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
A **complete** binary tree

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

$\left\lceil \frac{n}{2} \right\rceil$
Building a Binary heap

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

Trivial solution:

(Building the Binary heap incrementally)

CreateHeap($H$);
For($i = 0$ to $n - 1$)
Insert($x_i, H$);

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 = \( \lceil n/2 \rceil \),

\[ \Rightarrow \textbf{Theorem}: \text{Time complexity of building a binary heap incrementally is } O(n \log n). \]
Building a Binary heap incrementally

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

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

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

Top-down approach
Think of alternate approach for building a binary heap

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

heap property: “Every node stores value smaller than its children”

We just need to ensure this property at each node.
Think of alternate approach for building a binary heap

heap property: “Every node stores value smaller than its children”

We just need to ensure this property at each node.
A new approach to build binary heap

1. Just copy the given $n$ elements $\{x_0, ..., x_{n-1}\}$ 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.
A new approach to build binary heap

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

The third node to be processed.
A new approach to build binary heap
A new approach to build binary heap
A new approach to build binary heap
A new approach to build binary heap
A 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 \texttt{Heapify}(i,H).
Heapify($i, H$)

Heapify($i, H$)
{
    $n \leftarrow \text{size}(H) - 1$ ;

    While ( ? and ? )
    {
        For node $i$, compare its value with those of its children
        If it is smaller than any of its children  ➔ Swap it with smallest child and move down ... 
        Else stop !
    }
}
Heapify($i$, $H$)

{  
  $n \leftarrow \text{size}(H) - 1$ ;
  
  Flag $\leftarrow$ true;
  
  While ( $i \leq \lfloor (n-1)/2 \rfloor$ and Flag = true )
  
  {  
    min $\leftarrow i;$
    
    If( $H[i] > H[2i + 1]$ ) min $\leftarrow 2i + 1;$
    
    If( $2i + 2 \leq n$ and $H[min] > H[2i + 2]$ ) min $\leftarrow 2i + 2;$
    
    If( min $\neq i$ )
      
      {  
        H(i) $\leftrightarrow$ H(min);
        
        i $\leftarrow$ min;
      }

    else
      
      Flag $\leftarrow$ false;

  }

}
Building Binary heap in $O(n)$ time

Time complexity of algorithm = \[ \sum_{h} O(h) \cdot N(h) \]

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

Time to heapify node $v$?
Each subtree is also a complete binary tree.

A subtree of height $h$ has at least $2^h$ nodes.

Moreover, no two subtrees of height $h$ in the given tree have any element in common.
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 = $\sum_{h=1}^{\log n} \frac{n}{2^h} \cdot O(h)$

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

$$= O(n)$$

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;

$\textbf{While (} H \textbf{ is not empty)}$

$\{$

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

print $x$;

$\} \hspace{1cm}$

This is $\textbf{HEAP \ SORT}$ algorithm

$\textbf{Time complexity : } O(n \ \log n)$

$\textbf{Question:}$

Which is the best sorting algorithm : $\textbf{(Merge sort, Heap sort, Quick sort)}$ ?

$\textbf{Answer:}$ Practice 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 $S = < x_0, ..., x_{n-1} >$

- 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 \( S = \langle x_0, \ldots, x_{n-1} \rangle \) 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 \( x_i \).

**Multi-Increment**\( (i, j, \Delta) \)

\[
\begin{array}{l}
\text{For } (i \leq k \leq j) \quad A[k] \leftarrow A[k] + \Delta; \\
\end{array}
\]

\( O(j - i) = O(n) \)

**ReportElement**\( (i) \)

\[
\begin{array}{l}
\text{return } A[i]; \\
\end{array}
\]

\( O(1) \)
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.