Problem 1: Medians [20 pts]

There is a well known linear-work algorithm for finding the median of a set of values (the median value is the value \( v \) such that if the values are sorted, \( v \) would be in the middle). In this question you don’t need to understand why the algorithm works, but you need to answer some questions about it and analyze its costs based on a description of its steps:

(a) If the input has 5 or fewer values, find the median by brute force, otherwise:
(b) Group the input into \( n/5 \) groups of 5 and find the median of each group in parallel.
(c) Find the median of the medians recursively. Call this \( p \).
(d) Use \( p \) to filter out \( 3/10^\text{th} \) (s) of the values in \( \Theta(n) \) work and \( \Theta(\log n) \) span.
(e) Recurse on the remaining \( 7/10^\text{th} \) (s) of the values.

Questions:

(a) [8 pts] Write down recurrences for work and span.
(b) [5 pts] Based on your recurrence, is the work linear (i.e. \( \Theta(n) \))? Please explain.
(c) [7 pts] Based on your recurrence, calculate the span in terms of \( \Theta \).
Problem 2: Parallel Merging [30 pts]

For simplicity, in the following problems you can assume all keys are distinct.

(a) [15 pts] Assume you are given two arrays of size $n$ and $m$, each with keys in sorted order, and an integer $k, 0 \leq k \leq n + 1$. Let $x_k$ denote the $k$th smallest element out of the $n + m$ total elements. Describe an algorithm that returns the cut point in each array such that all elements below the cut point are less than or equal to $x_k$, and all elements above are greater than $x_k$. It must run in $O(\log n)$ work. To make it simple, you can assume the keys are unique.

(b) [15 pts] Design an algorithm for merging two arrays, each of length $n$, in $O(n)$ work and $O(\log n)$ span. You might find part 1 useful.

Problem 3: Euler Tours [40 pts]

In class we described how to do list ranking in linear work and $O(\log n)$ span. An Euler tour of a rooted tree is a path that starts at the root and goes around the tree visiting each edge twice, once on the way down and once on the way up as in the following figure:

Assume that the Euler tour is represented as a linked list, where each link consists of a pointer to the tree node below the edge (e.g. 1 and 8 both point to tree node 1 above) and a pointer to the next list element. The tree nodes consist of just a value and a pointer to their parent.

(a) [15 pts] Using list ranking, describe how to calculate for every node the sum of values stored at and below it. Your solution must run in linear work and $O(\log n)$ span. You can assume list ranking runs in linear work and $O(\log n)$ span and that an Euler tour (in the form described above) is already created.

(b) [25 pts] Your second task is to “line break” in parallel. You are given a sequence
of integers $W$ representing word lengths in a paragraph, and a line length $L$, with the condition that all word lengths are at most $L$. Your job is to describe a parallel algorithm that generates the same output $B$ as the following serial code.

```c
int s = W[0];  B[0] = 1;
for (i=i; i < n; i++) {
    s += W[i];
    B[i] = 0;
    if (s > L) {
        B[i] = 1;
        s = W[i];
    }
}
```

In particular this code simply marks in $B$ the start of every new line when filling text up to width $L$ (for simplicity, we are ignoring the width of the spaces). Your algorithm must use at most linear work and $O(\log n)$ span. You cannot assume $L$ is a constant (i.e. the work and span must be independent of $L$). You might find the two problems you already solved (2 and 3(a)) useful. Please make your solution as clear as possible.

**Problem 4: Buffered B-tree [30pts]**

Consider the following mix of a B-tree and buffer tree supporting insertions and searches. Each internal node has $\Theta(B^\epsilon)$ children, for some constant parameter $0 < \epsilon < 1$. Moreover, each node has a buffer containing $\Theta(B)$ objects. A node (the buffer, child pointers, and pivots partitioning the children) is stored contiguously in 1 block.

To insert an object in the tree, first insert it into the root buffer. Partition that buffer according to the pivots. While any partition contains at least $\Omega(B^{1-\epsilon})$ objects, recursively insert $\Theta(B^{1-\epsilon})$ objects into the appropriate child. When a leaf node becomes full, split it, inserting a new child pointer into its parent. If an internal node has the maximum number of children, split it into two internal nodes, inserting a new child pointer into the parent. Each split is implemented as in a buffer tree and costs $\Theta(\text{nodesize}) = \Theta(1)$ block transfers.

To search for an object, follow the appropriate root-to-leaf path according to the pivots, and also search the entire buffer within each node along the path.

For parts (a)–(e), justify your answer with a brief proof sketch (at most a few sentences).

(a) [5 pts] What is the height of a tree containing $N$ objects?

(b) [5 pts] Assuming the leaf node is already in memory, what is the amortized cost of
inserting an object into a leaf (i.e., the cost of performing splits up the tree)? More specifically, by “amortized cost” here, we mean suppose you start with an empty tree and perform a sequence of \( N \) insertions directly into the appropriate leaves (without moving down the tree through buffers). What is the average cost of each insertion, assuming that the relevant leaf is already in memory?

(c) [5 pts] What is the amortized cost of moving an object down the tree to a leaf buffer? Assume the root node is always in memory. Specifically, suppose you start with a tree containing \( N \) objects in which all buffers are empty; then perform a sequence of \( N \) insertions into the tree. What is the average cost of each insertion, assuming that node splitting happens for free?

(d) [5 pts] What is the amortized cost of an insert starting from an empty tree? That is, when performing a sequence of \( N \) insertions starting from an empty tree, what is the average insertion cost?

(e) [4 pts] What is the cost of a search?

(f) [3 pts] The insertion cost in (d) is usually worse than the \( O(\frac{1}{B} \log_{M/B} \frac{N}{B}) \) amortized insertion cost for a buffer tree. What is the advantage of the buffered B-tree?

(g) [3 pts] When should a buffered B-tree be used instead of a regular B-tree?