Outline

1. Priority Queue
2. Heap
3. Heap Sort
Priority Queue is a data structure maintaining an array of elements, each element has an index and a key value.

One only can remove the first element and insert a new element to the end.
Priority Queue

- **Extract:** remove the first element.
- **Insert:** Add a new element to the end.

Index: 1  2  3  4  5  6  7

key value: 2  1  9  7  9  0  5
Priority Queue

- **Extract**: remove the first element.
- **Insert**: Add a new element to the end.

Index: 1 2 3 4 5 6 7

**key value**: 2 1 9 7 9 0 5
Priority Queue

- **Extract**: remove the first element.
- **Insert**: Add a new element to the end.

<table>
<thead>
<tr>
<th>Index</th>
<th>Key Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>
**Priority Queue**

- **Extract**: remove the first element.
- **Insert**: Add a new element to the end.

<table>
<thead>
<tr>
<th>Index</th>
<th>Key Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
1. Priority Queue

2. Heap

3. Heap Sort
Heap is a special priority queue:


Heap can be visualized by a nearly complete binary tree.
Build A Heap

**BuildMaxHeap(A)**

1: **For** \( i = n/2 \) to 1  
2: \hspace{1em} MaxHeapify(A,i)  
3: **End For**

**MaxHeapify(A, i)**

1: if \( A[i] \geq \max\{A[2i], A[2i + 1]\} \) then  
2: \hspace{1em} Do nothing  
3: else  
4: \hspace{2em} if \( A[2i] = \max\{A[2i], A[2i + 1]\} \) then  
5: \hspace{3em} Swap the key values of \( A[i] \) and \( A[2i] \)  
6: \hspace{3em} MaxHeapify(A, 2i)  
7: \hspace{1em} else  
8: \hspace{2em} Swap the key values of \( A[i] \) and \( A[2i + 1] \)  
9: \hspace{2em} MaxHeapify(A, 2i+1)  
10: **end if**  
11: **end if**
**Build A Heap: Complexity**

- **MaxHeapify(A, i):** $O(\text{height}(A[i])) = O(\log n)$, $\text{height}(A[i])$ is the height of $A[i]$ in the corresponding visualizing tree.

- **BuildMaxHeap(A):** $O(n \log n)$, but actually is $\Theta(n)$ by a more careful analysis.
Outline

1. Priority Queue
2. Heap
3. Heap Sort
Heap Sort

- **Basic idea:**
  1. Build a MaxHeap.
  2. Repeat the following steps $n$ times:
     1. Swap the key values of the first and final elements.
     2. Reduce the size of the heap by one.
     3. Run MaxHeapify(A, 1).

- **Complexity:** $\Theta(n \log n)$ time and $\Theta(1)$ space.
Comparison of Three Sorting Algorithms

- **Insertion Sort:** $\Theta(n^2)$ time in worst case and $\Theta(n)$ time in best case, $\Theta(1)$ extra space.

- **Merge Sort:** $\Theta(n \log n)$ time and $\Theta(n)$ extra space; easy to be parallel.

- **Heap Sort:** $\Theta(n \log n)$ time and $\Theta(1)$ extra space.