Algorithms for Massive Data Sets

Student name and id: Alexander R. Johansen (s145706)

Collaborator name(s) and id(s): Jonas (s142957)

Hand-in for week: 6

1 Sort problem

The Sort(i, j) Problem can be solved in $O(nlgn)$ space using a 1-D range over the indexes of A, with an auxiliary sorted version of A in every node(range). The query time, in a format of $f(n) + occ * g(n)$ can be solved in $f(n) = O(lgn)$ and $g(n) = lglgn$ which gives an algorithm that runs in $O(lgn + occ * lglgn)$ that validly can solve the "sort" problem.

1.1 Analysis of datastructure and algorithm

Suppose a 1D range tree over the indexes of A, in which each node holds a sorted list of the elements within the index's range of a given node.
Using the range query on $range(i, j)$, in $O(lgn)$ time, the leaves hanging inbetween (at max $O(lgn)$ leaves) each contains a sorted list of the ranges of the hanging leaves. Suppose the "merge" part of the "merge-sort" subroutine is used to merge these "sorted" lists, such that at completion, a complete sorted list from $A[i, ..., j]$ is created.

1.2 Specific data structure

Given a list/array A
1. Compute a 1D range tree over A using indexes as keys, with a sorted list of the elements in the index range at each node. Space: $O(nlgn)$

1.2.1 Evaluation data structure

Step 1: $O(nlgn)$
Total: $O(nlgn)$

1.3 Specific algorithm

Given two indexes, i, j
1. Run the range(i,j) algorithm(lgn time), collecting all hanging leafs containing sorted lists of the A indexes in the given hanging nodes. Such that at max lgn lists is collected.
time: $O(lgn)$
2. Use the merge part of the merge-sort subroutine, merge the sorted lists from step 1
time: $O(occ * lglgn)$*
* Reason for lglgn is that; seeing as only lgn amount of sorted lists can be extracted from step 1. The merge subroutine halves the amount of lists at every step, thus only lg og lhn steps can occur, and with occ(linear) amount of work at every step, it thus becomes O(occ*lglgn)
time: $O(occ * lglgn)$

1.3.1 Evaluating algorithm

Step 1: $O(lgn)$
Step 2: $O(occ * lglgn)$
Total: $O(lgn + occ * lglgn)$

1.3.2 Extra

Using the different size of the sorted lists, it might be of interest which lists to merge in which order, e.g. by merging the smaller lists before merging the larger lists, however, I haven't been able to arrive at a mathematical bound for such algorithm yet.