5.A. (Binary) Heap Properties

1. Binary tree.

2. Like a complete tree, but missing some “rightmost” leaves in deepest level.

3. a. Each parent has a priority $\geq$ the priority of its children - maxheap.

   b. Each parent has a priority $\leq$ the priority of its children - minheap.

Common Mapping of a Heap to an Array

1. Subscript 0 is unused.

2. Subscript 1 stores the root of the heap.

3. If it exists, the left child for subscript $i$ is stored at subscript $2i$.

4. If it exists, the right child for subscript $i$ is stored at subscript $2i+1$.

5. Parent of node with subscript $i$ is $\left\lfloor \frac{i}{2} \right\rfloor$. 
Aside: Alternate Mapping of a Heap to an Array

1. Subscript 0 stores the root of the heap.
2. If it exists, the left child for subscript \( i \) is stored at subscript \( 2i + 1 \).
3. If it exists, the right child for subscript \( i \) is stored at subscript \( 2i + 2 \).
4. Parent of node with subscript \( i \) is \( \left\lfloor \frac{i-1}{2} \right\rfloor \).

Under either mapping, the number of nodes/slots used \( (n) \) must be stored.

*No pointers are needed.*

5.B. CONVERTING AN UNORDERED ARRAY INTO A MAXHEAP (BOTTOM-UP HEAP CONSTRUCTION)

Special Case: Parent Node with Two Subtrees Having Maxheap Properties
(maxHeapify/fixDown/sink)

Worst case, \( 2^k \leq n \leq 2^{k+1} - 1 \), processes _________ parents.
void maxHeapify(Item a[], int k, int N)
{
    int j;

    while (left(k) <= N)
    {
        j = left(k);  // left child
        if (j < N && less(a[j], a[j+1]))
            j = right(k);  // use right child instead
        if (!less(a[k], a[j]))
            break;
        exch(a[k], a[j]); // Swap child with parent
        k = j;            // Descend
    }
}

General Case: Move through the parents in descending subscript order, applying the special case at each parent.

Time

Lower bound?

Worst case?

Worst case time

Each level may receive values from all ancestors in shallower levels.

5.C. SORTING USING A HEAP (http://ranger.uta.edu/~weems/NOTES2320/heapsort.c)

Bottom-up heap construction leaves the maximum at the root.

Swap root with element at subscript n. Remove subscript n from heap.

maxHeapify/fixDown/sink at root to restore maxheap property.

...
Time

Bottom-up heap construction takes $\Theta(n)$ for worst case.

Even though the heap loses one node after each call to \texttt{maxHeapify},
1/2 the nodes are leaves taking $\Theta(\log n)$ worst-case time for each.

$\Theta(n \log n)$ for entire sort in worst case.

Other

Similar to selection sort, but heapsort repeatedly finds maximum instead of minimum

Heapsort is optimal, but involves larger constants than other optimal sorts

Stable?

5.D. PRIORITY QUEUES

Entry with the maximum priority is the next item to be removed.

Operations are naturally supported by a heap in $\Theta(\log n)$ worst-case time.

At least the following two operations are required to support a PQ:

\texttt{maxHeapInsert} – New item is placed at next available leaf and is swapped with ancestors
having lower priorities (\texttt{fixUp/swim}).

\texttt{heapExtractMax} – Remove item with maximum priority and fix heap, just like \texttt{HEAPSORT}.

(\texttt{http://ranger.uta.edu/~weems/NOTES2320/intPQi.c})

\begin{verbatim}
int less(int i, int j)
{
  // Notice how heap entries reference a[]
  return a[pq[i]] < a[pq[j]];
}

void heapIncreaseKey(int *pq,int k) // AKA swim
{
  while (k > 1 && less(parent(k), k))
  {
    exch(k, parent(k));
    k = parent(k);
  }
}
\end{verbatim}
void maxHeapify(int *pq, int k, int N) // AKA sink
{
  int j;

  while (left(k) <= N)
  {
    j = left(k);
    if (j < N && less(j, j+1))
      j = right(k);
    if (!less(k, j))
      break;
    exch(k, j);
    k = j;
  }
}

Other convenient PQ operations, based on an “external” dictionary:

  Dictionaries – Topic covered in detail for later exams.

  Supports finding a desired (key, satellite data) pair

Some possibilities

  Table (see http://ranger.uta.edu/~weems/NOTES2320/intPQi.c for a subscript-as-key table as dictionary)

  Linked List

  (Binary) Search Tree

  Hash Table

Linking Between Dictionary and Heap (“Client Arrays”)

  Handles (qp) are used to track movement of heap entries.

  Satellite data in each dictionary (e.g. BST) entry includes subscript of a handle.

  Priority may be stored in heap entry, handle entry, or in the dictionary (as in intPQi.c).

  Invariants for handles that are in use: pq[qp[i]]==i and qp[pq[i]]==i

  Each heap entry includes the number of its handle.

  [An efficient scheme for allocating handles is considered in Notes 9.]
void exch(int i, int j)
{
  // Swaps parent with child in heap
  int t;
  t = pq[i];
  pq[i] = pq[j];
  pq[j] = t;
  qp[pq[i]] = i;
  qp[pq[j]] = j;
}

Change Priority - maxHeapChange

  maxHeapDecreaseKey (maxHeapify/fixDown/sink)
  maxHeapIncreaseKey (fixUp/swim)

Delete Entry – maxHeapDelete

  Replace with contents of highest-numbered leaf, then fix ordering.