Stacks.

**Push (obj)**

\[
A[s] \leftarrow \text{obj} \\
S \leftarrow S + 1 \\
\text{if } S = a \text{ then} \\
A \leftarrow A \times 2 \\
\text{Copy elements in the new array.}
\]

Consider a set of n Push operations.

- Worst case: cost \( n \) on \( \text{push} = n \).
- \# times this happen \( \leq \log n \).
- \( \text{Total Cost: } O(n \log n) \).
Total cost = Cost to push + Cost to copy + Cost to allocate.

\[
\frac{1111}{2^1 + 2^2 + 2^3 + 2^4} = n + \left(1 + 2 + 2^2 + \cdots + 2^{\frac{\ln n}{\ln 2}}\right) + \\
2^1 - 1
\]

\[
= n + \Theta(n^2) \times X
\]

\[
= n + 2^{-1} + 2^{-1}
\]

\[
= n + 2n + 4n - 2
\]

\[
\approx \frac{7n}{n} = \Theta(n)
\]

Amortized cost of one push ≤ \( \frac{7n}{n} = 7 = \Theta(1) \).
Account Method:

$1: push
$1: copy
$2: allocate array

$1: push
$2: own copy + copy & friend
$4: 4 slots in the new array

$7 in total.

$1: own push
$2: own copy + final copy
$4: allocate

Total contribution of all elements (can be all the costs) \( \Rightarrow T_n = \Theta(cn) \).

Pop( )

when the # elements becomes \( \frac{a}{y} \) then we reduce the array size by half.
when the # elements in the array < \( \frac{a}{2} \),
reduce the size of the array by half.

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
\hline
\end{array}
\]

\[
\text{\textbackslash pop.}
\]

Queues:

Enqueue, Dequeue \quad \text{FIFO}

add at the end \quad \text{to remove from front of the queue.}

\text{in the queue.}
Heaps:

A Heap is an array object that can be viewed as an almost complete binary tree.

except possibly
binary tree that is full at all levels but the last.
The last level is full until the last element, i.e., there are no "heli" at the last level.

heapsite : # elements in the heap.

Max heaps
Min heaps

arraysite : array capacity.

Heap property (Maxheap) : For any index i,

\[ A[\text{parent}(i)] \geq A[i] \]

For minheap it is

\[ A[\text{parent}(i)] \leq A[i] \]

\[ \left\lceil \frac{7}{2} \right\rceil = 4 \]

\[ A: \begin{array}{ccccccccc}
59 & 39 & 28 & 19 & 30 & 27 & 21 & 11 & 8 \\
\end{array} \]

\[ \text{parent}(i) : \left\lceil \frac{i}{2} \right\rceil \]

\[ \text{left}(i) : 2i \]

\[ \text{right}(i) : 2i + 1 \]

\[ \text{Maxheap} \]

\[ \text{Minheap} \]
Height of heap: let \( h \) be the height of the heap.

\[
h = \lceil \log n \rceil = \Theta(\log n).
\]

Let \( n \) be the number of elements in the heap.

Note that

\[
n \geq 2^0 + 2^1 + \cdots + 2^{h-1} + 1
\]

\[
= \frac{2^h - 1}{2 - 1} + 1
\]

\[
= 2^h - 1 + 1
\]

\[
= 2^h
\]

\[
2^h \leq n
\]

\[
n \leq \text{all nodes in a complete bin tree of height } h
\]
\[= 2^0 + 2^1 + \ldots + 2^h\]
\[= 2^{h+1} - 1\]
\[< 2^{h+1}\]
\[n < 2^{h+1}\]
\[\therefore 2^h < n < 2^{h+1}\]

Take logs
\[h \leq \log n < h+1\]
\[\therefore h = \lfloor \log n \rfloor = \Theta(\log n)\]

MaxHeapify (A, i) \leftarrow fixes the heap property at the tree rooted at i

// prerequisites: tree rooted at left (i) is a max heap
// root of tree at node i is a max heap.
\[
\begin{align*}
l &\leftarrow \text{left}(i) \\
r &\leftarrow \text{right}(i) \\
\begin{cases}
\text{if } l \leq \text{heapsize} \text{ and } A[l] > A[i] \text{ then} \\
\text{largest} &\leftarrow l \\
\text{else} & \text{largest} &\leftarrow i \\
\text{if } r \leq \text{heapsize} \text{ and } A[r] \geq A[\text{largest}] \text{ then} \\
\text{largest} &\leftarrow r
\end{cases}
\end{align*}
\]
if \( \text{largest} \neq i \) then

\[
\text{Swap } A[i] \text{ and } A[\text{largest}]
\]

\text{MaxHeapify}(A, \text{largest})

\[T(n) = T\left(\frac{n}{2}\right) + \Theta(1)\]

\[a = 1, \quad b = \frac{3}{2}, \quad k = 0\]

Case II: \( T(n) = \Theta\left(a \cdot b^k \cdot n\right) \)
\[ \Theta \left( \log^{\frac{1}{n}} \right) = \Theta \left( \log n \right). \]

**BuildHeap**

\[
A: \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 9 & 7 & 8 & 9 & 10 \\
19 & 25 & 36 & 12 & 1 & 86 & 79 & 31 & 46 & 51
\end{array}
\]

heapSize \leftarrow |A|

\[
\text{for } i \leftarrow |A| \text{ down to } 1 \text{ do }
\]
Maxheapify (i) 

Runtimes: \( T(n) = \Theta (n \log n) \).

Lemma: There are \( \sum_{h=1}^{\log n} \frac{n}{2^{h+1}} \) nodes at height \( h \) in a complete binary tree.

Runtime: \( O \left( \sum_{h=1}^{\log n} \frac{n}{2^{h+1}} \cdot h \right) \)

\[= O \left( n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) \left[ \sum_{i=0}^{\infty} \frac{c^i}{(1-c)^2} \right] \]

\[= O \left( n \frac{1/2}{(1-1/2)^2} \right) \]

\[= O(n). \]
Running time: $\Theta(n)$