Simplified [Kleene '52, Zuckerman]

\[
\begin{align*}
\text{push-heap} & \quad O(1) \\
\text{insert} & \quad O(1) \\
\text{extract-min} & \quad \begin{cases} O(1) & \text{randomized} \\ O(\log n) & \text{deterministic} \end{cases}
\end{align*}
\]

"error parameter" $\varepsilon \leq 1$

\begin{itemize}
\item meld
\item delete
\end{itemize}

$\varepsilon > 0$ arbitrary constant parameter

Comparison model (keys only accessed via comparisons)

- e.g. $\varepsilon = 0.1 = \Rightarrow \text{extract-min} = O(1)$
- The one of min/extract-min must have cost $\geq \Omega(n)$

Why is this not a contradiction?

Answer:

Soft heap "corrupts" an $\varepsilon$-fraction of the keys.

We have no control over the corruption.

If $\varepsilon < \frac{1}{m}$, cost of operations $O(\log m)$.
If $\varepsilon = 1$, not covered in this class.

Q: What is soft-heap good for?

1. How to implement it?
   - a) [Kleene] Deterministic MST $O(m \alpha(m))$
   - b) Known deterministic MST algorithm
   - c) Randomized $O(m) \alpha(m)$ expected time

\begin{itemize}
\item $\alpha(n)$ extremely slowly growing: almost optimal
\item $\alpha(5) = 1$
\item $\alpha(10) = 2$
\item $\alpha(1000) = 3$
\item $\alpha(10^{1000}) = 3$
\end{itemize}

inverse Ackermann funct.
b) Deterministic selection (median, etc.) $\Rightarrow$ we'll talk about those applications

- Soft-heap interface

- **Invariant:**
  - After any insertion into Soft-heap, there will be $\leq 3 \cdot m$
  - Corrupted elements in the Soft-heap.

- **Corrupted = Key Increased**

- The number of elements currently in Soft-heap may be much smaller than $m$.

- It may even be possible that all elements in Soft-heap are corrupted.
  - [Actually will not happen]

- It may be the case that all extracted elements are corrupted
**Deterministic Selection**

Let \( X = \{x_1, \ldots, x_n\} \subseteq U \) be an ordered set.

**rank** \( (x) = \left| \left\{ y \in X \mid y \leq x \right\} \right| \)

- e.g. \( \text{min} \) has rad 1
- \( \text{max} \) has rad \( n \)
- median has rad \( \frac{n}{2} \) (\( n \) odd)

**Select** \( (X, k) \)

return element of \( X \) of rad \( k \).

**A good splitter** \( y \in X \) is an element of  

\[
\frac{n}{3} \leq \text{rad}(y) \leq \frac{2n}{3}
\]

more generally: \( \alpha \)-splitter:

\[
\alpha n \leq \text{rad}(y) \leq (1-\alpha)n
\]

for some fixed \( \alpha \in (0, 1) \)

Suppose we can find a good splitter of \( X \) in time \( c \cdot n \)

\[
\begin{align*}
\text{Select} & (X, k) \\
\text{if } k = 1, & \text{return } \text{min} (X) \\
\text{else } & \text{find good splitter } y \\
\text{partition } & \ X \ \text{into } X \leq y \ \cup \ X > y \\
\text{if } k \leq t & \ 	ext{return } \text{Select} \ (X \leq y, k) \\
\text{else } & \text{return } \text{Select} \ (X > y, k-t)
\end{align*}
\]

\[
T(n) \leq c \cdot n + T \left( \frac{2}{3} n \right) \\
\leq c \cdot n \left( 1 + \frac{2}{3} + \left( \frac{2}{3} \right)^2 + \ldots \right) \\
\leq 3 \cdot c \cdot n \leq O(n^3)
\]

Conclusion: Find good splitter in \( O(n) \) time \( \Rightarrow \) solve \ Select \ in \( O(n) \) time

(or any \( \alpha \)-splitter)
How to find a good splitter?
- Randomized → Quickselect
- Deterministically:
  median of median algorithm

Find median of medians
(recursive call to our select)

\( n/5 \) groups of 5

Observe: median of medians is an \( \chi \)-splitter!
\[
\frac{m}{10} + \frac{m}{5^2} \leq \text{smaller}
\]
\[
\leq \frac{3m}{10} \leq \text{smaller}
\]

\( \geq \frac{3m}{10} \text{ larger} \)

\( \Rightarrow \) median of medians is \( \frac{3}{10} \)-splitter

Solution using Soft-heap

\text{Insert} / \text{extract-min}
\leq 3 \cdot m \text{ items are corrupted}
\ # universe

\begin{align*}
X: \text{key} & \quad \text{"true key"} \\
X: \text{current-key} & \geq X: \text{key} \\
X: \text{current-key} & > X: \text{key} \quad \text{if } X \text{ is "corrupted"} \\
X: \text{current-key} & = X: \text{key} \quad \text{if } X \text{ is "not corrupted"} \\
\text{extract-min} & \rightarrow \text{return item with smallest current-key.}
\end{align*}

Soft heap works like a normal min-heap w.r. to current keys.
Find a good splitter of $X = \{x_1, \ldots, x_n\}$

0. Create a Soft-Heap $H$ with $\varepsilon = 1/3$
1. Insert $x_1, \ldots, x_n$ into $H$
2. Extract-min $\frac{n}{3}$ times (call set $A$)
3. Return max key item in $A$ (call it $x$)

Claim: $x$ is a good splitter of $X$

$\# \text{corrupted items} \leq \varepsilon \cdot n = \frac{n}{3} \Rightarrow |B| \geq \frac{n}{3}$

Proof:
- $x$ is max in $A$
- $x$ smaller than all in $B'$ (still in $H$ and not corrupt)

\[
\text{true key of those in } A \leq \text{current key of those in } A \leq \text{current key of those in } B' = \text{true key of those in } B'
\]

Find max of $A$ (acc. to true key)

$\frac{n}{3} \leq \text{rank}(x) \leq 2\frac{n}{3} \quad \Rightarrow x \in A$ is a good splitter

Total cost is $O(n)$

Running time:

0. $O(1)$

1. $O(n)$ (insert)

2. $O(n)$ (extract-min)

2. $O(n)$ (find max)

Total: $O(n)$
Soft heap recap

- `x.key`
- `x.current.key`

- `key`
- `current.key`

- `x.current.key > x.key`

- "x corrupted" \iff \( x.current.key > x.key \)

Parameter \( \varepsilon \): only a \( \varepsilon \)-fraction of insertions may be corrupted.

Soft heap behaves like a normal min-heap according to current key.

Selection recap

\[ \begin{align*}
|A| &= 2n/3 \\
|B| &= n/3 \\
\text{not corrupted} \iff \varepsilon = 1/3 \Rightarrow 3 \leq n/3 \\
\end{align*} \]

- \( x \in A \)
- \( y \in B \)
- \( x.key \leq x.current \leq y.current = y.key \)

Pick \( x \in A \) with max key.
Soft heap implementation

- binary heap

\[ \text{input} \quad O(1) \quad \text{amortized} \quad \text{extract-min} \quad O(1) \]

- each node has an annotated list
  - node key is the max of keys in list
  - node keys satisfy heap order

node is a "representative" for entire list
items in list have current key set to node key

\[ \text{rank of a node } x = \text{distance of } x \text{ from a leaf} \]

\[ \text{rank of leaf is } 0 \]

list sizes grow more slowly

\[ \Gamma = \log \frac{1}{\varepsilon} + C \]

- at rank \( k \), the list size is
  \[ S_k = 1.5 \cdot k - r \]

- at rank \( k \leq r \), the list size is
  \[ S_k = 1 \]
- Extract-min

- Pop an arbitrary element from list of root

- If list size $\geq \frac{S_k}{2}$, nothing else to do

- If list size $\leq \frac{S_k}{2}$, sift-up in the tree, until list size $> S_k$

  Sift-up: move up list of child with smaller key and concatenate with root list. (representative will change)

  Recursively sift-up to fix child node

  If root list still not large enough, sift-up second time

- Insert

  Min pointer

  - Soft-keep

  - List of trees, s.t.

  - Root rails are all distinct

  - Maintain pointers to root with smallest current-key

  Insert

  - Create new tree, singleton root

  - Merge trees until all root rails are unique distinct

  Fix list sizes by doing sift-up

  Update min-pointer
**Analysis**

Claim: \( \# \text{corrupted items} \leq 3 \cdot m \)

Proof:

\( S_k \)

5 items corrupted

5 \( k \)-1 items corrupted

Obs.: \#nodes of rank \( k \) is at most \( \frac{m}{2^k} \)

Proof. Induction \( k = 0 \):

\( \# \text{nodes of rank } 0 \leq m \)

\( \# \text{nodes of rank } 1 \leq \frac{m}{2} \)

\( \# \text{nodes of rank } r \leq \frac{m}{2^k} \)

\( \# \text{corrupted items} \)

\[ \leq \sum_{r \geq 1} \left( \frac{3}{4} \right)^k \cdot \frac{1}{2^k \cdot 2^r} \]

\[ \leq m \cdot \sum_{k \geq 1} \left( \frac{3}{4} \right)^k \cdot \frac{1}{2^k} \]

\[ \leq m \cdot \sum_{k \geq 1} \left( \frac{3}{4} \right)^k \cdot \frac{1}{2^k} \]

\[ \leq m \cdot \left( \frac{3}{4} \right)^2 \cdot \frac{1}{2} \]

\[ \leq \frac{3m}{4} \cdot \sum_{i=0}^{\infty} \left( \frac{3}{4} \right)^i \]

\[ \leq 3 \cdot m \]
Aborted cost of extrac-tion is \( O(\log \frac{1}{\varepsilon}) \)

- Wrong one list up and concatenate
- Worst cost is \( 1 \)
- Cost = \( r \)
- Cost of extra:
  - from \( \varepsilon \) to \( \varepsilon + 1 \)
  - Shred away \( 3k \) items

Cost of a single element moving from \( \varepsilon \) to \( \varepsilon + 1 \):
- First \( r \) steps \( \Rightarrow \) cost \( \frac{1}{\varepsilon} \)
- For steps \( r \rightarrow k \) \( \Rightarrow \) cost \( \sum_{k=r}^{\infty} \frac{1}{5k} \)

Total cost = \( \sum_{k=r}^{\infty} \frac{1}{5k} \leq \log \frac{1}{\varepsilon} + \sum_{k=r}^{\infty} \frac{2}{5k} \leq \frac{2}{5r} \leq \frac{2}{5} \)

\( \leq \log \frac{1}{\varepsilon} + c \)

\( \leq O(\log \frac{1}{\varepsilon}) \)

---

Some notes (see [Kaplan-Zwick] for details):
- \( 3k \) is the size of the list
- only fix when \( n \leq \frac{3k}{2} \)
- true size of a list \( k \geq \frac{3k}{2} \)
- other questions?
- where is win?
- cost of updating win-points?
- analysis of most cost
Soft heap analysis recap

- Insert: $O(k \log 1/k)$
- Extract-min: $O(1)$

Amortized:

- $\varepsilon$-error parameter
- Node key: the key of list node keys are in heap order

$k > \varepsilon$

$$S_k = 1.5 \cdot k - r$$

$$r = \log 1/k + 2$$

- Each root node is unique

Overall min

- Insert: $x$
  - Create a new tree with single root root of rank $0$
  - Merge equal rank roots into new tree
  - Sift up where necessary
  - Update max
    - Merge up to rad $k$ -> update $k$ min

-k + k ->

- k + k ->
Insert costs:

- $\psi \cdot 2^h$ to the element in list
  (will pay for all sift-ups to
  the future)

- $1 \in$ a tree node
  (will pay for all tree mergers)

- $2 \psi$ on tree node
  (will pay for update min)

Cost:

\[
\psi \cdot 3 = O(\log \frac{1}{3})
\]

\[
\frac{1}{\log 4} = \frac{1}{\log 16} = \frac{1}{\log 128}
\]

**Invariant:** each tree root has $1 \in$

each tree root of red $k$

has $k + 2 \psi$

---

1. $\psi$ paid for merge
2. $\psi$ put back to restore invariant

Same analysis as for binary counter

1. Available to pay for update min
2. Pays for update min $k$ to minimum
inequality true when \( k \geq 3r \). To handle the case of \( k < 3r \) we can pay another \( 3r \) \( ¥ \) at the time of insert, and store it with the list element.

\[ 24\sqrt{k(2r)^4} \]

if root node-key changes
need to update within

\[ O(\sqrt{k}) \]

\[ O(1) \]

Claim: \( $ \) pay for all sift-up

actual cost of binary list-up is \( \frac{1}{5k} \)

charge each element \( \frac{1}{5k} \)
$\sum_{i=0}^{\infty} \left( \frac{2}{3} \right)^i = 3$

\[
\frac{1}{S_k} = \left( \frac{2}{3} \right)^{k-r}
\]

Application of Soft-heaps