Problem 1.

Consider a B-tree with a branching factor of $B$ implementing a set of size $N$. Suppose that, at a particular time while running the database, you have loaded into memory one particular path of blocks from the room to one leaf, plus all the sibling blocks to nodes on that path. Assume that each node is implemented in memory as a sorted array of keys and pointers to descendant nodes. Assume that $B$-trees are of the form discussed in class (and in Siegel) where all the values are in the leaves, and are repeated as needed in internal nodes. Assume for simplicity that the same value $B$ is both the maximal number of children of an internal node and the maximum number of values in a leaf node.

A. What is the worst-case order-of-magnitude running time of searching for an element as a function of $N$ and $B$? How about adding an element? Explain your answers.

B. Consider the following scenario. There is a rule, to guard against crashes, that whenever an in-memory image of a block of the B-tree is modified, that block must be written out to disk. In some particular situation, a set is created by carrying out a sequence of $N$ Adds (no Deletes). After adding $N$ elements, the B-tree is still actually small enough to fit in main memory, so the only disk accesses that have occurred are the writes required by the above rule. Compute the number of disk writes carried out in total as a order-of-magnitude function of $N$ and $B$.

Hint: Splitting a node or transferring children between siblings requires two writes to disk. Adding an element to a leaf node or adding an arc to an internal node without other restructuring requires one write to disk. Therefore when an element is added to the set, that requires one or two leaf nodes to be written out. When an arc is added to the tree, that requires one or two internal nodes to be written out. When a new root is created, two arcs are created but only one node (the root) is written out.

C. Suppose that in (B) the items are added to the set in increasing order and $N = B^k$. Thus, at the end, the tree is the fully branching tree of height $K$, where every internal node has $B$ children and every leaf has $B$ values. Give an exact count of the total number of disk writes performed.
Problem 2

Describe a compound data structure for a set of integers of arbitrary size that supports the operations listed below with the given running times. (A single data structure must support all these.) You may assume that you are given an upper bound on the possible size of the set $|S|$. Give a brief description of what is involved in each operation. You may assume the data structures that have been described in class. (Thus, for instance, if you use a modified heap, you may say “Use the insert operations on heaps,” without detailing what that involves.)

Add($x$, $S$) – Worst case time $O(\log(|S|))$
Delete($x$, $S$) – Worst case time $O(\log(|S|))$
Member($x$, $S$) – Avg. time $O(1)$. Worst case time $O(\log(|S|))$
Min($S$) – Worst case time $O(1)$.
Max($S$) – Worst case time $O(1)$.
Next($x$, $S$) – Given an element $x$ of $S$, find the next larger element in $S$. Avg. time $O(1)$.
   Worst case $O(\log(|S|))$
Pred($x$, $S$) – Given an element $x$ of $S$, find the next smaller element in $S$. Avg. time $O(1)$.
   Worst case $O(\log(|S|))$.
Kth($k$, $S$) – Returns the $k$th smallest element in $S$. Worst case time $O(\log(|S|))$.
IndexOf($x$, $S$) – Returns the value of $k$ such that $x$ is the $k$th smallest element.
   Worst case time $O(\log(|S|))$. 