Vector to represent a heap

- Store the root as the first element of the array.
- If a node is stored at index i of the array its left child is at index $2i$ and right child at index $2i + 1$
- Equvalently, the parent of a node at index i can be found at index $\lfloor i/2 \rfloor$.

Heap example: insertion

- Insert the new node at leftmost vacant position in the bottomost level.
- Bubble up the value at the new node to maintain the heap property.
  - if the value at parent it greater then new value swap the values.
  - repeat till value is either bubbled up all the way to root or the parent value is less than the value being bubbled up.

Heap example: deletion

- The minimum element is stored at root.
- Deletion of minimum element means heap will fall apart.
- So to delete from a heap: connectedness and heap property must be preserved.
- Rightmost leaf node at bottom-most level replaces the value stored at root and the former node is deleted. This preserves the connectedness property.
- To maintain heap property sink the new element at root down to its proper position as follows:
  - compare the element with the values stored at children and swap it with the value in parent node, and
  - repeat the above step until no swap is needed.

Analysis of heap building

- Operations for heapifying takes $O(\log n)$ steps.
- Building a heap requires heapify as many times as nodes are inserted starting with an empty heap.
- The maximum number of nodes at the bottom-most level (at height 0) is $n/2$. Note that these are roots of heaps of height 0.
- The maximum number of nodes at level one up is $n/4$. These are precisely the roots of heaps of height 1 each.
- In general at level i there are at most $\lceil \frac{n}{2^{i+1}} \rceil$ nodes and these are the roots of the heaps of height i each.

Analysis of building a heap

- Another way to look at the problem of constructing heap is to think of a recursive approach.
- Partition the given set of n numbers into two subsets of size $(n-1)/2$ each.
- Construct the heap of each subset recursively.
- Define the remaining one number as root of overall heap by joining the two heaps of size $(n-1)/2$ as left and right sub-heaps.
- Then sink-down, if required, the root to preserve the heap property.

Analysis of building a heap

- Therefore the time for building a heap of n elements from an intially empty heap requires fixing the roots of sub heaps of heights i, where i varies from 0 to h.
- Since heaps of height 0 require 0 heapify operation, the time for creating heap of height h is given by

$$T(n) = \sum_{i=1}^{h} i \frac{n}{2^{i+1}}$$

Analysis of heap building

To find a bound for

$$T_n = \sum_{i=1}^{h} \frac{i}{2^{i+1}}$$

Consider

$$\begin{aligned}
S_n &= \textstyle\sum_{i=0}^{n} \frac{i}{2^i} \\
2S_n &= 2\textstyle\sum_{i=0}^{n} \frac{i}{2^i} \\
&= \textstyle\sum_{i=0}^{n} \frac{i}{2^{i-1}} \\
&= \textstyle\sum_{j=-1}^{n-1} \frac{j+1}{2^j} \\
&= \textstyle\sum_{j=0}^{n} \frac{j+1}{2^j} - \frac{n+1}{2^n}
\end{aligned}$$

Analysis of heap construction

- This implies

$$
\begin{aligned}
2S_n - S_n &= \sum_{j=0}^{n} \frac{j+1}{2^j} - \frac{n+1}{2^n} - \sum_{i=0}^{n} \frac{i}{2^i} \\
&= \sum_{j=0}^{n} \frac{1}{2^i} - \frac{n+1}{2^n}
\end{aligned}
$$

- Therefore

$$S_n = 2 - \frac{1}{2^n} - \frac{n}{2^n}$$

- Implying $\lim_{n \to \infty} S_n = 2$.
- As $T_n \leq S_n$, the value of T_n cannot exceed 2.
- Therefore the time $T(n) \leq n \times T_n = O(n)$ to build a heap of n elements.