Storing binary trees as arrays
Heaps (Max-Heap)

**HEAP** represents a binary tree stored as an array such that:
- Tree is filled on all levels except last
- Last level is filled from left to right
- Left & right child of \( i \) are in locations \( 2i \) and \( 2i+1 \)
- **HEAP PROPERTY:**
  Parent value is at least as large as child’s value
HeapSort

• First convert array into a heap (**BUILD-MAX-HEAP**, p133)
• Then convert heap into sorted array (**HEAPSORT**, p136)
Max-Heapify(array a, integer i)

\[ l = \text{left}(i) \]
\[ r = \text{right}(i) \]
\[ \text{if } ((l \leq \text{size}(a)) \&\& (a[l] > a[i])) \text{ then} \]
\[ \text{largest} = l \]
\[ \text{else largest} = i \]
\[ \text{if } ((r \leq \text{size}(a)) \&\& (a[r] > a[\text{largest}]))) \text{ then} \]
\[ \text{largest} = r \]
\[ \text{if largest} \neq i \text{ then} \]
\[ \text{swap}(a[i], a[\text{largest}]) \]
\[ \text{Max-Heapify}(a, \text{largest}) \]
Build-Max-Heap(array a)

size[a] = length[a];

for i = \[ \lfloor \text{length}[a]/2 \rfloor \] downto 1 do
  Max-Heapify(a,i)
HeapSort(array a)

Build-Max-Heap(a);
for i = length(a) downto 2 do
    swap(a[1], a[i]);
    size[a] --;
    Max-Heapify(a, 1);

Total: O(n log n)
For the HeapSort analysis, we need to compute:

\[ \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \]

We know from the formula for geometric series that

\[ \sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} \]

Differentiating both sides, we get

\[ \sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1 - x)^2} \]

Multiplying both sides by \( x \) we get

\[ \sum_{k=0}^{\infty} kx^k = \frac{x}{(1 - x)^2} \]

Now replace \( x = 1/2 \) to show that

\[ \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \leq \frac{1}{2} \]
## Sorting Algorithms

- **Number of Comparisons**
- **Number of Data Movements**
- **Additional Space Requirements**
## Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Shaker Sort

- Merge Sort
- Heap Sort
- Quick Sort

- Bucket & Radix Sort
- Counting Sort
Animation Demos

http://www-cse.uta.edu/~holder/courses/cse2320/lectures/applets/sort1/heapsort.html

http://cg.scs.carleton.ca/~morin/misc/sortalg/
Bucket Sort

- N values in the range \([a..a+m-1]\)
- For e.g., sort a list of 50 scores in the range \([0..9]\).

**Algorithm**
- Make \(m\) buckets \([a..a+m-1]\)
- As you read elements throw into appropriate bucket
- Output contents of buckets \([0..m]\) in that order

- Time \(O(N+m)\)
Stable Sort

- A sort is **stable** if equal elements appear in the same order in both the input and the output.
- Which sorts are stable? Homework!
Radix Sort

Algorithm
for i = 1 to d do
    sort array A on digit i using any sorting algorithm

Time Complexity: $O((N+m) + (N+m^2) + \ldots + (N+m^d))$

Space Complexity: $O(m^d)$
Radix Sort

Algorithm

for i = 1 to d do

    sort array A on digit i using a stable sort algorithm

Time Complexity: $O((n+m)d)$
## Counting Sort

### Initial Array

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

### Counts

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Cumulative Counts

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>