Homework Assignment #1
Due: May 19, 2023 at 10:00 p.m.

[2] **1.** How many different binary search trees are there that contain the keys $\{1, 2, 3, 4\}$? Briefly justify your answer.

[2] **2.** Which nodes of a binary min-heap containing 20 distinct values could contain the third-smallest value? Briefly justify your answer.

[5] **3.** In class, we said a set $S \subseteq \{1, 2, \ldots, N\}$ could be represented as a bit vector, that is, an array of N bits, where the ith bit is 1 if and only if $i \in S$. This allows us to insert, delete and lookup values in the set in $O(1)$ time. However, initializing the bit vector to represent an empty set S could be expensive. If an array is allocated simply by reserving an appropriately-sized chunk of memory, the array may contain arbitrary garbage values (since that section of memory might have been used previously to store other information before being reallocated to be used for the array B). Going through the array to set each location's value to 0 would take $\Theta(N)$ time, which is expensive. Here, we consider a way of initializing the data structure in $O(1)$ time while still being able to do insert, delete and lookup in $O(1)$ time, at the cost of using more space.

We shall use two arrays $B[1..N]$ and $Written[1..N]$ of integers and an integer variable $NumWritten$. Each of the two arrays may initially contain arbitrary garbage values. $Written[1..NumWritten]$ will store the indices of entries of B that have been updated since the data structure was initialized. So, when we first write to an entry $B[i]$, we must add i to the array $Written$. To see if $B[i]$'s value is valid (as opposed to garbage), we must check if i is in $Written[1..NumWritten]$. To find i in $Written$ efficiently, $B[i]$ will store the index of i in $Written$ (together with a plus or minus sign). The plus or minus sign will represent whether i is in the set S or not.

More formally, we wish to maintain the following invariants.

- If an INSERT(i) or DELETE(i) has occurred since the data structure was initialized, then
  - $|B[i]| \leq NumWritten$ and $Written[|B[i]|] = i$, and
  - $B[i] > 0$ if and only if $i \in S$.
- If an INSERT(i) or DELETE(i) has *not* occurred since the data structure was initialized, then either $|B[i]| > NumWritten$ or $Written[|B[i]|] \neq i$.
- The values in $Written[1..NumWritten]$ are distinct.

Consider the following code. You can assume that the argument i is an integer from $\{1, 2, \ldots, N\}$. Fill in the missing bits of the code (see lines 3, 14, 18, 21). Each bit of code that you add should take $O(1)$ steps in the worst case. *Briefly* explain why your completion of the code is correct.

```
1:  function INITIALIZE                          ▷ Set up data structure to represent an empty set S
2:      Allocate arrays B[1..N] and Written[1..N]   ▷ These arrays may contain arbitrary garbage values
3:      Fill in the rest of this function
4:  end function

5:  function PREPARE(i)                          ▷ If B[i] has not been updated before, set it up
6:      if |B[i]| > NumWritten or Written[|B[i]|] ≠ i then
7:          NumWritten ← NumWritten + 1
8:          Written[NumWritten] ← i              ▷ Add i to NumWritten array
9:          B[i] ← NumWritten                    ▷ |B[i]| should have i's location in Written
10:     end if
11: end function
```

```
12: function INSERT(i)                    ▷ Add i to set S (if it is not already in S)
13:     PREPARE(i)
14:     B[i] ← _____
15: end function
16: function DELETE(i)                    ▷ Remove i from set S (if it is in S)
17:     PREPARE(i)
18:     B[i] ← _____
19: end function
20: function SEARCH(i)                    ▷ Return true if i ∈ S or false if i ∉ S
21:     Fill in this function
22: end function
```