Due: March 7th (Thursday), 6pm, submit to courseworks.

You may use "standard" arguments without a formal proof—either cite from class or add one sentence showing the main idea.

Problem 1: Van Emde Boas insertion

Recall the recursive definition of vEB trees discussed in class. Here's the pseudo code for the Search operation $\text{PREDECESSOR}(x, S)$ that returns x's predecessor within the set S:

```
procedure PREDECESSOR(x, S)
    if x ≤ S.min then
        return ∅
    if x > S.max then
        return S.max
    if low(x) > S[high(x)].min then
        return high(x)√u + PREDECESSOR(low(x), S[high(x)])
    else
        i ← PREDECESSOR(high(x), S.summary)
        return i · √|S| + S[i].max
```

Given this code, show that this data structure facilitates dynamic *updates* of inserting keys to S in worst-case $O(\lg w) = O(\lg \lg u)$ time, by completing the pseudo code for $\text{INSERT}(x, S)$.

Problem 2: Fast predecessor search for monotone-interval sets

1. (Evenly-spaced intervals) Suppose we are given set of *evenly spaced* numbers $S = x_1 \leq \cdots \leq x_n$, i.e. such numbers that there exists a k for which it holds for all $i = 1, \ldots, n-1 : k \leq x_{i+1} - x_i \leq 2k$. Build a linear-space static data structure that answers PREDECESSOR queries on such sequences in $t = O(1)$ time in the RAM model with word size $w = O(\log n)$.
   (You may use as a black-box the existence of a DICTIONARY data structure, that stores any vector $x \in \Sigma^n$ with d nonzero entries using space $O(d \lg |\Sigma| \lg n)$ and retrieves x_i in constant time $t = O(1)$).
2. (Increasing intervals) Do the same for *increasing intervals* sets $x_1 \leq \cdots \leq x_n$, in which, for all $i = 1, \ldots, n-2 : x_{i+1} - x_i \leq x_{i+2} - x_{i+1}$. (Hint: Reduce to (1) using fusion trees).

Problem 3: Predecessor applications

Show how to statically solve following problems using PREDECESSOR or SUCCESSOR data structure.

1. (1D range reporting) Suppose that for a given list of integers $x_1, \ldots, x_n$ you want to answer such queries: given an interval $[a, b]$ return all $x_i \in [a, b]$. $s = O(n)$ space, $t = O(\log \log n + \text{occ})$ time, where occ is the number of integers in the answer.

2. (Subsequence search) Given a string $T \in \Sigma^*$, for any string $P \in \Sigma^*$ determine whether T contains P as a subsequence, i.e. whether there is an increasing sequence of integers $i_1 < \ldots < i_{|P|}$, such that $P_j = T_{i_j}$ for all $j \in \{1, \ldots, |P|\}$. $s = O(|T| + |\Sigma|)$ space, $t = O(|P| \log\log |T|)$ time.

Problem 4: $O(1)$ predecessor search for near-linear universes

Recall that Van Emde Boas trees solve Predecessor search in a universe of size u with n keys in $t = O(\log\log u)$ time and linear space $s = O(n)$ words, assuming word size $w = O(\log u)$.

We will now show that this running time can be slightly improved for *small* universes $[u]$. Specifically, show that using s words of space (where word size is w bits), we can achieve static search time of $t = O((\log u - \log n)/a)$, where $a := \log(s/n) - \log w$. Conclude that for near-linear universes $u = n \cdot \mathrm{polylog}(n)$, Predecessor search can be done with $s = O(n)$ space, and constant $t = O(1)$ running time (with the standard word-size $w = O(\log n)$).