**m-way Mergeable Trees**

*m-way Mergeable Tree*: Empty, or if not empty then the root is an internal or an external node:

- Each internal node has \( n \) (\( 2 \leq n \leq m \))
  - children denoted by \( p_0, p_1, \ldots, p_{n-1} \), each \( p_i \) points to an internal or external node;
  - \( n \) pairs of values \((s_0, l_0), (s_1, l_1), \ldots, (s_{n-1}, l_{n-1})\), where \( s_i \) is the smallest element stored in the mergeable tree pointed at by \( p_i \), and \( l_i \) is the largest element in the mergeable tree pointed at by \( p_i \).

- Each external node has \( n \) (for some \( 1 \leq n \leq m \)) values, denoted by \( v_0, v_1, \ldots, v_{n-1} \).

In addition, every node (internal or external) has a pointer to its parent and the value of \( n \). The parent of the root of a tree is Null.
Example: m-way mergeable Tree

The letters are "pointers"

```
a: 3, 0, (3, 9, b), (2, 2, d), (1, 14, f)
b: 2, a, (3, 7, c), (9, 9, e)
c: 2, b, (4, 7, g), (3, 6, h)
d: 1, a, (2)
e: 1, b, (9)
f: 2, a, (5, 14, i), (1, 11, c)
g: 2, c, (4), (7)
h: 2, c, (3), (6)
i: 2, f, (10), (11)
j: 2, f, (12, 13, e), (5, 14, k)
k: 2, j, (14), (5)
l: 1, j, (12)
```
A mergeable B’-Tree of order \( m \) is an \( m \)-way mergeable tree. If the mergeable B’-tree is not empty, the corresponding tree satisfies the following properties:

- If the root is an internal node, then it at least two children.
- All internal nodes (other than the root) have at least \( \lceil m/2 \rceil \) children.
- If the root is an external node, then it at least one value.
- All the external nodes (other than the root) have at least \( \lceil m/2 \rceil \) values.
- All external nodes are at the same level.
Mergeable $B'$-Tree example $m = 4$

Not a mergeable $B'$-Tree

Problem: External nodes at different levels.
INSERTION CAN BE AT ANY EXTERNAL NODE

Insert \[\rightarrow\] \[10 \ 31\]

\[\{10 \ 31\} + \{18\} \Rightarrow \{10 \ 31 \ 18\}\]

Insert \[\rightarrow\] \[10 \ 31 \ 18\]

\[\{10 \ 31 \ 18\} + \{19\} \Rightarrow \{10 \ 31 \ 18 \ 19\}\]

overflow

Split
Deletion

Delete

2 10 \Rightarrow 2 30

Delete

Borrow from closest sibling

2 \Rightarrow 9 10

\Rightarrow 9 6

\Rightarrow 40 2
Combine with sibling
+ borrow from parent

Parent combines with parent's sibling and borrows from grandparent.
Mergeable B'–Trees

Delete

Combine with sibling and borrow from parent.

Parent borrows from parent's sibling.

Delete from closed sibling
Delete 9

Combine with sibling + borrow from parent.
Parent combines with parent's sibling + borrows from grandparent.
Since root has one child, the child becomes the root.
Merge

\[ m = 3 \]

New child for root
Merge $m = 3$

New child for root
But then root has too many children
So split the root
Final Exam Question

Answer to question gives you a (huge) hint on how to implement of Findk so that it takes $O(k \log k)$ time for the $B - DS$ part.

Let $n$ be a positive integer and let $Z = (z_1, z_2, ..., z_n)$ be an array of integers. The array $Z$ is a min-heap with $n$ elements (i.e. $z_1$ contains the smallest value in the heap). Let $k$ be any positive integer whose value is less than $n$. Consider the following procedure (to be executed as a C++ program after translating it to C++ code).
\[
\begin{align*}
\{ \ W & \leftarrow \ \{(z_1, 1)\} \quad \text{/* } W \text{ is a set of 2-tuples */}\ \\
& \text{for } i = 1 \ \text{to } k \ \text{do} \\
& \quad \{ \ (x, y) \leftarrow \text{the 2-tuple in } W \text{ whose first} \\
& \quad \text{component has the smallest value; } \\
& \quad \text{delete from } W \text{ the 2-tuple } (x, y); \\
& \quad \text{if } 2y \leq n \text{ then} \\
& \quad \quad \text{add } (z_{2y}, 2y) \text{ to set } W; \\
& \quad \text{if } 2y \leq n+1 \text{ then} \\
& \quad \quad \text{add } (z_{2y+1}, 2y+1) \text{ to set } W; \\
& \} \\
& \text{print } x;
\end{align*}
\]
a) [5 Points] What value does the above program print when $Z = (3, 4, 7, 11, 13, 15, 19, 21, 14, 19, 20, 17, 22, 30, 40)$ and $k = 5$.

b) [8 Points] Is it possible for $W$ to contain more than $k + 1$ tuples? Why or why not?

c) [7 Points] To receive full credit for this part, explain how to represent set $W$ so that the overall time complexity of the above algorithm becomes $O(k \log k)$. 