BIHEAPS AND PIVOT SELECTION

MATTHEW GREGORY KRUPA

ABSTRACT. We define and classify a new category of graphs, one for each positive integer, called BiHeap graphs, which we use to define a new data structure called a BiHeap that is then applied to pivot selection for the QuickSelect and QuickSort algorithms. A part of a BiHeap's data lies in a min heap, a part lies is a max heap, and a part lies in both simultaneously. We give a simple to implement $O(\mathrm{N})$ algorithm, BiHeapify (), that forms a BiHeap out of any array of N values. We define an $O(\mathrm{N})$ function that produces a pivot value, pivot_value, which is the middle element of the array of N values to which it was applied, such that at least N^r elements have values ≤ pivot_value and at least N^r elements have values ≥ pivot_value, where $r := 1/\log_2 3 \approx 0.6309$. Testing on randomly generated arrays shows that on average, approximately $0.48N$ elements have values ≤ pivot_value and approximately $0.48N$ elements have values ≥ pivot_value. Thus, this pivot represents a substantial improvement above random selection of a pivot for use in the QuickSort and QuickSelect algorithms.

Contents

Part 1. Introduction 1
1 Notation and Terminology 1

Part 2. BiHeaps Graphs 2
2 Definitions of the BiHeap Graph on N Nodes and a BiHeap on N Nodes 2
2.1 Example Construction of a Directed BiHeap Graph 4
2.2 Illustrations 4
2.3 Checking the BiHeap Condition 8
2.4 BiHeap Terminology 9
3 Properties of BiHeaps 10
3.1 Basic Properties 10
4 Classification of BiHeaps 12
4.1 BiHeaps with Interior Extended Edges 12
4.2 BiHeaps whose Heaps Share an Edge 13
4.3 BiHeaps with Two Distinct Leaf Extended Edges Having Out-Degree 1 14
4.4 BiHeaps Whose Heaps Lack Parents With a Single Child 15
4.5 Computing HeapSize (N) 17

Part 3. BiHeapification 17
5 Overview of BiHeapification 17
6 Bounded Sifting Up Heaps 18
7 The BiHeapify () Algorithm 19

Part 4. Sifting an Element 20

Part 5. Recursively BiHeapifying Extended In Nodes with BiHeapifyInwards () 21

Keywords and phrases. Heaps, BiHeaps, Median, Selection algorithm, QuickSelect, Median, linear complexity .

8. Ins and Outs 21
9. Minimum Number of Elements Below and Above Pivots 21
10. The BiHeapifyInwards() Algorithm 22

Part 1. Introduction

It is assumed that the reader is familiar with graphs, trees, heaps, the standard $O(\mathrm{N})$ heapify algorithm and the proof that it is $O(\mathrm{N})$, the usual algorithms for sifting elements up and down heaps, and the QuickSort and QuickSelect algorithms.[1] Working implementations in Perl and C++ of the algorithms presented in this paper may be found at the author's GitHub account `github.com/mgkrupa`.

1. Notation and Terminology We will henceforth assume that N is a positive integer and that we are given an ordered list of N nodes denoted by $\mathrm{first} = \mathrm{first}+0, \ldots, \mathrm{first}+(N-1)$. If these nodes are assigned values then we will denote the value of node w by $^*w$. Throughout this paper, we will use C++ like pseudocode so that in particular, the result of performing non-negative integer division $a = b/c$; assigns the value $\left\lfloor \frac{b}{c} \right\rfloor$ to a.

Define an involutory bijection $\mathrm{Flip} : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ by

$$\mathrm{Flip}(i) = N - 1 - i.$$

By *the* i^{th} *node* we mean the node $\mathrm{first}+i$ and if w is a node then by *the min heap coordinate of* w, *the heap coordinate of* w, or simply *the coordinate of* w we mean the unique integer i such that $w = \mathrm{first}+i$ while by its *mirror coordinate* or its *max heap coordinate* we mean the integer $\mathrm{Flip}(i)$. We will say that a node *has coordinates* $i\{m\}$ or $i(m)$, if i and m are integers in $\{0, \ldots, N-1\}$, i is the coordinate of the node, and m is the node's mirror coordinate, in which case we'll necessarily have that $i = \mathrm{Flip}(m)$ and $m = \mathrm{Flip}(i)$. If w is a node with coordinate i then, assuming that the node is defined, for any integer k, $w + k$ will represent the node $\mathrm{first} + (i+k)$. In the pseudocode, any variable that stores the heap coordinate (resp. mirror coordinate) of a node will will have _hc (resp. _mc) as a postfix to its name.

If the nodes $\mathrm{first}, \ldots, \mathrm{first}+(N-1)$ are assigned values and if $<$ is a total order on these values then there exists a permutation $\iota : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ such that $^*(\mathrm{first}+\iota(0)) \leq \cdots \leq {}^*(\mathrm{first}+\iota(N-1))$. If N is even then we will call $^*(\mathrm{first}+\iota(N/2-1))$ *the left median* and call $^*(\mathrm{first}+\iota(N/2))$ *the right median* while if N is odd then we will call the value $^*(\mathrm{first}+\iota((N-1)/2))$ *the median.* If N is odd then we may also refer to the median as *the left median* or as *the right median.*

Part 2. BiHeaps Graphs

2. Definitions of the BiHeap Graph on N Nodes and a BiHeap on N Nodes The concepts in the first two definitions are well known.

Definition and Convention 2.1. Define three endomorphisms on the integers by

$$\mathrm{Parent}(0) = 0, \quad \mathrm{Parent}(i) = \lfloor (i-1)/2 \rfloor, \quad \mathrm{LeftChild}(i) = 2i+1, \quad \text{and } \mathrm{RightChild}(i) = 2(i+1)$$

We will identify nodes with their coordinates as well as with their min heap coordinates (but never their max heap coordinates) so that for all $0 < n \leq N$, the sets $\mathbb{N}_{<n} := \{0, \ldots, n-1\}$ and $\mathbb{N}^{<n} := \mathrm{Flip}(\mathbb{N}_{<n}) = \{(\mathrm{N}-1) - (n-1), \ldots, N-1\}$ denote the first (resp. last) n nodes of $\mathrm{first}, \ldots, \mathrm{first}+(N-1)$. ■

Definition 2.2. Let $h \in \{1, \ldots, \mathrm{N}\}$. By *the directed complete binary tree rooted at* 0 *on* h *nodes* we mean the graph $(\mathbb{N}_{<h}, E)$ where $(i, j) \in \mathbb{N}_{<h} \times \mathbb{N}_{<h}$ is an edge in E (going from i to j) if and only if either $j = \mathrm{LeftChild}(i)$ or $j = \mathrm{RightChild}(i)$, where in the former (resp. latter) case we'll say that j is i's *(min heap) left* (resp. *right*) *child.* We may also refer to this graph as *the directed min heap graph of size* h. If these nodes are

[1] An introduction to these topics may be found in almost any introductory textbook on data structures and algorithms.

assigned partially ordered values then we'll say that this graph is a *min heap (rooted at* 0 *on* h *nodes)* if the value of a child is always greater than or equal to the value of its parent.

By *the directed complete binary tree rooted at* $N-1$ *on* h *nodes* we mean the graph $\left(\mathbb{N}^{<h}, E\right)$ where $(i,j) \in \mathbb{N}^{<h} \times \mathbb{N}^{<h}$ is an edge in E (going from i to j) if and only if either $\operatorname{Flip}(j) = \operatorname{LeftChild}(\operatorname{Flip}(i))$ or $\operatorname{Flip}(j) = \operatorname{RightChild}(\operatorname{Flip}(i))$, where in the former (resp. latter) case we'll say that j is i's *(max heap) left* (resp. *right*) *child*. Note that if $m = \operatorname{Flip}(i)$ and $n = \operatorname{Flip}(j)$ are the max heap coordinates of these nodes then (i,j) is in E if and only if either $n = \operatorname{LeftChild}(m)$ or $n = \operatorname{RightChild}(m)$; this characterization is the reason for defining max heap coordinates. We may also refer to this graph as *the directed max heap graph of size* h. If these nodes are assigned partially ordered values then we'll say that this graph is a *max heap (rooted at* $\mathrm{N}-1$ *on* h *nodes)* if the value of a child is always less than or equal to the value of its parent.

By *the (undirected) complete binary tree rooted at* 0 *(resp.* $N-1$*) on* h *nodes* we mean the undirected graph induced by the directed complete binary tree rooted at 0 (resp. $\mathrm{N}-1$) on h nodes. ∎

Definition 2.3. For any positive integer $n \leq \mathrm{N}$, let C_n (resp. C^n) denote the complete binary tree on nodes $\mathbb{N}_{<n}$ (resp. $\mathbb{N}^{<n}$) rooted at node 0 (resp. $N-1$). Let U_n denote the graph on $\{0,\ldots,N-1\}$ formed by the union of the graphs C_n and C^n (where we only allow at most one edge between any two given nodes). Let R_n (resp. R^n) denote the restriction of U_n to the nodes $\mathbb{N}_{<n}$ (resp. $\mathbb{N}^{<n}$). Due to the symmetry of this construction, note that R_n and R^n are necessarily isomorphic graphs (via $\operatorname{Flip}()$) and that R_n is a binary tree rooted at 0 if and only if R^n is a binary tree rooted at $\mathrm{N}-1$, in which case $R_n = C_n$, $R^n = C^n$, and they are isomorphic as rooted binary trees.

Let $\operatorname{HeapSize}(\mathrm{N})$ denote the unique largest integer $h \leq \mathrm{N}$ such that R_h is a tree rooted at 0. Note that by symmetry, $\operatorname{HeapSize}(\mathrm{N})$ is also the unique largest integer $h \leq N$ such that R^h is a tree rooted at $\mathrm{N}-1$. We will call $U_{\operatorname{HeapSize}(\mathrm{N})}$ *the BiHeap graph on* N *nodes* and denote it by B_{N}. By *the heap size of* B_{N} we mean the integer $\operatorname{HeapSize}(\mathrm{N})$.

Letting $h = \operatorname{HeapSize}(\mathrm{N})$, by *the heap* or *the min heap* of B_{N}, denoted by MinH, we mean the graph $R_h = C_h$ and by *the mirror heap* or *the max heap* of B_{N}, denoted by MaxH, we mean the graph $R^h = C^h$. Letting $\rho = \lceil \mathrm{N}/2 \rceil$, we will call $R_\rho = C_\rho$ (resp. $R^\rho = C^\rho$) *the pure heap*, *the pure min heap*, or the PMinH (resp. *the pure mirror heap*, *the pure max heap*, or the PMaxH) of B_{N}.

If the nodes $0,\ldots,\mathrm{N}-1$ are assigned values and if $\prec$ is a partial order on these values then we will say that B_{N} *is a BiHeap with respect to* $\prec$ or a *directed BiHeap* if both of the following conditions hold:

(1) $R_h = C_h$ is a min heap with respect to $\prec$ rooted at node 0 (def. 2.2), and
(2) $R^h = C^h$ is a max heap with respect to $\prec$ rooted at node $\mathrm{N}-1$ (def. 2.2)

where $h = \operatorname{HeapSize}(\mathrm{N})$. If the partial order $\prec$ is clear from context then we will simply say that B_{N} *is a BiHeap*. We will henceforth assume without mention that, whenever it is needed, we are given some partial order on the nodes of B_{N}. Since all of the algorithms that we present require that $\prec$ be a total order, we will henceforth assume this as well. ∎

Since the definition of a BiHeap is merely that heap conditions hold on two subgraphs of the BiHeap graph, we adopt to BiHeaps all of the usual terminology used with Heaps whenever its meaning is either obvious or else clearly refers to one or both of these two heaps.

Definition 2.4. Let c and p be two nodes with coordinates $i(m)$ and $j(n)$. We will say that c *is* p*'s min (heap) left child* (resp. *right child*) or its *heap left* (resp. *right*) *child*, written $c = \operatorname{LeftChild}_{\mathrm{Min}}(p)$ (resp. $c = \operatorname{RightChild}_{\mathrm{Min}}(p)$), if both nodes belong to the min heap and when considered as nodes in the min heap, c is p's left (resp. right) child. In either case we will call p the *min (heap) parent of* c, written $p = \operatorname{Parent}_{\mathrm{Min}}(c)$, and say that c is a *min (heap) child of* p, which we'll denote by $c = \operatorname{Child}_{\mathrm{Min}}(p)$. In any of these definitions we may replace c and p with i and j, respectively.

We will also say that c *is* p*'s max (heap) left child* (resp. *right child*) or its *mirror (heap) left child* (resp. *right child*), written $c = \operatorname{LeftChild}_{\mathrm{Max}}(p)$ (resp. $c = \operatorname{RightChild}_{\mathrm{Max}}(p)$), if both nodes belong to the max heap and when considered as nodes in the max heap, c is p's left (resp. right) child. In either case we will call p the *max (heap) parent of* c, written $p = \operatorname{Parent}_{\mathrm{Max}}(c)$, and say that c is a *max (heap) child of* p, which we'll denote by $c = \mathrm{Child}_{\mathrm{Max}}(p)$. In any of these definitions, if $m = \mathrm{Flip}(i)$ and $n = \mathrm{Flip}(j)$ are understood to be max heap coordinates then we may replace c and p with m and n, respectively.

The analogous definitions for the pure min heap and the pure max heap should be clear. ∎

Definition 2.5. An edge in a BiHeap graph is said to be *min* (resp. *max*) if it belongs to the min (resp. max) heap while it is *pure min* (resp. *pure max*) if it belongs to the pure min (resp. pure max) heap. An edge is *extended min* (resp. *extended max*) if it is an edge that belongs to the min (resp. max) heap but not to the pure min (resp. max) heap. An edge is *extended* (resp. *pure*) if it is an extended (resp. pure) min edge or an extended (resp. pure) max edge.

- Note that in the case where the BiHeap has an odd number, N, of nodes the last edge of the pure min (resp. pure max) heap, which ends at "the middle node" (i.e. node $\frac{\mathrm{N}-1}{2}$) is not an extended edge.

∎

As we shall see, there is at most one edge that belongs simultaneously to the min heap and to the max heap. Consequently, we can unambiguously define a direction for but possibly one edge in B_{N}.

Definition 2.6. Let $\vec{\mathrm{B}}_{\mathrm{N}}$ denote B_{N} Define *the directed BiHeap graph on* N *nodes* to be graph B_{N} with each edge directed so as to go from its parent to its child, where if an edge (v_1, v_2) simultaneously belongs to both the min heap and the max then this edge is to be interpreted as a bi-directional edge (or if more appropriate for the situation, it may instead be replaced by two directed edges $v_1 \to v_2$ and $v_2 \to v_1$). Explicitly, an edge in the min heap (resp. max heap) points from its parent to its child. ∎

There is thus a canonical one-to-one correspondence between BiHeap graphs and directed BiHeap graphs, which we will henceforth use to identify the two without mention.

2.1. Example Construction of a Directed BiHeap Graph We now explain how one may construct the directed BiHeap graph on N = 14 nodes.

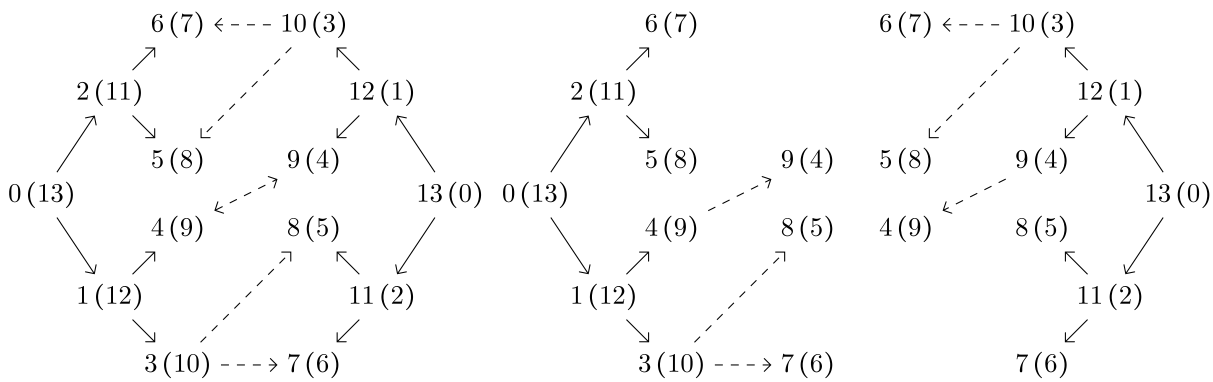

FIGURE 1. On the left is the directed (def. 2.6) BiHeap graph on 14 nodes, which has HeapSize (14) = 10. For the BiHeap graph, the double arrow should be interpreted as being a single edge in the graph. In the center is this BiHeap's directed (def. 2.2) min heap graph, whose restriction to the solid arrows forms the (directed) pure min heap graph. At the right is this BiHeap's directed max heap graph, whose restriction to the solid arrows forms the (directed) pure max heap graph.

Note that this BiHeap graph is the union of the min heap graph with the max heap graph, where of course when a min heap node and a max heap node have the same coordinates then they are considered to be the same node. Visually, one obtains the BiHeap graph by "rigidly sliding without rotation" the min heap graph (pictured in the center) to the right until all nodes with equal coordinates overlap one another.

To become comfortable with BiHeaps, the author recommends that the reader construct a few BiHeap graphs (for any 6 consecutive values of N) by hand by going through the construction in definition 2.3 and (symmetrically) adding two edges at a time. For instance first add edges $0\,(13) \to 1\,(12)$ and $13\,(0) \to 12\,(1)$ (i.e. from a min/max root to its left child) and then check the defining condition (which holds). Follow this by adding edges $0\,(13) \to 2\,(11)$ and $13\,(0) \to 11\,(2)$ (i.e. from a min/max root to its right child) and then checking the defining condition (which holds). Now add edges $1\,(12) \to 3\,(10)$ and $12\,(1) \to 10\,(3)$ and check the defining condition (which again holds). This process continues until eventually (after 7 more iterations in this case) the defining condition fails to hold, which tells you that you should not have added the last two edges (i.e. you should only have done 6 more iterations instead of 7).

2.2. Illustrations We now illustrate the directed BiHeap graphs on N nodes for $1 < N < 20$ where $N = 1$ is omitted since it is just a point. The binary tree formed by restricting the BiHeap graph to all arrows (together with their incident nodes) going from left to right (resp. right to left) form the min (resp. max) heap of the BiHeap graph. Note that some of the BiHeap graphs in the following illustrations have an arrow that is double headed, which should be interpreted in the obvious way as a bi-directional arrow. That is, an arrow is draw as double headed if and only if there is an edge between those two nodes that belongs to both the min heap and the max heap.

If one where to draw a line straight down the middle of one of these illustrated BiHeap graphs then the subgraph on the left (resp. right), which includes the middle node if N is odd, would be the pure min heap (resp. pure max heap. The pure min heap and the pure max heap are always isomorphic to each other via the isomorphism $i\,\{m\} \mapsto m\,\{i\}$. Note that if one were to rotate the min heap (resp. pure min heap) by 180 degrees around the center of the BiHeap graph then, ignoring all nodes' labels, one obtains the max heap (resp. pure max heap).

Henceforth, we will identify the node first $+c$ with the node in the BiHeap graph on N nodes labeled $c\,\{m\}$ or $c(m)$ where $m = \mathrm{Flip}(c)$. In this way, the node labeled $c\,\{m\}$ or $c(m)$ will be said to have coordinate c and mirror coordinate $m = \mathrm{Flip}(c)$.

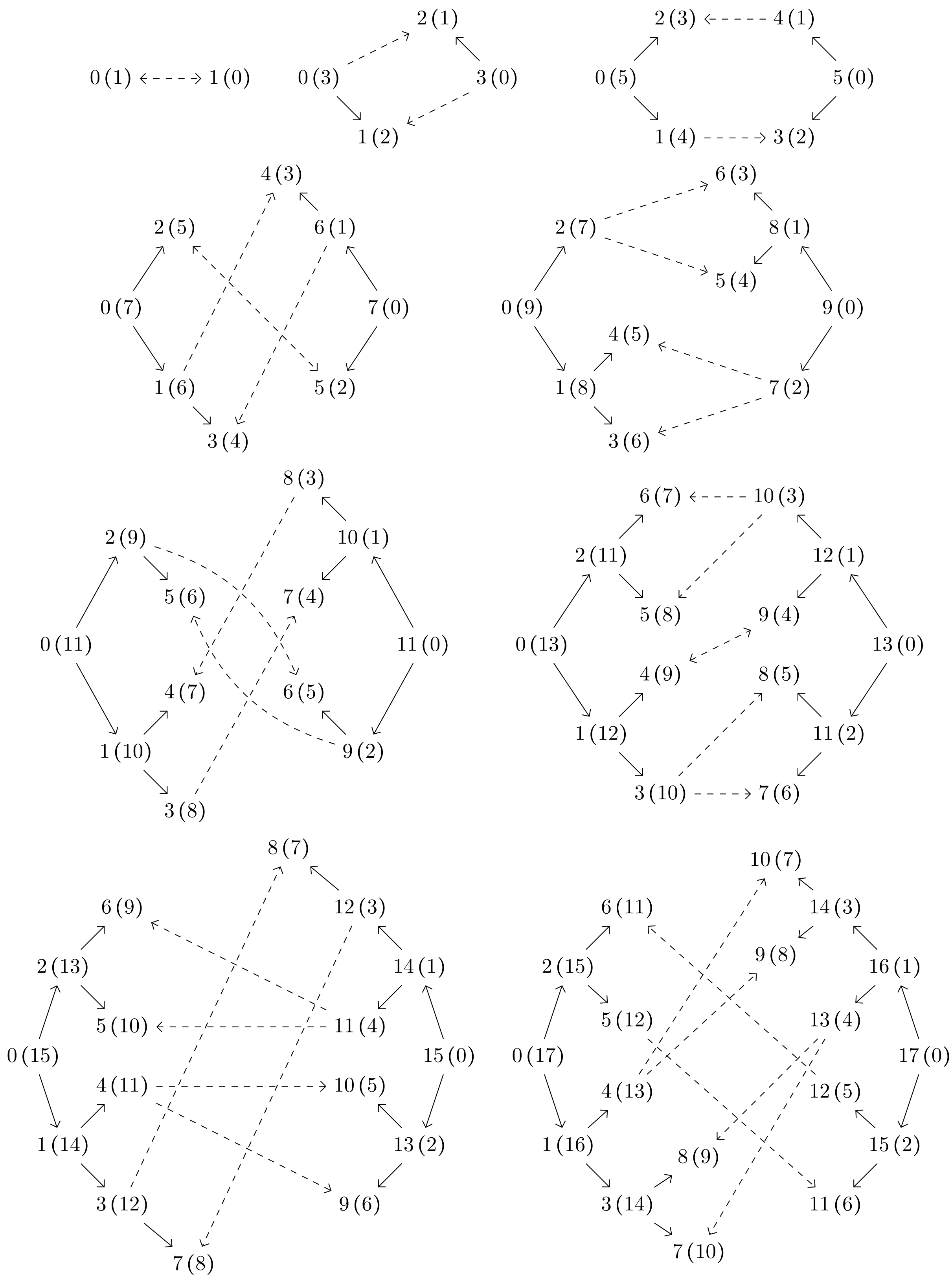

FIGURE 2. Going from left to right and from the top down, these are the BiHeap graphs on $2, 4, 6, 8, 10, 12, 14, 16,$ and 18 nodes, respectively. The two connected components of the subgraph consisting of only the solid arrows and their nodes form the two "pure" subheaps of these BiHeap graphs.

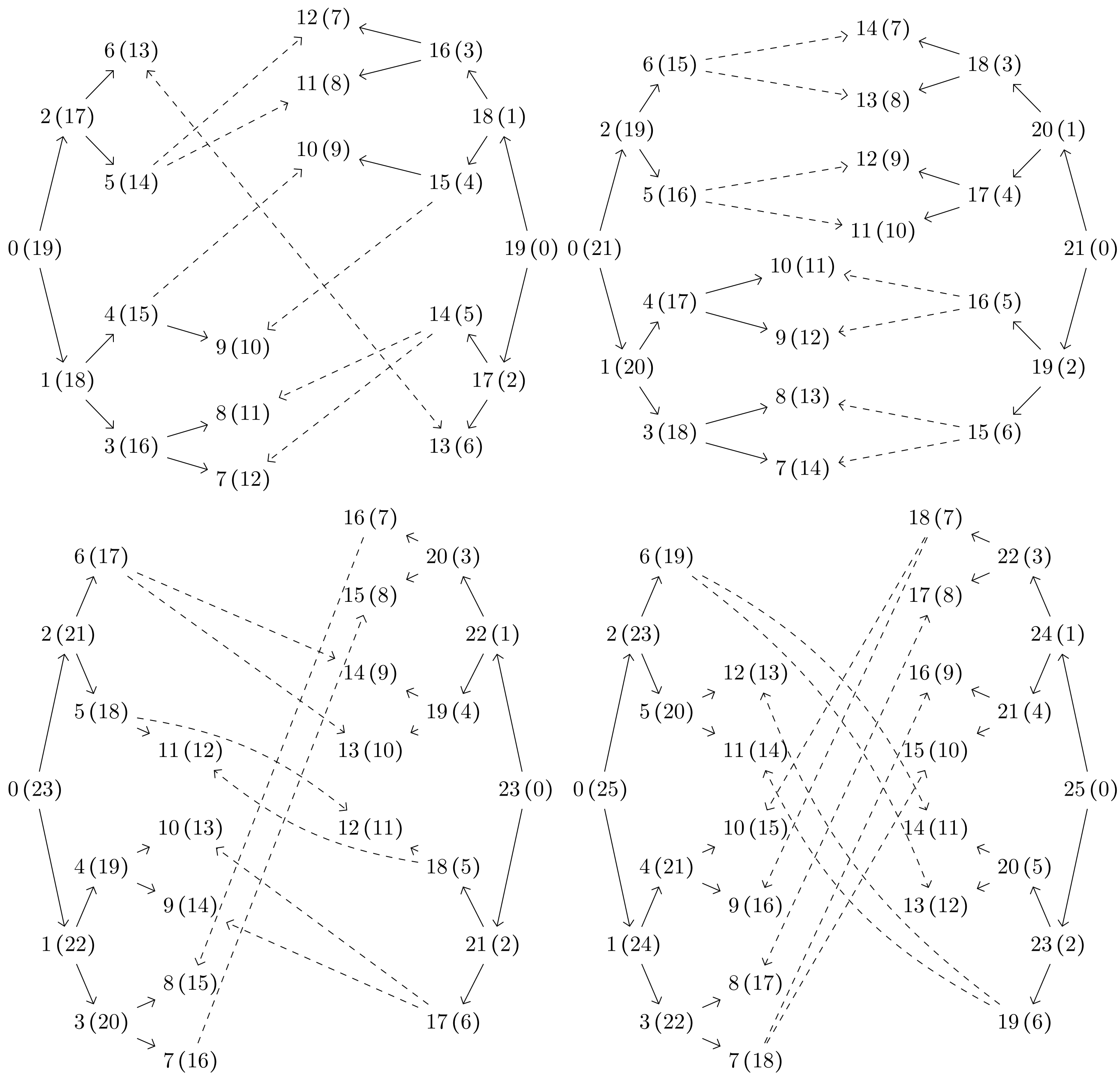

FIGURE 3. Going from left to right and from the top down, these are the BiHeap graphs on $20, 22, 24$, and 26 nodes, respectively.

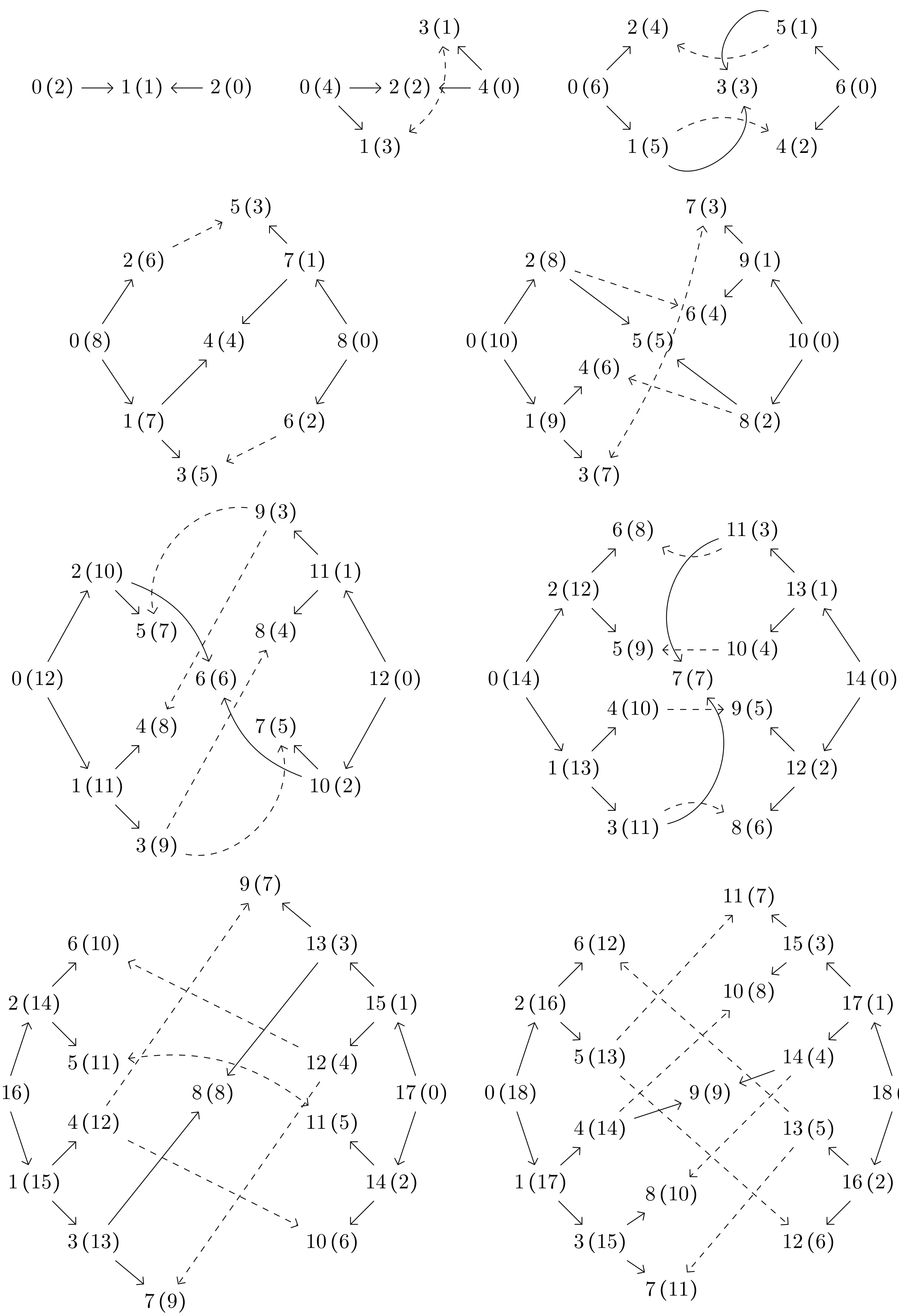

FIGURE 4. Going from left to right and from the top down, these are the BiHeap graphs on $3, 5, 7, 9, 11, 13, 15, 17$, and 19 nodes, respectively. The subgraph consisting of the middle node and all nodes on the left (resp. right) half of the graph together with the solid arrows between these nodes form the “pure” min (resp. “pure” max) heaps of these BiHeap graphs.

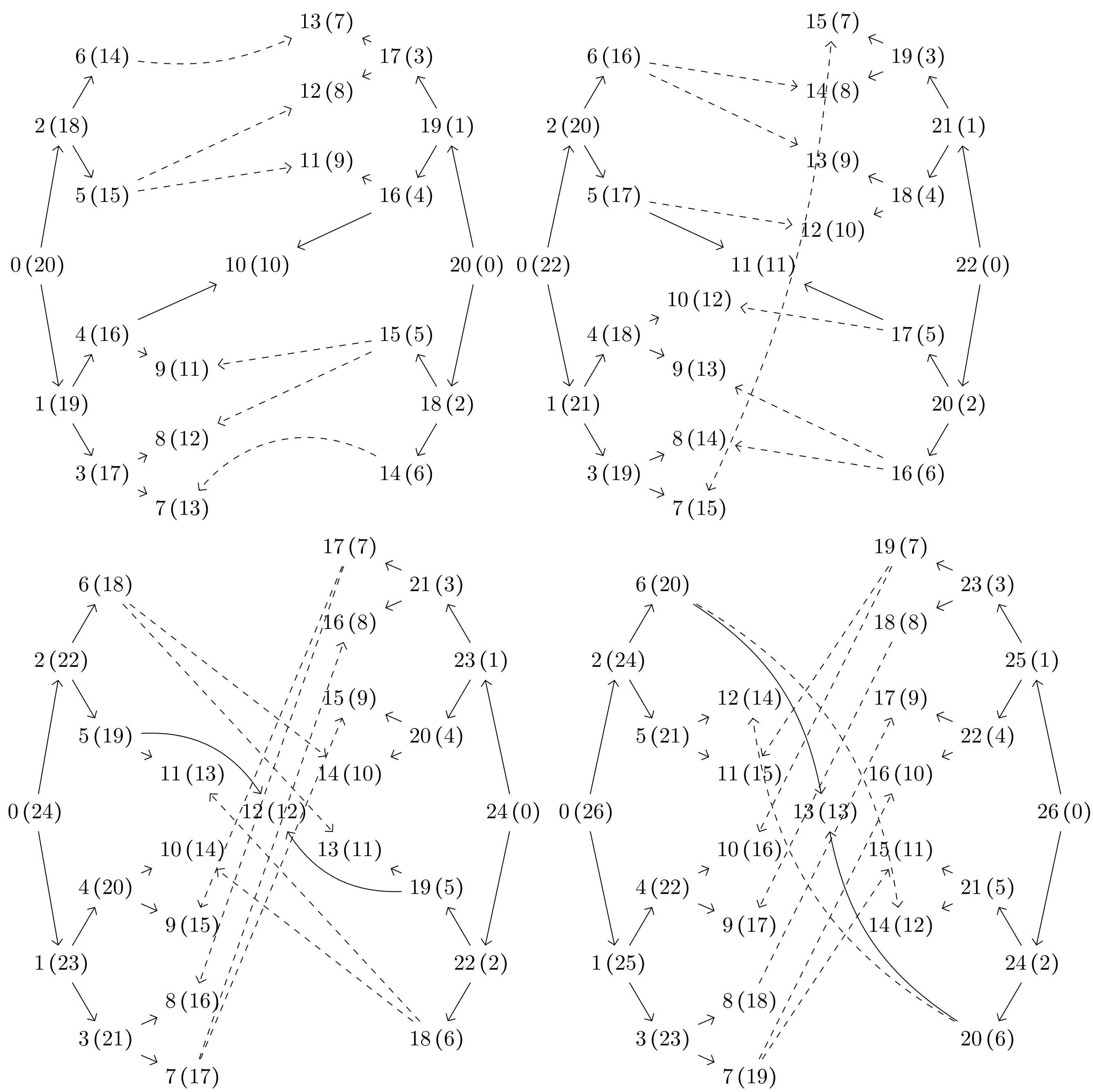

FIGURE 5. Going from left to right and from the top down, these are the BiHeap graphs on $21, 23, 25$, and 27 nodes, respectively.

2.3. Checking the BiHeap Condition The following C++ like pseudocode checks if the nodes, the first of which is labeled first, form a BiHeap. This consists of nothing more than checking the min heap and the max heap conditions. Recall that the i^{th} node in the list of nodes is denoted by $\mathrm{first} + \mathrm{i}$ and that this node's value is denoted by $^*\,(\mathrm{first} + \mathrm{i})$. We have already defined the function $\mathrm{HeapSize}\,()$ but we will give the formula for computing it after we've classified all BiHeap graphs.

```cpp
#define Flip(a) (N - 1 - (a))

//Returns true if and only if the nodes first, ..., first + (N - 1) form a BiHeap.
bool IsBiHeap(Node first, int N) {
  if (N <= 2) {
    if (N == 2)
      return *first <= *(first + 1);
    return true;
  }
  const int heap_size = HeapSize(N);
  //Check that the nodes first, ..., first + (heap_size - 1) form
  // a min heap with the minimum at first. This is half of the biheap condition.
  int parent_hc = 0; //This variable stores the parent's min Heap Coordinate.
  while(RightChild(parent_hc) < heap_size) { //While the right child is in the min heap.
    auto parent_value     = *(first + parent_hc);
    auto left_child_value = *(first + LeftChild(parent_hc));
    if (parent_value > left_child_value)
      return false;
    auto right_child_value = *(first + RightChild(parent_hc));
    if (parent_value > right_child_value)
      return false;
    parent_hc++;
  }
  if (LeftChild(parent_hc) < heap_size) { //If there is a parent with a single child.
    auto parent_value     = *(first + parent_hc);
    auto left_child_value = *(first + LeftChild(parent_hc));
    if (parent_value > left_child_value)
      return false;
  }
  //Check that the nodes first + Flip(0), ..., first + Flip(heap_size - 1)
  // form a max heap with the maximum at first + (N - 1).
  int parent_mc = 0; //This variable stores the parent's Max heap Coordinate.
  while(RightChild(parent_mc) < heap_size) { //While the right child is in the max heap.
    auto parent_value     = *(first + Flip(parent_hc));
    auto left_child_value = *(first + Flip(LeftChild(parent_mc)));
    if (parent_value < left_child_value)
      return false;
    auto right_child_value = *(first + Flip(RightChild(parent_mc)));
    if (parent_value < right_child_value)
      return false;
    parent_mc++;
  }
  if (LeftChild(parent_mc) < heap_size) {
    auto parent_value     = *(first + Flip(parent_mc));
    auto left_child_value = *(first + Flip(LeftChild(parent_mc)));
    if (parent_value < left_child_value)
      return false;
  }
  return true;
}
```

.

2.4. BiHeap Terminology Note that nodes and edges either belong only to the min heap, only to the max heap, or belong to both simultaneously. In particular, binary tree definitions such as interior node and leaf are now ambiguous. Recall that with definition 2.4, we have disambiguated the notions of child and parent.

Definition 2.7.

(1) Say that a node is a *min* (resp. *max*) *leaf* if it is a leaf in the min (resp. max) heap.
(2) Say that a node is a *pure min* (resp. *pure max*) *leaf* if it is a leaf in the pure min (resp. pure max) heap.
(3) Say that a node is *extended min* (resp. *extended max*) or a *non-pure min* (resp. *non-pure max*) if it belongs to the min (resp. max) heap but not to the pure min (resp. pure max) heap.
(4) Say that a node v is *leaf extended min* (resp. *leaf extended max*) if (a) it is an extended min (resp. extended max) node and (b) in the min (resp. max) heap, there exists an edge to v from a leaf of the pure min (resp. pure max) heap.
(5) Say that a node v is *interior extended min* (resp. *interior extended max*) if (a) it is an extended min (resp. extended max) node and (b) in the min (resp. max) heap, there exists an edge to v from an node that is in the interior of the pure min (resp. pure max) heap.

■

For edges, in addition to definition 2.5, we introduce the following terminology, where in reference to an edge, "extended" (resp. "pure") refers to the dashed (resp. solid) arrows in figures 2, 3, 4, and 5.

Definition 2.8.

(1) Say that an edge is *leaf extended min* (resp. *leaf extended max*) if it is an extended min (resp. extended) edge and it is incident to a leaf extended min (resp. leaf extended max) node. That is, an edge is leaf extended min (resp. leaf extended max) if one of its incident nodes is a leaf in the pure min (resp. pure max) heap while the other incident node does not belong to the pure min (resp. pure max) heap. It is *leaf extended* if it is a leaf extended min edge or a leaf extended max edge.
(2) Say that an edge is *interior extended min* (resp. *interior extended max*) if it is an extended min (resp. extended max) edge and it is incident to an interior extended min (resp. interior extended max) node. It is *interior extended* if it is an interior extended min edge or an interior extended max edge.

■

3. Properties of BiHeaps

3.1. Basic Properties We now make some observations that are clear from the definition of a BiHeap. For convienence, in (3), we give a list of algebraic equations involving various subsets of the variables i, j, m, and n that are equivalent to $j\{n\}$ being a child of $i\{m\}$. These equations will henceforth be used without comment.

Lemma 3.1. Let $N > 0$ and let v and w be two nodes in B_B with coordinates $i\{m\}$ and $j\{n\}$, respectively (so that by definition, $m = \mathrm{Flip}(i)$ and $n = \mathrm{Flip}(j)$) and recall that we identify a node with its coordinates. The following statements hold.:

(1) v belongs to the min heap (resp. pure min heap) if and only if the node with coordinate $m\{i\}$ belongs to the max heap (resp. pure max heap).
(2) $j - i = m - n$.
(3) The following are equivalent:
   (a) $j\{n\} = \mathrm{LeftChild}_{\mathrm{Min}}(i\{m\})$ (resp. $j\{n\} = \mathrm{RightChild}_{\mathrm{Min}}(i\{m\})$) in the min heap (def. 2.4).
   (b) $n\{j\} = \mathrm{LeftChild}_{\mathrm{Max}}(m\{i\})$ (resp. $n\{j\} = \mathrm{RightChild}_{\mathrm{Max}}(m\{i\})$) in the max heap.
   (c) (i, j relationship) $j = \mathrm{LeftChild}(i)$ (def. 2.1).
   (d) (i, n relationship) $N = n + 2(i+1)$ (resp. $N = n + 2(i+1) + 1$), which can be rewritten as $N = n + \mathrm{RightChild}(m)$ (resp. $N = n + \mathrm{RightChild}(m) + 1$).
   (e) (j, m relationship) $j + 2m + 1 = 2N$ (resp. $j + 2m = 2N$), which can be rewritten as $j + \mathrm{LeftChild}(m) = 2N$ (resp. $j + \mathrm{LeftChild}(m) = \mathrm{LeftChild}(N)$).
   (f) (m, n relationship) $2m = n + N$ (resp. $2m = n + N + 1$).
   (g) (i, m, n relationship) $m - n = i + 1$ (resp. $m - n = i + 2$).
   (h) (i, j, m relationship) $m + j = N + i$ (resp. $m + j = N + i + 1$).

(4) There is at most one pure min heap interior node that has an extended min heap edge leaving it along with an extended max heap edge entering it (since otherwise the max heap would have a cycle). Hence, if the edge from $i\{m\}$ to $j\{n\}$ is such an edge then $i >= \lfloor (P-1)/2 \rfloor$, which is the coordinate of the last min heap interior node, where $P = \lceil N/2 \rceil$.

(5) If v belongs to the min heap and there exists an extended edge in the BiHeap that is incident to both v and w then, so as not to violate the min heap condition, all of the following conditions must hold:
   (a) $i\{m\}$ must belong to the pure min heap and $j\{n\}$ to the pure max heap.
   (b) $i\{m\}$ is either a leaf of the pure min heap or else it is the unique pure min heap interior node with exactly 1 child (if such a node exists). The analogous statement holds for $j\{n\}$ with respect to the pure max heap. Consequently, letting $P = \lceil N/2 \rceil$ and $f = \lfloor (P-1)/2 \rfloor$, we have
      (i) The first pure min heap node to be adjacent to a pure max heap node is node $f\{\mathrm{Flip}(f)\}$.
      (ii) If $k > \mathrm{Flip}(f)$ then node $k\{\mathrm{Flip}(k)\}$ is not adjacent to any pure min heap node.
      (iii) $i \geq f$, $j > P$, and $j \leq \mathrm{Flip}(f)$.
      (iv) If $i\{m\}$ has exactly one child in the pure min heap then $i = \lfloor (P-1)/2 \rfloor$, and $j = P$.
      (v) If $i\{m\}$ is a pure min heap leaf node then $i \geq f$, where this inequality is strict if P is even.
   (c) If $k\{p\}$ is any min heap node such that it has an edge incident to $j\{n\}$ then $i = k$ and $m = p$. That is, there is at most one extended min heap edge incident to any give pure max heap node.

(6) It follows from the definition of the BiHeap B_N that all nodes in the directed BiHeap graph have an out-degree of at most 2 and an in-degree between 1 and 2. Furthermore, when B_N is viewed as an undirected graph then every node has degree no greater than 3 and no less than 2.

Proposition 3.2. Suppose that v is a pure min heap leaf such that v has two min heap children (which are then necessarily in the pure max heap). Then N is even if and only if $\mathrm{Parent}_{\mathrm{Max}}(\mathrm{LeftChild}_{\mathrm{Min}}(v)) = \mathrm{Parent}_{\mathrm{Max}}(\mathrm{RightChild}_{\mathrm{Min}}(v))$.

Proof. Let $i\{m\}$ be the coordinates of v. Note that $\mathrm{Parent}(\mathrm{Flip}(\mathrm{LeftChild}(i))) = \left\lfloor \frac{N-2i-3}{2} \right\rfloor$ while $\mathrm{Parent}(\mathrm{Flip}(\mathrm{RightChild}(i))) = \left\lfloor \frac{N-2i-4}{2} \right\rfloor$ so that if N is even then these are both equal to $\frac{N}{2} - i - 2$ while if N is even then they are not equal. ∎

Lemma 3.3. Let $i\{m\}$ be the first leaf in the pure min heap and let $j\{n\}$ be the first leaf in the pure max heap (i.e. the pure max heap leaf that is next to the pure max heap's last interior node). Let $\rho = \lceil N/2 \rceil$ be the size of the pure heap. Then

$$i = n = \lfloor \rho/2 \rfloor = \frac{1}{4}\left(N - 2 + ((3N+2) \mod 4)\right)$$

$$j = m = \mathrm{Flip}(\lfloor \rho/2 \rfloor) = \frac{1}{4}\left(3N - 2 - ((3N+2) \mod 4)\right)$$

and

$$\rho = \lceil N/2 \rceil = \frac{N + (N \mod 2)}{2}.$$

Proof. Recall that a complete binary tree with ρ nodes has exactly $\lfloor \rho/2 \rfloor$ interior nodes, from which it follows that the first leaf in the pure min heap has min heap coordinate $\lfloor \rho/2 \rfloor$. It then follows that the first leaf in the pure max heap has min heap coordinate $j = \mathrm{Flip}(\lfloor \rho/2 \rfloor)$. One can verify that the following equalities hold by going through each of the four cases: One can verify that the following equalities hold by going through each of the four cases:

$$\lfloor \lceil N/2 \rceil /2 \rfloor = \begin{cases} \frac{1}{4}(N-2+3) & \text{if } N \text{ is odd and } 4 \nmid N-1 \iff (3N+2) \mod 4 = 3 \iff 2 \nmid N \text{ and } 2 \mid \rho \\ \frac{1}{4}(N-2+2) & \text{if } 4 \mid N \iff (3N+2) \mod 4 = 2 \iff 2 \mid N \text{ and } 2 \mid \rho \\ \frac{1}{4}(N-2+1) & \text{if } N \text{ is odd and } 4 \mid N-1 \iff (3N+2) \mod 4 = 1 \iff 2 \nmid N \text{ and } 2 \nmid \rho \\ \frac{1}{4}(N-2+0) & \text{if } N \text{ is even and } 4 \nmid N \iff (3N+2) \mod 4 = 0 \iff 2 \mid N \text{ and } 2 \nmid \rho \end{cases}$$

It then remains to go through each of those cases again and note that in each case, the value is necessarily equal to $\frac{1}{4}(N - 2 + ((3N + 2) \mod 4))$. Since $j = \text{Flip}(i)$, it follows that $j = \frac{1}{4}(3N - 2 - ((3N + 2) \mod 4))$. By the symmetry in the construction of the BiHeap graph, we have that $i = n$ and $j = m$. ∎

Corollary 3.4. Let $i\{m\}$ be the last interior node of the pure min heap, let $j\{n\}$ be the last interior node in the pure max heap, and let $\rho = \lceil N/2 \rceil$ be the size of the pure heap. Then

$$i = n = \lfloor \rho/2 \rfloor - 1 = \frac{1}{4}(N - 6 + ((3N + 2) \mod 4))$$

and

$$j = m = \text{Flip}(\lfloor \rho/2 \rfloor - 1) = \frac{1}{4}(3N + 2 - ((3N + 2) \mod 4)).$$

Proof. The first leaf in the pure min heap comes immediately after the pure min heap's last interior node. ∎

4. Classification of BiHeaps One way to classify BiHeaps is whether or not they have a "middle node" (i.e. based on the BiHeap's size is even or odd), which is the content of proposition 3.2. Another way to classify BiHeaps is to group them depending on whether or not they have interior extended edges (def. 2.7(5)). And yet a third more complicated way to classify them is by the value of their number of nodes moduloed by 3. These three classifications broadly describe the most important distinguishing features of BiHeap graphs and they are independent of each other so that each BiHeap falls into exactly one of twelve categories. In terms of applications, the most useful result of these characterizations is that they prove the formula $\text{HeapSize}(N) = \left\lceil \frac{2N}{3} \right\rceil$. With this formula at hand, the reader may now skip the proofs of the characterizations and go straight to the algorithms if they so desire.

4.1. BiHeaps with Interior Extended Edges

Proposition 4.1. Let $\rho = \lceil N/2 \rceil$ be the size of the pure heap. Then the following are equivalent:

(1) ρ is even.
(2) 4 divides N or 4 divides $N + 1$.
(3) There exists an interior extended edge (def. 2.8(2)).
(4) There exists exactly one interior extended min edge and/or there exists exactly one interior extended max edge.

If this is the case and if the unique interior extended min edge goes from the pure min heap interior node $i\{m\}$ to the pure max heap node $j\{n\}$ then these coordinates are determined as follows:

$$\begin{aligned}
i &= \frac{1}{4}(N - 6 + ((3N + 2) \mod 4)) \\
m &= \frac{1}{4}(3N + 2 - ((3N + 2) \mod 4)) \\
j &= \frac{1}{2}(N - 4 + ((3N + 2) \mod 4)) = \text{RightChild}(\rho/2 - 1) \\
n &= \frac{1}{2}(N + 2 - ((3N + 2) \mod 4))
\end{aligned}$$

Proof. The equivalence of (1), (2), and (3) is obvious from the definitions of a BiHeap graph, the heaps, and the pure heaps and from the fact that a complete binary tree with ρ nodes has exactly $\lfloor \rho/2 \rfloor$ interior nodes. Everything else follows from lemma 3.3. ∎

4.2. BiHeaps whose Heaps Share an Edge

Proposition 4.2. The following are equivalent:

(1) There exist nodes w and x such that $x = \text{Child}_{\text{Min}}(w)$ in the min heap and $w = \text{Child}_{\text{Max}}(x)$ in the max heap.
(2) 3 divides $N - 2$ (or equivalently, $N \mod 3 = 2$).

If this is the case and if w and x have coordinates $i\{m\}$ and $j\{n\}$, respectively, then

$$i = n = \frac{N-2}{3} \qquad \text{and} \qquad j = m = \frac{2N-1}{3} = 2i + 1$$

so that, in particular, i, n, j, m, w, and x are uniquely determined and furthermore,

(a) $x = \text{LeftChild}_{\text{Min}}(w)$ and $w = \text{LeftChild}_{\text{Max}}(x)$.
(b) The pure min heap node w is the last node in the *max* heap and the pure max heap node x is the last node in the *min* heap.
(c) A node $k\{p\}$ is in the max heap if and only if $k \geq \frac{N-2}{3}$.
(d) A node $k\{p\}$ is in the min heap if and only if $k \leq \frac{2N-1}{3}$.
(e) The number of pure min heap leaves with exactly two children in the min heap is

$$\frac{1}{3}\left(\left\lceil \frac{\rho}{2} \right\rceil - 2 + (\rho \mod 2) - (N \mod 2)\right)$$

where $\rho = \left\lceil \frac{N}{2} \right\rceil$ is the size of the pure min heap.

Proof. Suppose for the sake of contradiction that $j = \text{LeftChild}(i)$ and $m = \text{RightChild}(n)$. Then $m - n = i + 1$ and $n + 2 = j - i$ so that $j = m + 1 = \text{Flip}(i) + 1 = N - i$. Now, $2i + 1 = \text{LeftChild}(i) = j = N - i$ so that $i = \frac{N-1}{3}$ and $j = \text{LeftChild}(i) = \frac{2(N-1)}{3} + 1$, from which we conclude that $N = \frac{3(j-1)}{2} + 1$. By symmetry, there also exist node w' and x', with coordinates $i'\{m'\}$ and $j'\{n'\}$, respectively, such that $x' = \text{LeftChild}_{\text{Max}}(w')$ and $w' = \text{RightChild}_{\text{Min}}(x')$. This implies that $n' = \text{LeftChild}(m')$ and $i' = \text{RightChild}(j')$. Proceeding as before, we can obtain $n' + m' = N$, $m' = \frac{N-1}{3}$, and $n' = \frac{2(N-1)}{3} + 1$, thus showing that $m' = i$ and $n' = j$. Now $j = \text{Flip}(i) = \text{Flip}(m') = \text{RightChild}(j') = 2(j' + 1)$, which shows that j is even. But $j = \text{LeftChild}(i) = 2i + 1$ so that j is also odd. An analogous argument obtains a contradiction from the assumption that $j = \text{RightChild}(i)$ and $m = \text{LeftChild}(n)$.

Now suppose for the sake of contradiction that $j = \text{RightChild}(i)$ and $m = \text{RightChild}(n)$. The former happens if and only if $m - i = n + 2$ while the latter happens if and only if $n + 2 = j - i$, giving ups $m = j$ from which $i = n$ follows. Now $2i + 2 = \text{RightChild}(i) = m = \text{Flip}(i) = N - 1 - i$ so that $N = 3(i + 1)$. But we also have that $j = m = \text{RightChild}(n) = \text{RightChild}(\text{Flip}(j))$, which gives us $j = 2N - 2j$. It follows that $3j = 2N$, $j = 2(i + 1)$, and $j = \frac{2N}{3}$, which is possible only when j is even. Now $i = \text{Parent}(j) = j/2 = i + 1$, giving us a contradiction.

Now suppose that $j = \text{LeftChild}(i)$ and $m = \text{LeftChild}(n)$. As before we can obtain $m = j$ and $i = n$ from which we conclude that $N - 1 - i = \text{Flip}(i) = j = m = i = \text{LeftChild}(n) = \text{LeftChild}(i) = 2i + 1$. This shows that $N = 3i + 2$ and $i = \frac{N-2}{3}$. Using $m = \text{LeftChild}(n) = \text{LeftChild}\left(\frac{N-2}{3}\right)$ we obtain $m = \frac{2N-1}{3}$.

We've thus proven both (a) and $(1) \implies (2)$. To prove $(2) \implies (1)$, let $i = \frac{N-2}{2}$ and $j = \text{LeftChild}(i)$, where of course their mirror coordinates are $m = \text{Flip}(i)$ and $n = \text{Flip}(j)$. We must show that $m = \text{LeftChild}(n)$. Expanding $\text{LeftChild}(n) = \text{LeftChild}(\text{Flip}(\text{LeftChild}(i)))$ gives us $\text{LeftChild}(n) = \frac{2N-1}{3}$ while expanding $m = \text{Flip}(i) = \text{Flip}\left(\frac{N-2}{3}\right)$ also gives us $\frac{2N-1}{3}$, as desired.

Suppose that x had a max heap sibling s that was also the min heap. Since x is its max heap parent's left child, the min heap coordinate of s is necessarily $j + 1$. Let $k = j + 1$ and $p = \text{Flip}(p)$ so that s's coordinates are $k\{p\}$ and $p = N - 1 - (j + 1) = \text{Flip}(j) - 1 = n - 1$. Note that (b) is clearly true for $N = 2$ and $N = 5$ so assume that $N > 5$. Since $N > 5$, $p < n$, and since there exists a max heap edge leading $j\{n\}$, by the symmetry in the construction of the BiHeap graph, there must also be a max heap edge leaving node leaving node $k\{p\}$ and going into a min heap node. Thus the restriction of the BiHeap graph, $\text{Bi}(N)$, to node

$\{0, \ldots, j+1\}$ contains the cycle:

$$w \to k\{p\} \to \mathrm{RightChild}_{\mathrm{Max}}(k\{p\}) \to 0\{N-1\} \to w$$

contradicting the definition of the BiHeap graph. Since x has no max heap sibling that is also in the min heap, it is therefore the last node in the min heap, which proves (b). The remaining statements follow immediately from what has already been shown.

To prove (e), let χ equal 1 if $\rho := \lceil \frac{N}{2} \rceil$ is even and 0 otherwise, let x be the desired quantity, and note that the number of BiHeap leaves is equal to both $6x + 2 + 2\chi + (N \mod 2)$ and $2\lceil \frac{\rho}{2} \rceil - (N \mod 2)$. ■

4.3. BiHeaps with Two Distinct Leaf Extended Edges Having Out-Degree 1

Proposition 4.3. The following are equivalent:

(1) There exist a node v in the min heap with exactly one min heap child w and the edge formed by any such pair does not belong to the max heap.
(2) 3 divides N.

For $N \neq 3$, we can add the following to the above list:

(3) There exists a pure min heap leaf node v such that v has a unique min heap child w such that v is not a max heap child of w.

In this case, the v and w given in (3) are the same v and w given in (1). If (1) holds and if such v and w have coordinates $i\{m\}$ and $j\{n\}$, respectively, then

$$i = \frac{N}{3} - 1, \qquad j = \frac{2}{3}N - 1, \qquad m = \frac{2}{3}N, \qquad \text{and} \qquad n = \frac{N}{3}$$

so that, in particular, i, j, m, n, v, and w are uniquely determined and furthermore,

(a) $w = \mathrm{LeftChild}_{\mathrm{Min}}(v)$ and w has no sibling in the min heap.
(b) N is even if and only if $v = \mathrm{LeftChild}_{\mathrm{Min}}(\mathrm{Parent}_{\mathrm{Min}}(v))$.
(c) If $N > 3$ then N is odd if and only if $v = \mathrm{RightChild}_{\mathrm{Min}}(\mathrm{Parent}_{\mathrm{Min}}(v))$.
(d) The pure min heap node v is the last node in the min heap with any outgoing edges (i.e. for any node $k\{p\}$, if $k > i$ then $k\{p\}$ does not have any children in the min heap).
(e) A node $k\{p\}$ is in the max heap if and only if $k \geq \frac{N}{3}$.
(f) A node $k\{p\}$ is in the min heap if and only if $k < \frac{2N}{3}$.
(g) If $N \neq 3$, then the number of pure min heap leaves with exactly two children in the min heap is

$$\frac{1}{3}\left(\left\lceil \frac{\rho}{2} \right\rceil - 3 + (\rho \mod 2) - (N \mod 2)\right)$$

where $\rho = \lceil \frac{N}{2} \rceil$ is the size of the pure min heap.

Remark 4.4. Note that statement 3 does *not* say that v has only one min heap child.

Proof. Note that when $N = 3$ then (1), (2), and $(a) - (f)$ are all true and the formulas for i, j, m, and n hold while for $N = 1, 2, 4$, and 5, statements (1), (2), and (3) are all false. We may now henceforth assume that $N \geq 6$, so that the pure min heap has at least one interior node and the min heap has at least one extended node. Since $N \geq 6$, once we prove that (1) and (3) are equivalent and that the formulas for i, j, m, and n hold, it will then easily follow that (1), (2), and (3) are all equivalent.

Suppose that v is a pure min heap leaf node having exactly one min heap child w such that v is not a max heap child of w. Let $i\{m\}$ and $j\{n\}$ be the coordinates of v and w, respectively.

Suppose for the sake of contradiction that v had two min heap children, call the second one z. Were v not a child of z in the max heap then the uniqueness property of w would be violated. Thus v is a max heap child of z. But now condition (1) of proposition 4.2 is satisfied so by ((b)) of that proposition, z must be both the last node of the min heap and the left child of v. These two properties are impossible to have due to w being z's sibling in the min heap. Thus w is v's only child in the min heap. This implies that w is necessarily v's left child in the min heap, which proves part (a). Since v has no right child, part (d) now follows immediately from the definition of the BiHeap graph on N nodes where note that the statement of part (d) implies that v is uniquely determined so that there can be at most one node in the min heap having v's defining property.

By the symmetric construction of the BiHeap graph on N nodes, there must also exist a (now known to be unique) pure max heap leaf v' having exactly one max heap child w' such that v' is not a min heap child of w'. Let $i'\{m'\}$ and $j'\{n'\}$ be the coordinates of v' and w', respectively. That the node w' is the last node in the max heap implies that $n' = j$, $j' = n$, and $m' = \operatorname{Parent}(n') = \left\lfloor \frac{j-1}{2} \right\rfloor = \left\lfloor \frac{(2i+1)-1}{2} \right\rfloor = i$ so that $i' = \operatorname{Flip}(m') = \operatorname{Flip}(i) = m$. Note that if $j' \leq i$ then w' being in the max heap would necessitate that v, and hence also w, also belong to the max heap which would cause the max heap to have a cycle. Thus $j' > i$. Now suppose that $j' > i+1$. Let $k = i+1$, $p = \operatorname{Flip}(k)$, $p' = m+1$, and $k' = \operatorname{Flip}(p')$. Let z be the node whose coordinate is $k\{p\}$ and let z' be the node whose coordinate is $k'\{p'\}$. Since $n = j' > i+1 = m+1$, z does not belong to the max heap and z' does not belong to the min heap. So by adding an edge between v and z' to the min heap and another edge from v' to z to the max heap, we could extend both the min heap and the max heap by one edge, thereby contradicting the maximality of the heaps in the definition of the BiHeap graph on N nodes. Thus $j' \leq i+1$ so that $j' = i+1$. From $j' = i+1 = n = \operatorname{Flip}(j) = \operatorname{Flip}(\operatorname{LeftChild}(i))$, we conclude that $i = \frac{N}{3} - 1$ so that in particular, 3 divides N. From $j = \operatorname{LeftChild}(i)$, $m = \operatorname{Flip}(i)$, and $n = \operatorname{Flip}(j)$ we obtain the formulas for j, m, and n.

Now suppose that 3 divides N and use the N to define i, j, m, and n by the formulas given in this proposition's statement. Let v be the node whose coordinates are $i\{m\}$ and let w be the node whose coordinates are $j\{n\}$. It is straightforward to show that $j = \operatorname{LeftChild}(i)$ and that $\operatorname{Parent}(m) = \left\lfloor \frac{N}{3} - \frac{1}{2} \right\rfloor = \frac{N}{3} - 1 \neq \frac{N}{3} = n$, which shows that w is not the max heap parent of v.

Suppose that v has a min heap right child, call it z, whose coordinates would then necessarily be $k\{p\}$ where $k = j+1$ and $p = n-1$. Note that $\operatorname{Parent}(m) = \left\lfloor \frac{N}{3} - \frac{1}{2} \right\rfloor = \frac{N}{3} - 1 = n - 1 = p$ so that v is z's max heap child. Thus either v has no min heap right child or else its min heap right child is v's max heap parent. Either way, it follows that w is the unique child of v such that v is not a max heap child of w, which finishes the proof that (2) implies (3).

To prove (b), note that since 3 divides N, $\operatorname{Parent}(i) = \left\lfloor \frac{i-1}{2} \right\rfloor = \left\lfloor \frac{N}{6} \right\rfloor - 1$ so if N is even then $\operatorname{Parent}(i) = \frac{N}{6} - 1$, which implies that $\operatorname{LeftChild}(\operatorname{Parent}(i)) = 2\left(\frac{N}{6} - 1\right) + 1 = \frac{N}{3} - 1 = i$. Conversely, $i = \operatorname{Parent}(\operatorname{LeftChild}(\operatorname{Parent}(i)))$ implies $\frac{N}{3} - 1 = 2\left\lfloor \frac{N}{6} \right\rfloor - 1$ so that $\frac{N}{6} = \left\lfloor \frac{N}{6} \right\rfloor$, which shows that N is divisible by 6 and thus also by 2.

To prove (c), suppose first that N is odd. Since 2 does not divide $\frac{N}{3}$, $\left\lfloor \frac{N}{6} \right\rfloor = \left\lfloor \frac{N/3}{2} \right\rfloor = \frac{N/3}{2} - \frac{1}{2}$ so that $\operatorname{RightChild}(\operatorname{Parent}(i)) = \frac{N}{3} - 1 = i$. Conversely, if $i = \operatorname{RightChild}(\operatorname{Parent}(i))$ then $\frac{N}{3} - 1 = 2\left(\left\lfloor \frac{N}{6} \right\rfloor - 1\right) - 2$ gives us $\frac{N}{6} = \left\lfloor \frac{N}{6} \right\rfloor - \frac{1}{2}$ where since the right hand side is not an integer, neither is $\frac{N}{6}$ so that at least one of 2 and 3 does not divide N. Since 3 divides N, it follows that 2 does not divide N, so that N is odd.

To prove (g), let χ equal 1 if $\rho := \left\lceil \frac{N}{2} \right\rceil$ is even and 0 otherwise, let x be the desired quantity, and note that the number of BiHeap leaves is equal to both $6x + 4 + 2\chi + (N \mod 2)$ and $2\left\lceil \frac{\rho}{2} \right\rceil - (N \mod 2)$. ∎

4.4. BiHeaps Whose Heaps Lack Parents With a Single Child

Proposition 4.5. The following are equivalent:

(1) There is no min heap node with exactly one child.
(2) 3 divides $N - 1$ (or equivalently, $N \mod 3 = 1$).

For $N > 1$, we can add the following to the above list:

(3) If v is the last node of the min heap then $v = \operatorname{RightChild}_{\mathrm{Min}}(\operatorname{Parent}_{\mathrm{Min}}(v))$.

Suppose that (2) holds, let v be the last node in the pure min heap, and let $i\{m\}$ be its coordinates. Then

$$i = \frac{2(N-1)}{3} \qquad \text{and} \qquad m = \frac{N-1}{3}.$$

so that, in particular, i, j, m, n, v, and w are uniquely determined and furthermore,

(a) A node $k\{p\}$ is in the max heap if and only if $k \geq \frac{N-1}{3}$.
(b) A node $k\{p\}$ is in the min heap if and only if $k \leq \frac{2(N-1)}{3} = 2m$.
(c) The number of pure min heap leaves with exactly two children in the min heap is

$$\frac{1}{3}\left(\left\lceil \frac{\rho}{2} \right\rceil - 1 + (\rho \mod 2) - (N \mod 2)\right)$$

where $\rho = \left\lceil \frac{N}{2} \right\rceil$ is the size of the pure min heap.

Proof. If (1) hold then propositions 4.2 and 4.3 imply that $N \mod 3 \neq 2$ and $N \mod 3 \neq 0$, respectively, so that (2) necessarily holds. The converse also follows immediately from these two propositions. Similarly, these same propositions prove that (2) and (3) are equivalent.

To prove (c), let χ equal 1 if $\rho := \left\lceil \frac{N}{2} \right\rceil$ is even and 0 otherwise, let x be the desired quantity, and note that the number of BiHeap leaves is equal to both $6x + 2\chi + (N \mod 2)$ and $2\left\lceil \frac{\rho}{2} \right\rceil - (N \mod 2)$.

Assume now that v and $i\{m\}$ are as described and let $\rho = \left\lceil \frac{N}{2} \right\rceil$. Let us determine how many nodes are in the min heap. Let

$$\chi = \begin{cases} 1 & \text{if } \rho \text{ is even} \\ 0 & \text{if } \rho \text{ is odd} \end{cases} \qquad \text{and} \qquad \alpha = \begin{cases} 0 & \text{if } N \text{ is even} \\ 1 & \text{if } N \text{ is odd} \end{cases}$$

Let τ be thrice the number of pure min heap leaves have exactly two children in the min heap. Note that:

(1) Every pure min heap leaf is incident to either 2 or 0 extended min heap edges.
(2) If ρ is even then by proposition 4.1, node $\operatorname{Parent}(\rho - 1)$ has exactly one extended min heap edge going to the last (if N is odd, also non-middle) node of the pure max heap, and by symmetry of the BiHeap, there is a similar edge going from the last interior pure max heap node into the last (non-middle) leaf of the pure min heap.
(3) If N is odd, then the node $\rho - 1$ is incident to both a pure min edge and a pure max edge.

From this we conclude that the min heap has exactly $\mu := \rho + \frac{2}{3}\tau + \chi$ nodes and that $\tau = \left\lceil \frac{\rho}{2} \right\rceil - \alpha - \chi$, where recall that $\left\lceil \frac{\rho}{2} \right\rceil$ is the number of leaves in the pure min heap and we subtract 1 if the pure min and max heap share a node and we also subtract 1 if there is a edge going from the last interior pure max heap node into the last pure min heap node. Note that among the τ nodes that were counted, for every node incident to 2 extended min heap edges, there are two such nodes not incident to any min heap edge, which is why the $\frac{2}{3}\tau$ was added while χ was added to count, in the case where ρ is even, the last pure max heap node (which will necessarily belong to the min heap). From this we get (c).

We will now go through the four cases of ρ and N being even or odd to show that $\mu = \frac{2N+1}{3}$ in each of them. If ρ is even then $\mu = \rho + \frac{2}{3}(\rho/2 - 1 - \alpha) + 1 = \frac{4\rho+1-2\alpha}{3}$ so if N is even then $\rho = \frac{N}{2}$ so $\mu = \frac{2N+1}{3}$ while if N is odd then $\rho = \frac{N+1}{2}$ so $\mu = \frac{2(N+1)+1-2}{3} = \frac{2N+1}{3}$. If ρ is off then $\tau = \frac{\rho+1}{2} - \alpha$ and $\mu = \rho + \frac{2}{3}\left(\frac{\rho+1}{2} - \alpha\right) = \frac{4\rho+1-2\alpha}{3}$ so if N is even then $\mu = \frac{2N+1}{3}$ while if N is odd then $\mu = \frac{2N+1}{3}$. This implies the formula for i, which in turn implies the formula for $m = \operatorname{Flip}(i)$ and statements (a) and (b). ∎

4.5. Computing HeapSize (N) From propositions 4.2, 4.3, and 4.5 we conclude that for all $\mathrm{N} > 0$,

$$\begin{aligned}\mathrm{HeapSize}(N) &= \mathrm{N} - \lfloor (\mathrm{N} - (\mathrm{N} \mod 3))/3 \rfloor \\ &= \mathrm{N} - \lfloor \mathrm{N}/3 \rfloor \\ &= \left\lceil \frac{2\mathrm{N}}{3} \right\rceil \\ &= \frac{2\mathrm{N} + (\mathrm{N} \mod 3)}{3}\end{aligned}$$

Similarly, these two propositions imply that for $\mathrm{N} > 3$, the number of pure min heap nodes incident to exactly one extended min heap edge is

$$((\rho + 1) \mod 2) + (((\mathrm{N} \mod 3) + 1) \mod 2).$$

where we let $\rho = \left\lceil \frac{\mathrm{N}}{2} \right\rceil$. They also imply that for $\mathrm{N} > 3$, the number of pure min heap leaves with exactly two children in the min heap is

$$\frac{1}{3}\left(\left\lfloor \frac{\rho}{2} \right\rfloor - (\mathrm{N} \mod 2) - 1 + (\rho \mod 2) - ((\mathrm{N}+2) \mod 3)\right)$$

while it is 0 for $\mathrm{N} \leq 3$.

Part 3. BiHeapification

Having obtained closed form formulas for HeapSize (), we can now define a function to take any collection of N elements and form a directed BiHeap with respect to any given total ordering on these elements.

5. Overview of BiHeapification Recall that if we want to heapify N elements using an $O(\mathrm{N})$ algorithm, then we proceed inductively by first heapifying elements $\mathrm{N} - (k-1), \ldots, \mathrm{N} - 1$ (which in a sense constitute "the current heap") and then we sift down element $N - k$ by at most $\lceil \log_2(\mathrm{N}+1) \rceil - \lceil \log_2(k+2) \rceil$ levels. Our BiHeapify () algorithm will essentially start by doing the same thing, except that after it has sifted down to an element that belongs to both the min heap and the max heap, it will begin to sift up.

For notational reasons, we describe the operation for the case where N is odd. The description in the case where N is even is almost identical, but for the notation of "the current BiHeap" that will be defined shortly. If $\rho = \left\lfloor \frac{\mathrm{N}}{2} \right\rfloor$, then the BiHeapify () algorithm will proceed inductively as follows:

(1) Suppose that we have BiHeapified elements $\rho - (k-1), \ldots, \rho - 1, \rho, \rho + 1, \ldots, \rho + (k-1)$, which we will call the *current BiHeap*, where this of course means that those elements in the min heap satisfy the min heap condition while those in the max heap satisfy the max heap condition. The elements that belong to both the min (resp. max) heap and the current BiHeap will be referred to as the *current min (resp. max) heap.*
(2) We now take element $\rho - k$, which necessarily belongs to the pure min heap, and sift it down the current min heap until either it has entered the current max heap or otherwise can go no further down the current min heap. If it enters into the max heap then we begin to sift up in the current max heap until we can go no further up. It is important to note here that all of this sifting takes place within the current BiHeap since this will end up guaranteeing that the BiHeapify () operation is $O(\mathrm{N})$.
(3) At this point, we will have BiHeapified elements $\rho - k, \rho - (k-1), \ldots, \rho, \ldots, \rho + (k-1)$.
(4) We now repeat the process in (2) with element $\rho + k$ except that, since we are now starting in the pure max heap, we reverse the roles of the current min heap and the current max heap. That is, we first sift down the current max heap and then sift up the current min heap.
(5) At this point, we have BiHeapified elements $\rho - k, \ldots, \rho, \ldots, \rho + k$ so if $k = \rho$ then we're done. Otherwise, we start (1) with k in place of $k - 1$ and continue.

Note that all of the above operations consist of nothing more than the usual operations to sift up/down a min/max heap so its implementation requires just slightly more work than what it takes to fully implement both a min heap and a max heap.

Now, performing (2) requires moving element $\rho - k$ no more than $2d + 1$ levels, where $d = \lceil \log_2(\rho+1) \rceil - \lceil \log_2(k+2) \rceil$, and the same is true of performing (4). From here, it should be now obvious that the usual proof that the heapify operation is $O(\mathrm{N})$ generalizes immediately to proving that the BiHeapify () operation is $O(\mathrm{N})$.

6. Bounded Sifting Up Heaps Throughout, if a variable's name is suffixed with "hc" (resp. "mc"), which stands for "(min) heap coordinate" (resp. "mirror coordinate"), then that variable stores the (min) heap coordinate (resp. mirror/max heap coordinate) of some node. Any variable suffixed with "nd" represents a node. For the reader who is unfamiliar with the notion of iterators, it is sufficient for our needs to consider "iterator" and "node" to be synonyms. The function SwapValues (v, w) can then be considered to simply be a function that swaps the values $^*v$ and $^*w$. The variables beginning with "pos" store "the current node"'s position.

The following functions are nothing more than the usual sift up functions for min and max heaps, except that the operations are limited to some given range of nodes. That is, the sifting operation stops once a node's min heap or max heap coordinate goes below a certain lower bound. This is what will guarantee that the BiHeapify () operation is $O(\mathrm{N})$.

```
//Assumes that pos_mc is a node in the max heap.
void SiftUpMaxHeapMC(Node first, int N, int pos_mc, int smallest_node_in_max_heap_mc) {
  if (pos_mc == 0 || Parent(pos_mc) < smallest_node_in_max_heap_mc)
    return ;
  int  parent_mc = Parent(pos_mc);
  Node pos_node  = first + Flip(pos_mc);
  do {
    parent_mc        = Parent(pos_mc);
    Node parent_node = first + Flip(parent_mc);
    if (*pos_node > *parent_node) {
      SwapValues(pos_node, parent_node);
      pos_mc   = parent_mc;
      pos_node = parent_node;
    } else {
      return ;
    }
  } while (pos_mc > 0 && Parent(pos_mc) >= smallest_node_in_max_heap_mc);
  return ;
}

//Assumes that pos_hc is a node in the max heap.
void SiftUpMaxHeapHC(Node first, int N, int pos_hc, int smallest_node_in_max_heap_mc) {
  SiftUpMaxHeapMC(first, N, Flip(pos_hc), smallest_node_in_max_heap_mc);
}

//Assumes that pos_hc is a node in the min heap.
void SiftUpMinHeapHC(Node first, int pos_hc, int smallest_node_in_biheap_hc) {
  if (pos_hc == 0 || Parent(pos_hc) < smallest_node_in_biheap_hc)
    return ;
  int  parent_hc = Parent(pos_hc);
  Node pos_node  = first + pos_hc;
  do {
    parent_hc        = Parent(pos_hc);
    Node parent_node = first + parent_hc;
    if (*pos_node < *parent_node) {
      SwapValues(pos_node, parent_node);
      pos_hc   = parent_hc;
```

```
      pos_node = parent_node;
    } else {
      return ;
    }
  } while (pos_hc > 0 && Parent(pos_hc) >= smallest_node_in_biheap_hc);
  return ;
}
```

.

7. The BiHeapify() **Algorithm** To implement BiHeapify(), we must first implement two other functions, the first (resp. second) of which takes an element in the min (resp. max) heap and sifts it down until it reaches the max (resp. min) heap, at which point it proceeds to sift the element up for as long as it remains within bounds (as was discussed in the overview (2) above).

```
//Assumes that the node pos_hc belongs to the min heap
// and that pos_hc <= largest_node_in_biheap_hc.
void BiHeapifySiftFromMinToMax(Node first, int N, int heap_size,
                               int first_node_in_mirror_heap, int pos_hc,
                               int largest_node_in_biheap_hc) {
  //Sift down the min heap while not yet in the max heap.
  while (pos_hc < first_node_in_mirror_heap) {
    int  left_child_hc  = LeftChild(pos_hc);
    int  right_child_hc = RightChild(pos_hc);
    Node left_node      = first + left_child_hc;
    Node right_node     = first + right_child_hc;
    Node pos_node       = first + pos_hc;
    bool is_right_child_valid = right_child_hc <= largest_node_in_biheap_hc &&
                                right_child_hc < heap_size;
    Node smaller_node;
    if (is_right_child_valid && *right_node < *left_node) {
      smaller_node = right_node;
      pos_hc       = right_child_hc;
    } else {
      smaller_node = left_node;
      pos_hc       = left_child_hc;
    }
    if (*pos_node > *smaller_node)
      SwapValues(pos_node, smaller_node);
    else
      return ;
  }
  SiftUpMaxHeapHC(first, N, pos_hc, Flip(largest_node_in_biheap_hc));
  return ;
}

//Assumes that the node pos_mc belongs to the max heap
// and that Flip(pos_mc) >= smallest_node_in_biheap_hc.
void BiHeapifySiftFromMaxToMin(Node first, int N, int heap_size,
                               int first_node_in_mirror_heap, int pos_mc,
                               int smallest_node_in_biheap_hc) {
  auto pos_hc = Flip(pos_mc);
  //Sift down the max heap while not yet in the min heap.
  while (pos_mc < first_node_in_mirror_heap) {
    int  left_child_mc  = LeftChild(pos_mc);
    int  right_child_mc = RightChild(pos_mc);
    int  left_child_hc  = Flip(left_child_mc);
    int  right_child_hc = Flip(right_child_mc)
    Node pos_node       = first + pos_hc;
```

```
    Node left_node        = first + left_child_hc;
    Node right_node       = first + right_child_hc;
    //Note that right_child_hc >= smallest_node_in_biheap_hc necessarily holds.
    bool is_right_child_valid = right_child_mc < heap_size;
    Node larger_node;
    if (is_right_child_valid && *right_node > *left_node) {
      larger_node = right_node;
      pos_hc      = right_child_hc;
      pos_mc      = right_child_mc;
    } else {
      larger_node = left_node;
      pos_hc      = left_child_hc;
      pos_mc      = left_child_mc;
    }
    if (*pos_node < *larger_node)
      SwapValues(pos_node, larger_node);
    else
      return ;
  }
  SiftUpMinHeapHC(first, pos_hc, smallest_node_in_biheap_hc);
  return ;
}

int IndexOfLastMinHeapNode(int size_of_heap) {
  return size_of_heap - 1;
}

void BiHeapify(Node first, int N) {
  if(N < 2)
    return ;
  int heap_size                    = HeapSize(N);
  int first_node_in_mirror_heap    = N - heap_size;
  //Ignore all extended in nodes,
  int largest_node_in_biheap_hc    = IndexOfLastMinHeapNode(heap_size);
  if (N % 3 == 2)                  //Unless N % 3 == 2, in which case
    largest_node_in_biheap_hc--; //don't ignore the double headed arrow's nodes.
  int smallest_node_in_biheap_hc = Flip(largest_node_in_biheap_hc);
  while (smallest_node_in_biheap_hc > 0) {
    smallest_node_in_biheap_hc--; //Sift the next pure min heap node.
    BiHeapifySiftFromMinToMax(first, N, heap_size, first_node_in_mirror_heap,
                              smallest_node_in_biheap_hc, largest_node_in_biheap_hc);
    largest_node_in_biheap_hc++; //Sift the next pure max heap node.
    BiHeapifySiftFromMaxToMin(first, N, heap_size, first_node_in_mirror_heap,
                         Flip(largest_node_in_biheap_hc), smallest_node_in_biheap_hc);
  }
  return ;
}
```

.

Part 4. Sifting an Element

If we replace some element in a BiHeap with another, it may cease to be a BiHeap. We can correct this with an $O(\log N)$ algorithm that consists of the algorithm obtained by replacing, in the above "sift" functions used in defining BiHeapify (), every instance of smallest_node_in_biheap_hc with 0 and every instance of largest_node_in_biheap_hc with N − 1. We describe this function although it will not be used elsewhere.

```
inline void BiHeapSift(Node first, int N, int pos_hc) {
```

```
  int heap_size            = HeapSize(N);
  int first_node_in_mirror_heap = N - heap_size;
  if (pos_hc < heap_size) { //If the node is in the min heap.
    int parent_hc = Parent(pos_hc);
    if (pos_hc != 0 && *(first + parent_hc) > *(first + pos_hc))
      SiftUpMinHeapHCfirst, pos_hc, 0);
    else
      BiHeapifySiftFromMinToMax(first, N, heap_size, first_node_in_mirror_heap,
                                 pos_hc, N - 1);
    return ;
  }
  //Else the node is in the max heap.
  int pos_mc    = FLIP(pos_hc);
  int parent_mc = Parent(pos_mc);
  int parent_hc = Parent(parent_mc);
  if (pos_mc != 0 && *(first + parent_hc) < *(first + pos_hc))
    SiftUpMaxHeapMC(first, N, pos_mc, 0);
  BiHeapifySiftFromMaxToMin(first, N, heap_size, first_node_in_mirror_heap,
                             pos_mc, 0);
  return ;
}
```

.

Part 5. Recursively BiHeapifying Extended In Nodes with BiHeapifyInwards()

8. Ins and Outs

Notation 8.1. If T is a tree then let $\Lambda(T)$ denote the leaves of the tree. ■

Notation 8.2. For any $N > 0$, let $\mathrm{In}_{\mathrm{PMin}}(\mathrm{B}_N)$ $(\mathrm{In}_{\mathrm{PMax}}(\mathrm{B}_N))$ denote those leaves of $\mathrm{PMin}(N)$ (resp. $\mathrm{PMax}(N)$) that belong to the max (resp. min) heap. and let $\mathrm{In}_{\mathrm{PMin}}(N) = |\mathrm{In}_{\mathrm{PMin}}(\mathrm{B}_N)|$ (resp. $\mathrm{In}_{\mathrm{PMax}}(N) = |\mathrm{In}_{\mathrm{PMax}}(\mathrm{B}_N)|$). Let $\mathrm{In}(\mathrm{B}_N)$ denote the union of $\mathrm{In}_{\mathrm{PMin}}(\mathrm{B}_N)$ and $\mathrm{In}_{\mathrm{PMax}}(\mathrm{B}_N)$. ■

Observe in particular that

- $\mathrm{In}(\mathrm{B}_N)$ is just the set of nodes common to both the min heap and the max heap.
- $\mathrm{In}(\mathrm{B}_2) = \mathrm{B}_2$.
- If N is odd then $\mathrm{In}(\mathrm{B}_N)$ contains the BiHeap's middle node.
- $\mathrm{In}(\mathrm{B}_3)$ consists solely of the B_3's middle node.
- $|\mathrm{In}(\mathrm{B}_N)| = N - 2(N - \mathrm{HeapSize}(N)) = 2\left\lceil \frac{2N}{3} \right\rceil - N = \frac{N+2(N \mod 3)}{3}$ so that in particular, $3|\mathrm{In}(\mathrm{B}_N)| - 4 \leq 3|\mathrm{In}(\mathrm{B}_N)| - 2(N \mod 3) = N \leq 3|\mathrm{In}(\mathrm{B}_N)|$.

9. Minimum Number of Elements Below and Above Pivots

Definition 9.1. Let T be a rooted tree on N nodes. By *an ancestor* of a node v in T we mean any node, including v, that lies along the unique shortest path from v to the root. If v is a node in T then $\mathrm{Ancestors}(v)$ denotes the set of all ancestors of v in T. If S is a subset of T then $\mathrm{Ancestors}(S)$ denotes $\bigcup_{v \in S} \mathrm{Ancestors}(v)$. ■

Lemma 9.2. Let B be rooted binary tree on $N > 1$ nodes. Let Λ be a non-empty subset of B's leaves and let $T = \mathrm{Ancestors}(\Lambda)$. Let S denote the set of nodes v in T that have exactly one child in T (i.e. those non-root nodes in T that have degree 2 in T and including the root if it has degree 1 in T). Then T is a binary sub-tree of B rooted at B's root and if there is a node in B that is not in T then S is not empty.

Proof. It is clear that T is a binary rooted at B's root. Suppose for the sake of contradiction that S is empty and observe that this implies that every node in T that is not in Λ has exactly two children in T. In particular, the root must have two children in T. Let v_0 be any node in B that is not in T and note that since the root is in T, there must be some node $v \in B \setminus T$ along the unique path from v_0 to the root such that v's parent, call it p, belongs to T. Since p is not a leaf in B, p does not belong to Λ but since p belongs to $T = \text{Ancestors}(\Lambda)$, it must therefore have a child that does belong to T. Thus p is a node with exactly one child in T, which contradicts the assumption that S is empty. ∎

Proposition 9.3. Suppose that the $N > 0$ and that the values $^*\text{first}, \ldots, ^*(\text{first} + (N-1))$ form a BiHeap, B. Let $I = \text{In}(B)$ and let pivot_value be the value of some element in I. Let Σ (resp. Γ) denote the set of values in B that are less than or equal to (resp. greater than or equal to) pivot_value. Let C denote $\Sigma \cap I$ (resp. $\Gamma \cap I$). Let χ be 1 if $N \mod 3 = 2$ and C contains the last node of the min (resp. max) heap and let it be 0 otherwise.

If $C = I$ then $|\Sigma| \geq \text{HeapSize}(N) = \left\lceil \frac{2N}{3} \right\rceil = 2|\Sigma \cap I| - (N \mod 3)$ (resp. $|\Gamma| \geq \left\lceil \frac{2N}{3} \right\rceil = 2|\Gamma \cap I| - (N \mod 3)$).

If any of the following conditions hold

(1) $N \mod 3 = 0$,
(2) $N \mod 3 = 1$ and $C \neq I$,
(3) $N \mod 3 = 2$, $|C| < |I| - 1$, and C does not contain the last node of the min (resp. max) heap.

then $|\Sigma| \geq 2(|\Sigma \cap I| - \chi)$ and $\frac{|\Sigma|}{N} \geq \frac{2}{3}\frac{(|\Sigma \cap I| - \chi)}{|I|}$ (resp. $|\Gamma| \geq 2(|\Gamma \cap I| - \chi)$ and $\frac{|\Gamma|}{N} \geq \frac{2}{3}\frac{(|\Gamma \cap I| - \chi)}{|I|}$).

Proof. Note that if $N = 5$ and Σ does not contain the last node of the min heap then the conclusion can be immediately verified. So assume that $N \neq 5$. If $N = 2, 4$, or 7 and $\Sigma \cap I \neq I$ then the conclusion can be directly verified. The conclusion can also be directly verified for $N = 3$ and $N = 6$ so assume that N is neither of these. We may thus assume that $N > 7$.

Assume for now that $C = I$. Then all of the min heap's leaves (and thus also the min heap itself) belongs to Σ so that $|\Sigma| \geq \text{HeapSize}(N) = \left\lceil \frac{2N}{3} \right\rceil = \frac{N + 2(N \mod 3)}{3}$. Now, $2|\Sigma \cap I| - (N \mod 3) = 2|I| - (N \mod 3) = 2\left(\frac{N+2(N \mod 3)}{3}\right) - (N \mod 3) = \text{HeapSize}(N)$. Note in particular that if $N \mod 3 = 0$ then $|\Sigma| \geq \text{HeapSize}(N) = 2|\Sigma \cap I|$. With this observation, it now suffices to prove the conclusion under the assumption that $C \neq I$.

Note that $\Sigma \cap I$ is contained in the leaves of the min heap so let T denote the set $\Sigma \cap I$ together all all of $\Sigma \cap I$'s min heap ancestors. It follows from the definition of a min heap that T forms a binary tree rooted at first. Note that if $N \mod 3 \neq 2$ then all of the remaining assumption imply that there exists some leaf in the min heap that does not belong to T. Lemma 9.2 now allows us to conclude that T necessarily contains some node with exactly one child. If, however, $N \mod 3 = 2$ then the assumption that $|C| < |I| - 1$ and that C does not contain the last node of the min heap allow us, by simply ignoring the last node of the min heap, to apply this previous reasoning again to conclude that T necessarily contains some node with exactly one child.

Recall that in any binary tree, if l is the number of leaves in the tree and if d_1 (resp. d_2) is the number of nodes in the tree with exactly one child (resp. exactly two children), then $d_2 = l - 1$ and the number of nodes in the tree is $2l + d_1 - 1$. Thus $|T| \geq 2\Lambda(T) = 2|\Sigma \cap I|$. Since $T \subseteq \Sigma$, we have the desired conclusion.

Recalling that $3|\text{In}(\text{B}_N)| - 4 \leq 3|\text{In}(\text{B}_N)| - 2(N \mod 3) = N \leq 3|\text{In}(\text{B}_N)|$, the inequality $\frac{|\Sigma|}{N} \geq \frac{2}{3}\frac{|\Sigma \cap I|}{|I|}$ now follows from $\frac{1}{N} \geq \frac{1}{|I|}$.

A similar argument proves the inequalities involving Γ. ∎

10. The BiHeapifyInwards() **Algorithm** We want a function that finds a pivot value with a guaranteed minimum number of values less/greater than or equal to this pivot value. In light of proposition 9.3, the following $O(\text{N})$ function is the most natural candidate. It finds a pivot value by applying BiHeapify(), restricting the list of nodes to the BiHeap's extended in nodes, and then repeating.

```
void BiHeapifyInwards(Node first, int N) {
  if (N < 10) {
    sort(first, first + N);
```

```
    return;
  }
  BiHeapify(first, N);          //This is an O(N) operation.
  int heap_size                = HeapSize(N);
  int first_extended_in_node = N - heap_size;//= Flip(heap_size - 1);
  int num_extended_in_nodes   = heap_size - first_extended_in_node;
  //The inequality below necessarily holds, which implies that this is an O(N) algorithm.
  assert(num_extended_in_nodes <= (N + 4) / 3);
  BiHeapifyInwards(first + first_extended_in_node, num_extended_in_nodes);
  return ;
}
```

.

Since this is a tail recursive function, it can also be implemented as a loop in which case it will then use $O\,(\text{constant})$ additional memory.

Definition 10.1. Suppose that we're given a list of N values $*\,\text{first}, \ldots, *\,(\text{first} + (\text{N} - 1))$ and that after calling BiHeapifyInwards () on this list, this list of values has been permuted so as to become the list $*\,(\text{first} + \iota(0)), \ldots, *\,(\text{first} + \iota(\text{N} - 1))$.

(1) If N is even then by *the* BiHeapifyInwards ()*-left pivot value* (resp. *the* BiHeapifyInwards ()*-right pivot value*) of this list, denoted by $\text{pivot\_value}_{\text{L}}$ (resp. $\text{pivot\_value}_{\text{R}}$), we mean the value $*\left(\text{first} + \iota\left(\frac{\text{N}}{2} - 1\right)\right)$ (resp. $*\left(\text{first} + \iota\left(\frac{\text{N}}{2}\right)\right)$).
(2) If N is odd then
   (a) by *the* BiHeapifyInwards ()*-pivot value* of this list, denoted by pivot_value, we mean the value $*\left(\text{first} + \iota\left(\frac{\text{N}-1}{2}\right)\right)$.
   (b) by *the* BiHeapifyInwards ()*-left pivot value* (resp. *the* BiHeapifyInwards ()*-right pivot value*) of this list, denoted by $\text{pivot\_value}_{\text{L}}$ (resp. $\text{pivot\_value}_{\text{R}}$), we mean the BiHeapifyInwards ()-pivot value.

■

However, although this function is, relative to proposition 9.3, the simplest and most natural candidate for a function that finds a desirable pivot value, it has the unfortunate property that the conditions of proposition 9.3 that give desired inequalities are not necessarily satisfied. We now show how a slight modification rectifies this issue. First, we will need any $O\,(\text{N})$ function that moves a maximal element to the end of the given list and moves a minimal element to the start of the list. The following function satisfies these requirements.

```
void EmplaceMinAndMax(Node first, int N) {
  BiHeapify(first, N);
  return ;
}
```

.

The function used to obtain the desired pivots is now defined as follows.

```
void BiHeapifyInwardsNicerMath(Node first, int N) {
  if (N < 10) {
    sort(first, first + N);
    return;
  }
  int new_N = N;
  if (new_N % 3 == 1) {
    EmplaceMinAndMax(first, new_N);
    first = first + 1;
    new_N = new_N - 2;
  }
  if (new_N % 3 == 2) {
    EmplaceMinAndMax(first, new_N);
```

```
    first = first + 1;
    new_N = new_N - 2;
  }
  //At this point, new_N % 3 == 0.
  BiHeapify(first, new_N);
  int heap_size               = HeapSize(new_N);
  int first_extended_in_node = new_N - heap_size;
  int num_extended_in_nodes  = heap_size - first_extended_in_node;
  BiHeapifyInwardsNicerMath(first + first_extended_in_node, num_extended_in_nodes);
  return ;
}
```

.

It now immediately follows from proposition 9.3 that as N increases exponentially so too does the number of values that are less (resp. greater) than or equal to the BiHeapifyInwardsNicerMath ()-left (resp. right) pivot value, for *any* set of N inputs. We state this formally.

Theorem 10.2. Suppose that the $N > 0$ and that the values $^*\text{first}, \ldots, ^*(\text{first} + (N-1))$, denoted by B, are passed to BiHeapifyInwardsNicerMath (). Let $\text{pivot\_value}_{\text{L}}$ (resp. $\text{pivot\_value}_{\text{R}}$) be the BiHeapifyInwardsNicerMath ()-left (resp. right) pivot value (def. 10.1) and let Σ (resp. Γ) denote the set of values in B that are less than or equal to $\text{pivot\_value}_{\text{L}}$ (resp. greater than or equal to $\text{pivot\_value}_{\text{R}}$). Then $|\Sigma| \geq 2^{\lceil \log_3 N \rceil}$ and $|\Gamma| \geq 2^{\lceil \log_3 N \rceil}$, where observe that $2^{\lceil \log_3 N \rceil} \geq N^{\frac{1}{\log_2 3}}$ and $\frac{1}{\log_2 3} \approx 0.6309$.

Proof. Use induction and the inequalities in lemma 9.3. ∎

Now although $\mathrm{N}^{\frac{1}{\log_2 3}}$ is a lower bound for $|\Sigma|$ and $|\Gamma|$, testing on randomly generated data shows that on average, one should expect the minimum of $|\Sigma|$ and $|\Gamma|$ to be approximately $0.48N$. This means that in applications, this $O(\mathrm{N})$ function provides a pivot that is a very good approximation of the median.

For actual applications, the author recommends using BiHeapifyInwards () rather than BiHeapifyInwardsNicerMath () since it is less computationally intensive, produces a pivot value that is generally just as good as that produced by BiHeapifyInwardsNicerMath (), and, by virtue of the inequality in proposition 9.3, is guaranteed to have a minimum of *approximately* $\mathrm{N}^{\frac{1}{\log_2 3}}$ values above and below it (although the exact expression for this lower bound is of course much more complicated than that given in theorem 10.2).