Priority Queues

- Priority Queues are an example of an application of heaps.
- A priority queue is a data structure for maintaining a set of elements \( S \), each with an associated key and a corresponding handle (typically, the corresponding array index).
- Uses include:
  - Job scheduling (OS)
  - Bandwidth prioritization
  - Dijkstra’s Algorithm (shortest path algorithm)
  - Prim’s algorithm (minimum spanning tree)
  - Huffman coding (error detection, lossless data compression)
Priority Queue: Example

<table>
<thead>
<tr>
<th>Elements (Set S)</th>
<th>Key (Priority Level)</th>
<th>Handle (Index)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish HW 3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Eat</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Drink Coffee</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Sleep</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Plan for Spring Break</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

We assume throughout this lecture that the associated underlying heap satisfies the **max-heap property** unless otherwise noted.

Priority Queues: Generic

- **Max-priority queue (ADT)** supports dynamic set operations:
  - `MAXIMUM(S)`: returns element of S with largest key.
  - `INSERT(S, x)`: inserts element x into set S.
  - `EXTRACT-MAX(S)`: removes and returns element S with largest key.
  - `INCREASE-KEY(S, x, k)`: increases value of element x’s key to k. Assume k \(\geq\) x’s current key value.

Analogous functions for Min-priority queues
Priority Queues

- Generally, algorithmically discuss the Heap (implemented as an array A) directly:
  - **HEAP-MAXIMUM**(A) : returns element of A with largest key.
  - **MAX-HEAP-INSERT**(A, key) : inserts element key into A.
  - **HEAP-EXTRACT-MAX**(A) : removes and returns element A with largest key.
  - **HEAP-INCREASE-KEY**(A, i, key) : increases value of element i’s key A[i] to key. Assume key >= i’s current key value of A[i].

**HEAP-MAXIMUM:**

```plaintext
HEAP-MAXIMUM (A)

1 return A[1]
```

Time complexity?
HEAP-EXTRACT-MAX (Motivation)

Given the array A:
- Make a copy of the maximum element.
- Make the last node in the tree the new root.
- Re-heapify the heap, with one fewer node.
- Push the new element at root down the tree to it's appropriate position
- Return the copy of the maximum element.

HEAP-EXTRACT-MAX (Implementation)

HEAP-EXTRACT-MAX (A)

1 if A.heap-size < 1
2   error "heap underflow"
3 max = A[1]
5 A.heap-size = A.heap-size - 1
6 MAX-HEAPIFY(A,1)
7 return max

Time Complexity?
Example: HEAP-EXTRACT-MAX(A)

• 15 8 5 6 7 4 3 1 2

HEAP-INCREMENT-KEY (Motivation)

Given A, element i, and new key value key:
• Make sure \text{key} \geq i's current key A[i].
• Update i's key value to key.
• Traverse the tree upward comparing i’s key to its parent’s key and swapping keys if necessary, until i’s key is smaller than its parent’s key.
HEAP-INCREASE-KEY

HEAP-INCREASE-KEY(A,i,key)

1  if key < A[i]
2      error "new key smaller than current key"
3    A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5    exchange A[i] with A[PARENT(i)]
6    i = PARENT(i)

Time Complexity?

Example: HEAP-INCREASE-KEY

- Increase key of node 6 in previous example to 20
MAX-HEAP-INSERT

- Given a key to insert into the heap:
  - Insert a new node in the very last position in the tree with the key $-\infty$
  - Increase the $-\infty$ key to key using the HEAP-INCREASE-KEY procedure.

MAX-HEAP-INSERT(A, key)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$A$.heap-size = $A$.heap-size + 1</td>
</tr>
<tr>
<td>2</td>
<td>$A[A$.heap-size$] = -\infty$</td>
</tr>
<tr>
<td>3</td>
<td>HEAP-INCREASE-KEY($A$, $A$.heap-size, key)</td>
</tr>
</tbody>
</table>

Time Complexity?

Example

- Insert 12 into the previous heap.