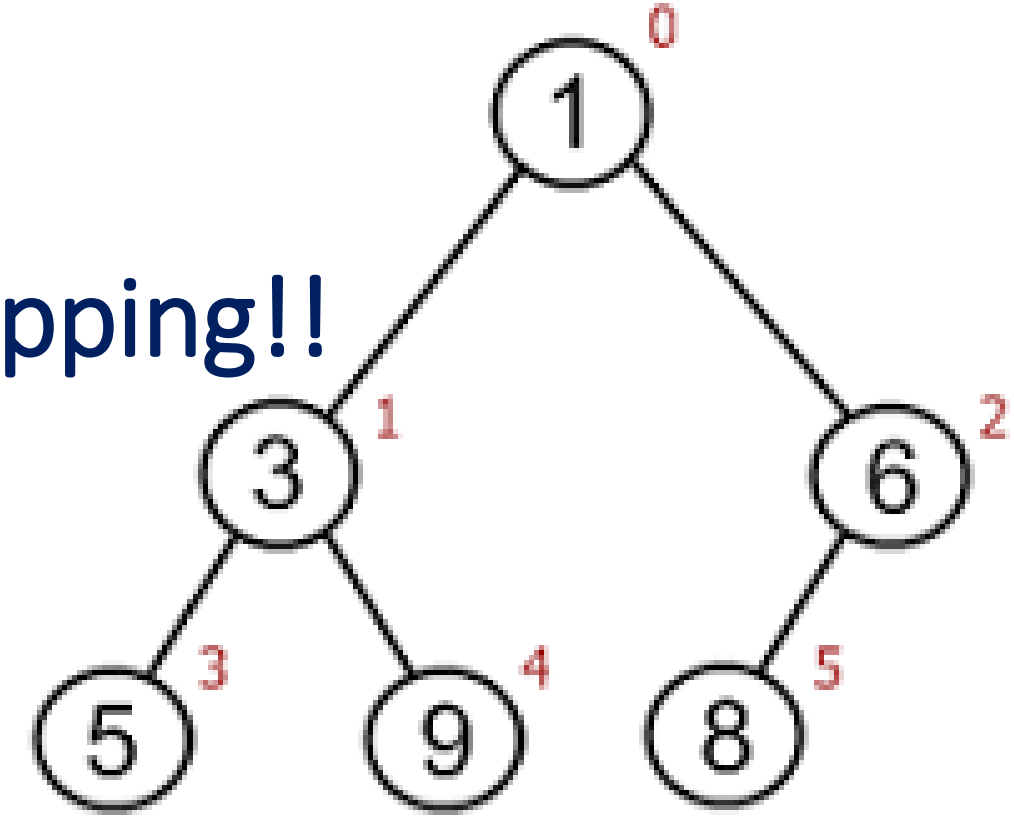
Extra Credit Homework: #4

Heap to Array Mapping!!



Value	1	3	6	5	9	8
Index	0	1	2	3	4	5

Position Mapping:

P	**Cs**
0 →	1, 2
1→	3,4
2→	5

N → 2N+1, 2N+2

Computing Heap Array Indices

Node index	Parent index	Child indices
0	N/A	1, 2
1	0	3, 4
2	0	5, 6
3	1	7, 8
4	1	9, 10
5	2	11, 12
…	…	…
i	$\lfloor (i-1)/2 \rfloor$	2 * i + 1, 2 * i + 2

➔ Node Index 57: Parent = 56/2 = 28. Children 115, 116

➔ Node index 58; Parent = 57/2 = 28, Children 117, 118

Objects / Structures you will need

The elements contained in the Heap Data Structure will be pointers to “hItem”s.

There are four aspects of this which will be defined on the next few pages:

- **hitem structure** (Contains Priority and Student Class Ptr)
- **Student class** (Identical to that in HM 1)
- **EHI class** (Orderable Array whose elements are hItem* instead of Student*)
- **MaxHeap class** (Supports MAX priority queues with “push / pop / look”)

This homework will involve writing one of the MaxHeap class functions

Array Heap Item & Student Class

```
// hItem:  Array Heap Item
   struct hItem {    // Element being "heaped"
   int priority;    // Priority of Node (~ the Array Item "key")
   Student* studp;  // Ptr to Thing represented in Heap
};
```

// Student.h … If you need the Student class code (unlikely) take it from HM 1

```
class Student  {
private:
   int sid; // Student ID
   string sname; // Full Name (Ex: KleinmanRon)
public:
   Student();  Student(const Student&);  Student(int, string);  ~Student();
   //Getters
   string getName() { return (sname); };
   int getId() { return (sid); };
};
```

EHI: HM #1 Orderable Array for *hItem* Pointers

```
class EHI {  // Enhanced Array of Heap Item Pointers (All "resizing" auto-done here)
   public:
      EHI (int initSz); // Create internal hItem* Array of that Size
      EHI (ESI&) (); ~EHI();  // Copy constructor & Destructor
       unsigned int getNum ();  // Size of Array
       hItem* get (int index); // Get elem at index // Return -1 if specified index illegal
     int set (int index, hItem*ep); // Overwrite existing elem at index
     int append (hItem* ep); // Append to back. Resize if needed
     Int prepend (hItem* ep); // Prepend to front, push others back.  Resize if needed
     int remove (int index);  // Remove element, move others down
     int insert (int index, hItem *ep); // Insert  element, push others back. Resize if needed
     };
```

Max Heap API (based on Orderable Array Data Structure

```
class MaxHeap {  // Collection is max Heap.  Element 0 is Root
   private:
        EHI* ehi; // Ptr to Orderable array (of hItem ptrs)
                                  // 0th element is Heap root (maximum)
        int hmax;  // Current size of the EHI (from EHI::getNum())
        int hidx;  // Index to current EHI element
  public:
        MaxHeap(int); MaxHeap (MaxHeap&); ~MaxHeap();
        int push (hItem* hi); // Push to bottom, percolate up
        hItem*  pop(); // Pops root & repositions lower nodes
                                  // Size adjusted (so last node known)
        hItem*  look() const;  // Peeks at Root Node (No Pop)
};
```

Max Heap Array: Push (& Percolate Up): Pseudocode

```
// Add new item to heap array
// Rebalance and return position of where new item is
      Append current item to back
//Perculate up
      While (entry !=root)  {    // Check if priority > parent
            Get parent index
            Get ptr to parent hItem from EHI
            If (new priority > parent)  // Swap
                  Store appended item in parent index
                  Store parent item in current index of appended item
                  Set current item index to parent index
            else
                  break   // Node in correct position
       }
      Return current item final index // Node percolated
```

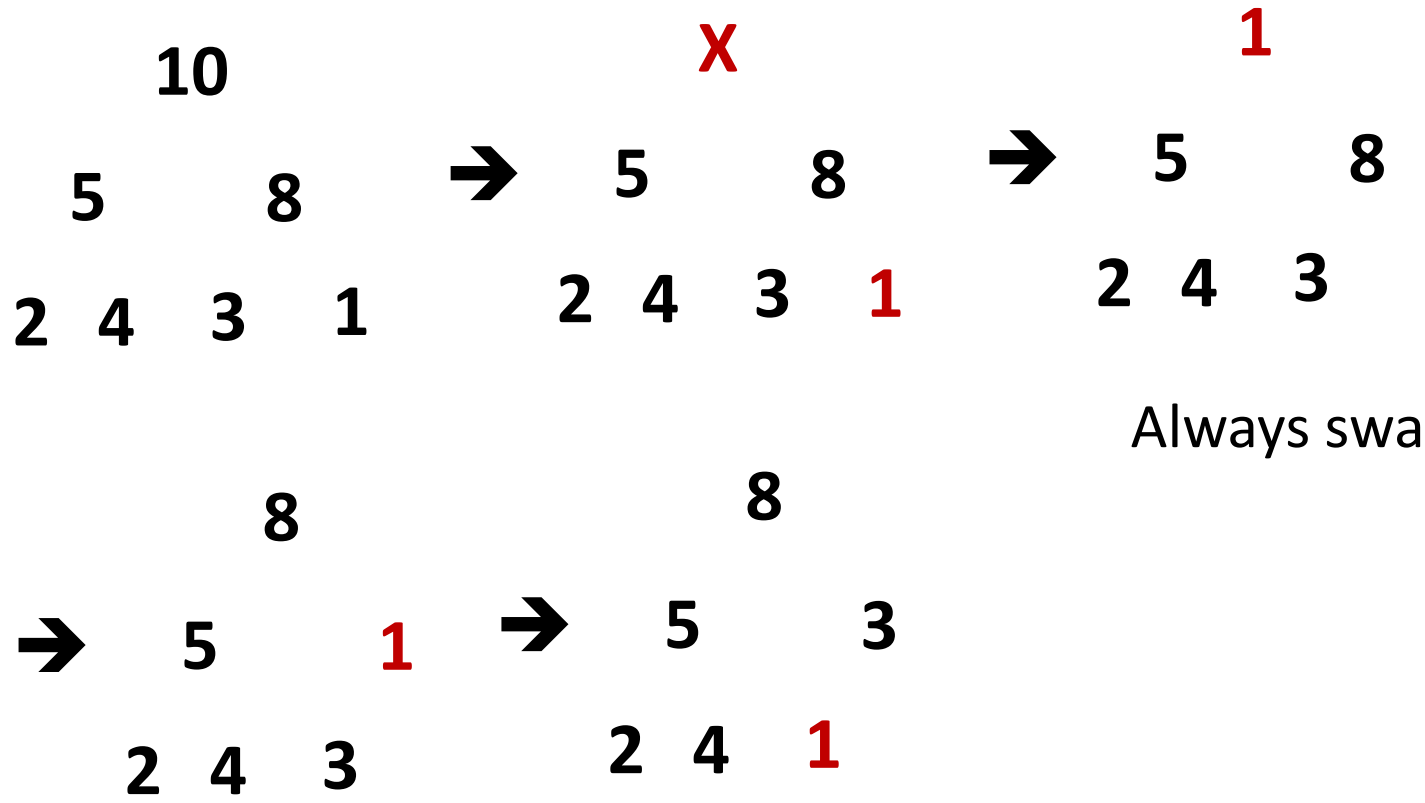
Max Heap Array: Append & Percolate Up

```
int maxHeap:: push (hItem* hp) { // Add new item* to heap array (hmax & hidx private data in .h)
        // Rebalance and return position of where new item is
    int pidx = 0; // Parent item index
    hItem*pp = null; // Holding item ptr of parent
    hidx = ehi->append(hp);//Append current element to back.  Then percolate up
     while  hidx > 0   { // While entry !=root, check if new priority > Parent priority
        pidx = (hidx -1) /2;  // Get Parent index
        pp = ehi->get(pidx); // Get ptr to Parent Item
        if ((hp->priority) > (pp->priority)) { // New priority > Parent priority. Swap
            ehi->set (pidx, hp);
            ehi->set (hidx, pp);
            hidx = pidx;  // Set current index to parent index, repeat
         }
         else  break; // It isn't greater.  Node in correct position
    }  return (hidx);  // Return position of inserted node
};
```

➔ **This code is unchecked. EC credit to the 1st student finding a specific bug**

Max Heap Removal (pop)

(Root removed / Replaced by last element / "percolated")



Always swap with larger child

Extra Credit Homework Assignment #4

- **Fairly Minimum EC: Code MaxHeap “look” in C++**
- **Maximum EC: Write MaxHeap “pop” in C++**