Algorithms in your answers can be either explained (in plain text), written in pseudocode or in a programming language amongst C++ or Python. It has no consequence on the notation.

Correctness and Complexity of each algorithm must be justified. If your answer is not justified, then you get at most half of the points. You may also receive up to half of the points if your algorithm is correct but sub-optimal.

The result of a question can be used to answer another question even if you did not manage to answer to it.

Course materials are allowed.

1) Let v be an n-size vector of integers. Show that after a pre-processing in O(n*log(n)) time, we can answer to the following type of queries q(i,j) in O(log(n))-time:
   a. q(i,j): are all elements v[i],v[i+1],….,v[j] pairwise different? **/1**

      Solution 1: We scan the vector once. For each element e present in the vector, we map e to the ordered vector of all positions k such that v[k] = e. It can be done in expected O(n) using a Hash table. Then, if e appears in positions k_1, k_2, …, k_r, we create points (k_1,k_2), (k_2,k_3), …, (k_{r-1},k_r). We put all these points in a 2-range tree, in O(n*log(n)). Now, to answer to q(i,j), it suffices to check whether there exists a point (a,b) where $i \leq a < b \leq j$. It can be done in O(log(n)).
      ***Remark:* this is a variation of ex.5, seminar 7…**

      Solution 2: For every index k let repet[k] be the largest index s < k such that v[k] = v[s], if it exists (else, repet[k] = -1). If can be computed in O(n) by scanning the vector once and storing in an auxiliary Hash table the latest position of each element. Then, we use a partial sum trick. More precisely, let last-repet[k] be the largest value repet[s] for $s \leq k$. It can be constructed in O(n). Now, to answer to q(i,j), it suffices to check whether last-repet[j] $\geq$ i.

   b. q(i,j): is the subvector v[i…j] sorted? **/1**

      Solution 1: Let a[] contain in order all indices k such that v[k] > v[k+1]. It can be computed in O(n) by scanning once vector v. Now, to answer to q(i,j), we compute using binary search the smallest index s such that a[s] $\geq$ i, if it exists, and then we check whether a[s] < j. We output YES if and only if either a[s] does not exist or a[s] $\geq$ j.
      ***Remark:* this is a variation of ex.8, seminar 1…**

      Solution 2: For every index k, let inv[k] denote the number of indices s < k such that v[s] > v[s+1]. It can be constructed in O(n) by scaning the vector v once. Now, to answer to q(i,j), we output YES if either i=j or inv[j] – inv[i] = 0.

      Comment: we could also check whether v[i…j] is either sorted by non-increasing values or sorted by non-decreasing values.

   c. q(i,j): is there some k between i and j such that v[k] is odd? **/1**

      Solution 1: Let b[] contain in order all indices k such that v[k] is odd. It can be

computed in O(n) by scanning once vector v. Now, to answer to q(i,j), we compute using binary search the smallest index s such that a[s] ≥ i, if it exists, and then we check whether a[s] ≤ j. We output YES if and only if either a[s] does not exist or a[s] ≥ j. *Remark:* this is also a variation of ex.8, seminar 1…

Solution 2: We scan the vector once. For every index k, let odd[k] denote the number of indices p ≤ r such that v[p] is odd. It can be constructed in O(n) by scanning vector v once. Now, to answer to q(i,j), we output YES if v[i] is odd or if odd[j]-odd[i] > 0.

d. q(i,j): are there some distinct k and k' between i and j such that v[k]+v[k']=0? **/1**

Solution 1: We scan the vector once. During the scan, we keep in a Hash table the latest position pos(e) found for each element e in the vector (by default, pos(e) = -1). Upon reading v[i] = e, we first create a point (pos(-e),i), then we set pos(e) = i. It takes O(n). We put all these points in a 2-range tree, in O(n*log(n)). Now, to answer to q(i,j), it suffices to check whether there exists a point (a,b) where i ≤ a < b ≤ j. It can be done in O(log(n)). *Remark:* this is a variation of ex.5, seminar 7…

Solution 2: For every index k, let opp[k] denote the largest index s < k such that v[k] = -v[s], if it exists (else, opp[k] = -1). We discussed above how it can be computed in O(n) by scanning the vector v once and using an auxiliary Hash table. Now, for every index k, let last-opp[k] be the largest value of opp[s], s ≤ k. It can also be computed in O(n). Now, to answer to a query q(i,j), we just need to check whether last-opp[j] ≥ i.

2) In what follows, let T be a tree of order n such that we associate to every node x some integer value x.val.
   a. Show that after a pre-processing in O(n*log(n)) time, we can answer to the following type of queries $q_1(x,y,p)$ in $O(\log^2(n))$ time: compute the number of nodes z on the path between x and y so that z.val < p? **/1**
   Hint: use a Heavy-path decomposition…

Pre-processing: First we compute a HP decomposition, and we pre-process T in O(n) for answering to LCA queries (we could also use the naïve LCA scheme in O(log(n)), using the HP decomposition). For each HP, we create points (node.lvl, node.val) for all nodes in it. If we insert those points in 2-range trees, the total runtime is in O(n*log(n)). Note that nodes in different HP are stored in different 2-range trees.

Consider a query $q_1(x,y,p)$. We compute w = lca(x,y). Consider all HP fully between x,w (resp., y,w). For each of them, we count the number of nodes with value at most p. Equivalently, we count how many points (a,b) there are in their respective 2-range trees so that b < p. It can be done in O(log(n)). There may exist HP that are not fully between x,w or y,w (namely, the HP containing x,y,w). For each of them, we first compute the max/min levels L_{min}/L_{max} of nodes on the HP that are between x,w or y,w. We count how many points (a,b) there are in their respective 2-range trees so that $L_{min} \le a \le L_{max}$ and b < p. Overall, since there are only O(log(n)) HP to be considered, the runtime is in $O(\log^2(n))$.

b. Compute in O(n*log(n)) time a set S of O(√n) nodes such that *every* path of length at least √n must contain at least one node of S. **/1**
Remark: if T is a path, then we can partition it in O(√n) sub-paths, and then we can define S as containing the ends of every sub-path…

We edge-partition T in O(√n) subtrees of O(√n) nodes by repeatedly computing a centroid node. We insert in S the roots of all subtrees. *See also seminar 7, ex. 6.a and 6.b.*

In the next three questions, we consider the following type of queries $q_2(x,y)$: compute the number of pairs (w,z) such that `(i)` w,z are on the path between x and y, `(ii)` dist(x,w) < dist(x,z), and `(iii)` w.val > z.val.

c. Show that *for a fixed node* x, we can compute in O(n*log(n)) time the values $q_2(x,y)$ *for every node* y. **/1** Simpler variant: O(n* $\log^2$(n)) **/.75**

We root T at x and then we start a DFS at x. When we visit a node y for the first time (preorder), we add y in a self-balanced binary search tree with key y.val. When we visit a node y for the last time (postorder), we remove the corresponding element in the BST. In doing so, whenever we visit a node y, the only nodes in the BST are on the path between x and y. We compute in O(log(n)) time the number y.inv of such nodes with a larger value than y.val (range query). We end up using the formula $q_2(x,y) = q_2(x,y.father) + y.inv$.

d. Show that after a pre-processing in O(n) time, we can compute $q_2(x,y)$ in O(√n*log(n)) time for every two nodes x and y such that dist(x,y) ≤√n. **/1**

After a pre-processing in O(n), we can compute any lca in O(1). In particular, we can enumerate in order all nodes on the xy-path in O(√n). We insert all associated values in a vector, of which we compute the number of invertions.

e. Deduce the following result from the four previous questions: after a pre-processing in O(n*√n *log(n)) time, we can compute $q_2(x,y)$ in O(√n*$\log^2$(n)) time for every two nodes x and y. **/1**

Let us pre-compute S (see 1.b) and, for each node s in S, let us pre-compute all values $q_2(s,y)$ (see 1.c). Furthermore, we pre-compute the levels of every node in the tree. Doing so, by computing the LCA of two nodes, we can also compute the distance between these two nodes.

To answer to a query $q_2(x,y)$, we do as follows.
***Case* 1. If dist(x,y) ≤√n, then we apply 1.d.**
***Case* 2. If dist(x,y) > √n, then we enumerate in order (starting from x) the nodes on the xy-path until we find some node s in S (see 1.b). Recall that we pre-computed $q_2(s,y)$ (see 1.c). In order to compute $q_2(x,y)$ it suffices to compute:**
- the number of invertions on the xs-path (for that, see 1.d)
- and for each node w on the xs-path, the number of nodes z on the sy-path with a smaller value (see 1.a).

3) Recall that, given a sequence $\mathcal{S}$ of m operations on a disjoint-set data structure, the graph $G(\mathcal{S})$ has a vertex x for each `makeset(x)` operation, and an edge xy for each `union(x,y)` operation. Show that, if $G(\mathcal{S})$ is a cycle, then we can execute all m operations in $\mathcal{S}$ in O(m) time. – Hint: reduce to a sequence $\mathcal{S}'$ such that $G(\mathcal{S}')$ is a tree… **/1**

We ignore the last union operation (which is useless because at this point all elements are already in the same set). In doing so, the graph associated to this smaller sequence is a path.