Lecture 29:

- **Building** a Binary heap on $n$ elements in $O(n)$ time.
- **Applications of** Binary heap: sorting
- **Binary trees**: beyond searching and sorting
Recap from the last lecture
Theorem: A complete binary tree can be implemented by an array.
A complete binary tree
A complete binary tree

How many leaves are there in a complete Binary tree of size $n$?

No. of Leaf nodes = No. of Internal nodes + 1
How many leaves are there in a Complete Binary tree of size $n$?

No. of Leaf nodes = No. of Internal nodes
Binary heap

A complete binary tree satisfying heap property at each node.
### Binary heap

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Find-min}(H)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$\text{Insert}(x, H)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$\text{Extract-min}(H)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$\text{Decrease-key}(p, \Delta, H)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$\text{Merge}(H_1, H_2)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Building a Binary heap
Building a Binary heap

Problem: Given \( n \) elements \( \{x_0, \ldots, x_{n-1}\} \), build a binary heap \( H \) storing them.

Trivial solution:

(Building the Binary heap incrementally)

\[
\begin{align*}
&\text{CreateHeap}(H); \\
&\text{For( } i = 0 \text{ to } n - 1 \text{ )} \\
&\quad \text{Insert}(x_i, H);
\end{align*}
\]

What is the time complexity of this algorithm?
Building a Binary heap incrementally

The time complexity for inserting a leaf node = $O(\log n)$

# leaf nodes = $\lfloor n/2 \rfloor$,

⇒ **Theorem**: Time complexity of building a binary heap incrementally is $O(n \log n)$. 

What useful inference can you draw from this **Theorem**?

Top-down approach
Building a **Binary heap incrementally**

The $O(n)$ time algorithm must take $O(1)$ time for each of the $\lceil n/2 \rceil$ leaves.

What useful inference can you draw from this **Theorem**?
Think of alternate approach for building a binary heap

In any complete binary tree, how many nodes satisfy heap property?

all leaf nodes

Bottom-up approach

heap property: “Every node stores value smaller than its children” We just need to ensure this property at each node.

Does it suggest a new approach to build binary heap?
A new approach to build binary heap

1. Just copy the given \( n \) elements \( \{x_0, \ldots, x_{n-1}\} \) into an array \( H \).

2. The **heap property** holds for all the leaf nodes in the corresponding complete binary tree.

3. Leaving all the leaf nodes, process the elements in the decreasing order of their numbering and set the heap property for each of them.
new approach to build binary heap

The first node to be processed.
new approach to build binary heap

The second node to be processed.
new approach to build binary heap

The third node to be processed.
new approach to build binary heap
new approach to build binary heap

Let $v$ be a node corresponding to index $i$ in $H$. The process of restoring heap property at $i$ called $\text{Heapify}(i,H)$. 
Heapify($i, H$)

Heapify($i, H$)

\[
\begin{align*}
  &n \leftarrow \text{size}(H) - 1; \\
  &\text{Flag} \leftarrow \text{true}; \\
  &\text{While}( \quad i \leq \left\lfloor (n-1)/2 \right\rfloor \quad \text{and} \quad \text{Flag} = \text{true} \quad ) \\
  &\quad \{ \\
  &\quad \quad \text{\textit{min} } \leftarrow i; \\
  &\quad \quad \text{If}(H[i] > H[2i + 1]) \quad \text{\textit{min} } \leftarrow 2i + 1; \\
  &\quad \quad \text{If}(\text{it is smaller than any of its children} \quad \Rightarrow \text{Swap it with } \text{smallest} \text{ child} \quad \text{and move down} \ldots \\
  &\quad \quad \text{\textit{H}(i) } \leftrightarrow \text{\textit{H}(\textit{min})}; \\
  &\quad \quad \text{\textit{i} } \leftrightarrow \text{\textit{min}}; \\
  &\quad \quad \text{Else stop!} \\
  &\quad \quad \text{else} \\
  &\quad \quad \text{\textit{Flag} } \leftarrow \text{false}; \\
  &\quad \} \\
  &\} \\
\end{align*}
\]
Building Binary heap in $O(n)$ time

How many nodes of height $h$ can there be in a complete Binary tree of $n$ nodes?

Time to heapify node $v$?

Height($v$)

Time complexity of algorithm = $\sum_h O(h) \cdot N(h)$

No. of nodes of height $h$
A complete binary tree

How many nodes of height \( h \) can there be in a complete Binary tree of \( n \) nodes?

Each subtree is also a complete binary tree.

\[ \Rightarrow \text{A subtree of height } h \text{ has at least } 2^h \text{ nodes} \]

Moreover, no two subtrees of height \( h \) in the given tree have any element in common.

Hence the number of nodes of height \( h \) is bounded by \( \frac{n}{2^h} \)
Building Binary heap in \( O(n) \) time

**Lemma:** the number of nodes of height \( h \) is bounded by \( \frac{n}{2^h} \).

Hence Time complexity to build the heap \( = \sum_{h=1}^{\log n} \frac{n}{2^h} O(h) \)

\[
= n \cdot c \cdot \sum_{i=1}^{\log n} \frac{h}{2^h}
\]

\( = O(n) \)

As an exercise (using knowledge from your JEE preparation days), show that \( \sum_{h=1}^{\log n} \frac{h}{2^h} \) is bounded by 2
Sorting using a Binary heap
Sorting using heap

Build heap $H$ on the given $n$ elements;

While ($H$ is not empty)

{} $x \leftarrow$ Extract-min($H$);
    print $x$;

}  

This is **HEAP SORT** algorithm  

**Time complexity**: $O(n \log n)$

**Question**: Which is the best sorting algorithm: (Merge sort, Heap sort, Quick sort) ?

**Answer**: Next programming assignment 😊
Binary trees: beyond searching and sorting

• Elegant solution for two interesting problem

• An important lesson:

  Lack of proper understanding of a problem is a big hurdle to solve the problem
Two interesting problems on sequences
What is a sequence?

A sequence $S = \langle x_0, \ldots, x_{n-1} \rangle$

- Can be viewed as a mapping from $[0, n]$.
- Order does matter.
Problem 1

Multi-increment
Problem 1

Given an initial sequence $S = < x_0, \ldots, x_{n-1} >$ of numbers, maintain a compact data structure to perform the following operations:

- **ReportElement($i$):**
  Report the current value of $x_i$.

- **Multi-Increment($i$, $j$, $\Delta$):**
  Add $\Delta$ to each $x_k$ for each $i \leq k \leq j$

**Example:**
Let the initial sequence be $S = < 14, 12, 23, 12, 111, 51, 321, -40 >$

After **Multi-Increment(2,6,10)**, $S$ becomes
$$ < 14, 12, 33, 22, 121, 61, 331, -40 > $$

After **Multi-Increment(0,4,25)**, $S$ becomes
$$ < 39, 37, 58, 47, 146, 61, 331, -40 > $$

After **Multi-Increment(2,5,31)**, $S$ becomes
$$ < 39, 37, 89, 78, 177, 92, 331, -40 > $$
Problem 1

Given an initial sequence \( S = < x_0, ..., x_{n-1} > \) of numbers, maintain a compact data structure to perform the following operations:

- **ReportElement\( (i) \):**
  
  Report the current value of \( x_i \).

- **Multi-Increment\( (i, j, \Delta) \):**
  
  Add \( \Delta \) to each \( x_k \) for each \( i \leq k \leq j \)

**Trivial solution:**

Store \( S \) in an array \( A[0..n-1] \) such that \( A[i] \) stores the current value of \( x_i \).

\[
\text{Multi-Increment}\( (i, j, \Delta) \) \\
\{ \\
\text{For } (i \leq k \leq j) \quad A[k] \leftarrow A[k] + \Delta; \\
\} \\
\text{ReportElement}\( (i) \)\{ return A[i] \} \\
\]

\[
O(j - i) = O(n) \quad \text{and} \quad o(1)
\]
Problem 1

Given an initial sequence \( S = < x_0, ..., x_{n-1} > \) of numbers, maintain a compact data structure to perform the following operations:

- **ReportElement\( (i) \):**
  Report the current value of \( x_i \).

- **Multi-Increment\( (i, j, \Delta) \):**
  Add \( \Delta \) to each \( x_k \) for each \( i \leq k \leq j \)

---

**Trivial solution:**
Store \( S \) in an array \( A[0..n-1] \) such that \( A[i] \) stores the current value of \( x_i \).

**Question:** the source of difficulty in breaking the \( O(n) \) barrier for Multi-Increment()?

**Answer:** we need to explicitly maintain in \( S \).

**Question:** who asked/inspired us to maintain \( S \) explicitly.

**Answer:** 1. incomplete understanding of the problem  
2. conditioning based on incomplete understanding
Towards efficient solution of Problem 1

**Assumption:** without loss of generality assume $n$ is power of $2$.

Explore ways to maintain sequence $S$ *implicitly* such that

- **Multi-Increment**($i, j, \Delta$) is efficient
- **Report**($i$) is efficient too.

**Main hurdle:** To perform **Multi-Increment**($i, j, \Delta$) efficiently
Problem 2

Dynamic Range-minima
Problem 2

Given an initial sequence \( S = < x_0, ..., x_{n-1} > \) of numbers, maintain a compact data structure to perform the following operations efficiently for any \( 0 \leq i < j < n \).

- **ReportMin(\( i, j \))**: Report the minimum element from \( \{ x_k \mid \text{for each } i \leq k \leq j \} \)
- **Update(\( i, a \))**: \( a \) becomes the new value of \( x_i \).

**AIM:**

- \( O(n) \) size data structure.
- **ReportMin(\( i, j \))** in \( O(\log n) \) time.
- **Update(\( i, a \))** in \( O(\log n) \) time.

All data structure lovers must ponder over these two problems 😊. We shall discuss them in the next class.