a well-formed parenthesis sequence. E.g. , "(())()" is a well-formed sequence, "())(()" is not.

2. Implement a queue (FIFO) via two stacks so that the amortized cost of an operation is $O(1)$.

3. A function `Max_Heapify`(a, i) resolves collisions in the top-down direction beginning with the tree node corresponding to the i-th element of the array. A function `Build_Max_Heap` is defined as follows:

```
Function Build_Max_Heap (a) :
    n = a.size;
    for i = ⌊n/2⌋ downto 1 do
        Max_Heapify(a, i);
    end
end
```

Demonstrate the run of `Build_Max_Heap` on the input

$$[10,\ 7,\ 6,\ 8,\ 5,\ 4,\ 3,\ 18,\ 16].$$

4. Construct an $O(n + k \log n)$ algorithm that outputs (first) k minimum elements of the array preserving their order (i.e., the first minimum is first the second minimum is second, etc.)

5. 1. Construct a Huffman code for the following frequences (via Heap):

$$a : 0.25,\ b : 0.02,\ c : 0.4,\ d : 0.1,\ e : 0.2,\ f : 0.03.$$

2. Let $\mathrm{len}(x)$ be the length of the code for the letter x and $\Pr(x)$ is its frequency. Prove that if all frequences are of the form 2^{-k}, then

$$\mathrm{len}(x) = \log_2 \frac{1}{\Pr(x)}.$$

6. A k-bit variable is used as a counter that subsequently increases from 0 up to $n = 2^k - 1$. During each increment the variable is changing bitwise, i.e., the only bit is changed during the increment from $00\ldots00$ to $00\ldots01$, but during the increment from $00\ldots01111$ to $00\ldots10000$ five bits are changed. Prove that during the increment from 0 to n the number of operations is $O(n)$ (the coarse analysis gives the bound $O(kn)$). Try using amortized analysis.

7. The input of the problem are numbers n, k and two sequences of integers: $a_1, \ldots, a_n$ and $b_1, \ldots, b_k$. Construct an $O(n + k)$ algorithm verifying that a and b do not share common elements and each of the sequences does not have repetitions as well.

8 [Open-addressing Hash Map]. Define hash functions

$$h_n(x) = x \mod n \quad \text{и} \quad f_k(x) = 1 + (x \mod k).$$

We assume that a key x is (an arbitrary large) positive number. Consider the following implementation of HashTable data structure.

On the preprocessing the algorithm allocates an array a (a hash table) of size M and the functions h_n and f_k are chosen randomly. For the key x on the input the algorithm computes $i = h_n(x) \mod M$. If $a[i]$ is an empty cell, then $a[i] = x$. Otherwise if $a[i] \neq x$, the algorithm computes $j = i + f_k(x) \mod M$ and tests whether $a[j]$ is empty and so on: if $a[j]$ is empty or $a[j] = x$, then $a[j] = x$, otherwise repeat with $j_2 = i + 2f_k(x) \mod M$ until $a[j_m]$ is empty for some m.

1. Let $M = 6$, $n = 5$, $k = 4$ demonstrate the algorithm for the sequence of keys: $7, 12, 2, 22$.

2. How M, n and k should be chosen to guarantee that algorithm fills a completely? I.e., any M keys $x_1, x_2, \ldots, x_M$ will be stored.

3. Let M, n and k be chosen in the proper way. How many operations are needed in the worst case to add a new key? Assume that the table is not full.

Remark. Consider a hash table of size M with m elements. A *load factor* of the hash table is $\alpha = \frac{m}{M}$. We have studied the double hashing algorithm with open addressing. The mathematical expectation (average number of operations) of steps in the array needed for insertion (or retrieving an element) is $\frac{1}{1-\alpha}$. This algorithm is convenient since it requires an array only and no additional data structures. Read more about hashing with open addressing in [CRLS].