Exercises with special data structures

1. Given an array containing n integers, show how to find the largest k elements in the array in $O(n + k \log n)$ time. Given k arrays which are individually sorted in non-decreasing order, show how to merge them into a single sorted array in $O(n \log k)$ time, where n is the total number of elements in all the arrays.

2. Given n tasks such that the ith task has a release time r_i, execution time t_i and deadline d_i. The task can start execution at any time $\geq r_i$, and must finish at or before deadline d_i. There is only one machine available and it can execute at most one task at a time. Other tasks can be kept pending. It is possible to switch from one task to another without any loss of time. A task must be executed for t_i time, but it can be interrupted any number of times. Describe an $O(n \log n)$ time algorithm to determine whether it is possible to complete all tasks before their deadlines.

3. The suffix tree of a string S of length n is a compressed trie that contains all the $n + 1$ suffixes of the string. Usually a distinct `$` symbol is added at the end of the string, so that no suffix is a prefix of another. Each leaf node of the tree is labelled by the starting index of the corresponding suffix. A compressed trie means that nodes that have only one child are eliminated and edges are labelled by substrings rather than single characters. The label is represented by a pair of integers giving the starting position and length of the substring. This reduces the space requirement to $O(n)$. A slightly easier structure to construct is the suffix array, which is a permutation of indices $0, \ldots, n$ in the sorted order of the suffixes. Given the suffix tree, show how the suffix array can be constructed in $O(n)$ time. Constructing the suffix tree in $O(n)$ time is non-trivial. Can you construct the suffix array directly in $O(n)$ time? Another useful array is the lcp array, that gives the longest common prefix between the ith and $(i + 1)$th suffix in sorted order. Show how this can be constructed in $O(n)$ time, from the suffix tree, and also directly.

Given the suffix tree, show how you can test whether a given pattern string p is a substring of S in time $O(p)$. Show how to compute the number of different substrings of S from the suffix tree in $O(n)$ time. Show how to do this using the suffix and lcp arrays, with an extra $O(\log n)$ time in the case of pattern matching.

4. A hash fuction can be used to check for errors in a file that may occur during transmission. Assuming the file is a bit string s, a hash value $h(s)$ is transmitted along with s, and recomputed for the received file. If the values mismatch, it indicates an error. It is assumed there is no error in the hash value received. Describe a simple hash function h such that if the received file differs in exactly one bit, the hash value is guaranteed to be different. Suppose a hash function is required that can detect errors in at most 2 bits. Describe such a hash function. What is the minimum number of bits required in the hash value? Can you get a function that achieves the minimum? One hash function that is actually used is called the md5 check-sum.