Soft-heap [Chazelle '00] – ESA "test of time award" 2019

Simplified [Kaplan, Zwick, Tarjan] '13

make-heap (ε)	$O(1)$
insert	$O(\log 1/\varepsilon)$
extract-min	$O(1)$

} Amortized

(meld, delete)

"error-parameter" $\varepsilon \leq 1$

$\varepsilon > 0$ arbitrary constant parameter

Comparison-model (keys only accessed via comparisons)

---

e.g. $\varepsilon = 0.1 \Rightarrow$ extract-min cost $O(1)$

(Thm. one of insert / extract-min must have cost $\Omega(\log n)$) via sorting

Why is this not a contradiction?

Answer: Soft heap "corrupts" an ε-fraction of the keys.

(user has no control over this corruption)

if $\varepsilon < \frac{1}{n}$, cost of operations $O(\log(n))$

if $\varepsilon = 1$

Q: ① What is Soft-heap good for?

② How to implement it?

① a) [Chazelle] Deterministic MST $O(m\,\alpha(n))$

\# edges, \# vtcs.

inverse Ackermann-fct.

almost optimal

best known deterministic MST algorithm

$\alpha(n)$ extremely slowly growing fct.

e.g. $\alpha(5) = 1$

$\alpha(10) = 2$

$\alpha(100) = 3$

$\alpha(10^{10}) = 3$

$\alpha(10^{1000}) = 3$

(there is randomized $O(m)$ expected-time algorithm)

---

not covered in this class

b) - deterministic selection (median, etc.)
- 2018+ "structured selection"

we'll talk about these applications

Soft-heap interface

Invariant:
after doing m insertions into soft-heap, there will be $\leq \varepsilon \cdot m$ corrupted elements in the soft-heap.

corrupted = key increased

\# 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

$X = \{x_1, \dots, x_n\} \subseteq \mathcal{U}$ ← ordered set

$\mathrm{rank}(x) = |\{y \in X \mid y \leq x\}|$
$x \in X$

e.g. min. has rank 1
max. has rank n
median has rank $\frac{n+1}{2}$ (n odd)

Select (X, k)

return element of X of rank k.

A good splitter $y \in X$ is an element of ($|X| = n$)

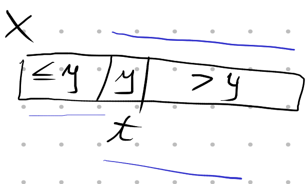
$$\frac{n}{3} \leq \mathrm{rank}(y) \leq \frac{2n}{3}$$

more generally: α-splitter: for some fixed $\alpha \in (0,1)$

$$\alpha n \leq \mathrm{rank}(y) \leq (1-\alpha) n$$

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

Select (X, k)
- $O(n)$ if $k=1$, return $\min(X)$
- $c \cdot n$ find good splitter y
- $O(n)$ partition X into $X_{\leq y}$, $X_{>y}$
- $O(1)$ $t = \mathrm{rank}(y) = |X_{\leq y}|$
- if $k \leq t$
  - return Select $(X_{\leq y}, k)$
- else
  - return Select $(X_{>y}, k-t)$



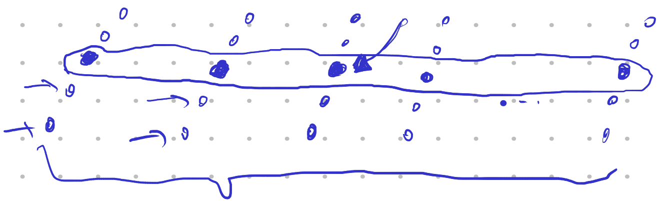
$$T(n) \leq c' \cdot n + T\left(\tfrac{2}{3} n\right)$$
$$\leq c' \cdot n \underbrace{\left(1 + \tfrac{2}{3} + \left(\tfrac{2}{3}\right)^2 + \dots\right)}_{3}$$
$$\leq 3 \cdot c' n \in O(n)$$

Conclusion: Find good splitter in $O(n)$ time $\Rightarrow$ Solve Select in $O(n)$ time
(or any α-splitter)

How to find a good splitter?

- randomized $\longrightarrow$ Quickselect
- deterministically:

  median of medians algorithm [Blum, Floyd, Pratt, Rivest, Tarjan] 1973 (see in any Algo-textbook)



$n/5$ groups of 5 $\longrightarrow$ find median of medians (recursive call to our Select)

Obs. median of medians is an α-splitter!

$\geq \frac{n}{10} + \frac{n}{5 \cdot 2} \cdot 2$ smaller

$= \frac{3n}{10}$ smaller

$\geq \frac{3n}{10}$ larger

$\Longrightarrow$ median of medians is $\frac{3}{10}$-splitter

---

Solution using Soft-heap

insert / extract-min

$\leq \varepsilon \cdot m$ item are corrupted

$\quad m$: # insertions

$X.key$ "true key"

$X.currentkey \geq X.key$

$X.currentkey > X.key$ if X is "corrupted"

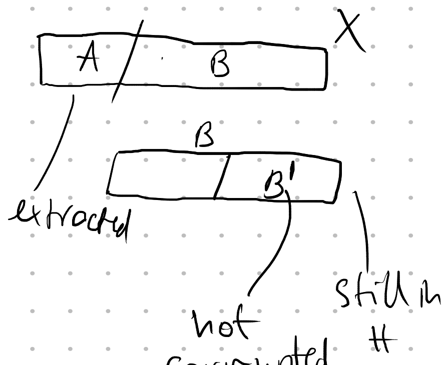
$X.currentkey = X.key$ if X "not corrupted"

extract-min $\longrightarrow$ returns item with smallest current-key.

Soft heap works like a normal min-heap w.r.t. to current keys.

Finding a good splitter of $X = \{x_1, \dots, x_n\}$

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



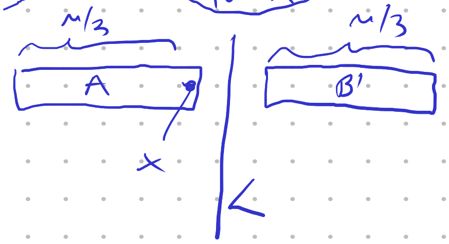
$|A| = n/3$

Claim: x is a good splitter of X

$\#$ corrupted items $\leq \varepsilon \cdot n = n/3 \Rightarrow |B| \geq n/3$

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

true key of those in A $\leq$ current key of those in A $\leq$ current key of those in B' $=$ true key of those in B'



find max of A (acc. to true key)
$= x$

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

total cost is $O(n)$

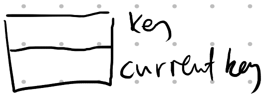
Running time:

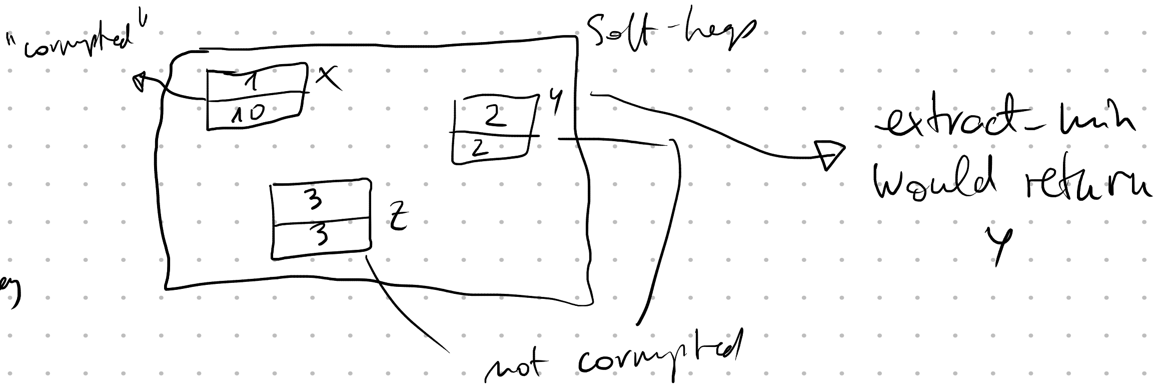
0. $O(1)$
1. $O(n)$ ($n \times$ insert)
2. $O(n)$ ($\frac{n}{3} \times$ extract min) $O(\log 1/\varepsilon) \in O(1)$
3. $O(n)$ (finding max)

Total: $O(n)$

Soft heap recap

X.key
x.currentkey





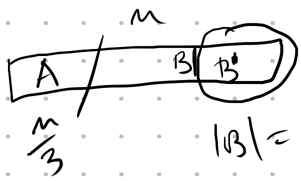
$x.\text{currentkey} \geq x.\text{key}$

"x corrupted" $\iff x.\text{currentkey} > x.\text{key}$

parameter ε: only ε-fraction of inserted items may be corrupted

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

Selection recap



$|B| = \frac{2n}{3}$

$\varepsilon = 1/3 \Rightarrow \#\text{ corrupted} \leq \frac{n}{3}$

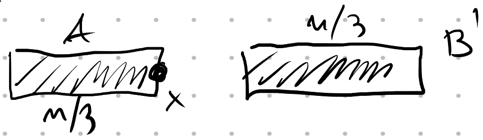
$\Rightarrow \exists \frac{n}{3}$ not corrupted

extract-min

$\forall x \in A$ $\qquad \forall y \in B'$

$x.\text{key} \leq x.\text{current} \leq y.\text{current} = y.\text{key}$

pick $x \in A$ with max key

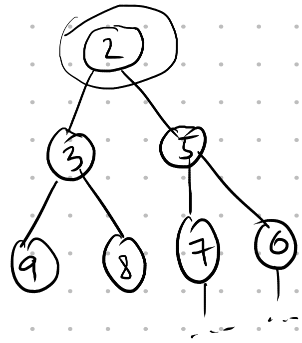


Soft heap implementation

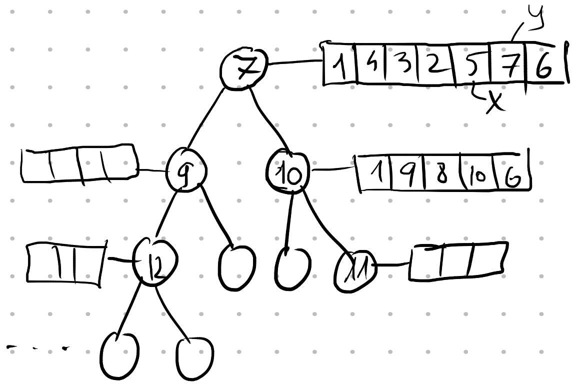
ε - error parameter

insert $O(\log 1/\varepsilon)$ } amortized
extract-min $O(1)$ }

- binary heap



$O(\log n)$



$x.key = 5$
$x.current = 7$
x is corrupted

$y.key = y.current$
y is not corrupted

- each node has an associated list
- nodekey is the max of keys in list
- nodekeys satisfy heap order

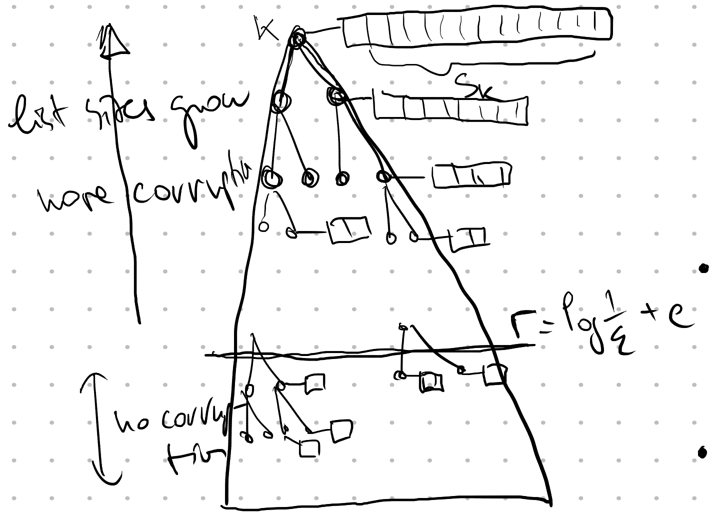
---

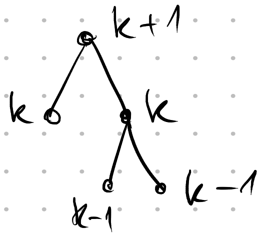
node is a "representative" for entire list
items in list have currentkey set to nodekey

---

rank of a node x = distance of x from a leaf

rank of leaf is 0





- at rank $k > r$, the ideal list size is

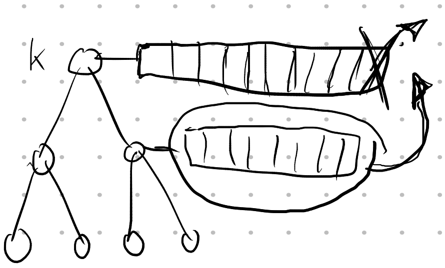
$$S_k = 1.5^{k-r}$$

- at rank $k \le r$, the list size is

$$S_k = 1$$

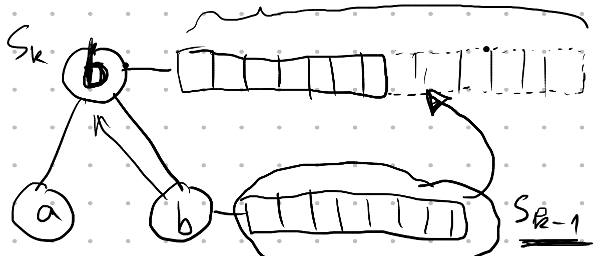
- extract-min

pop an arbitrary element from list of root



ideal list size S_k

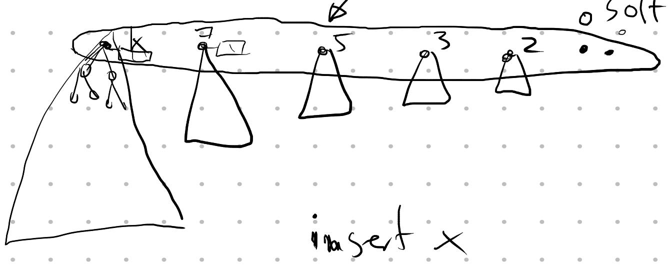
- if list size $> \frac{S_k}{2}$ nothing else to do
- if list size $\leq \frac{S_k}{2}$, sift-up in the tree, until list size $\geq S_k$



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

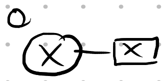
recursively sift-up to fix child node
if root list still not large enough, sift-up second time

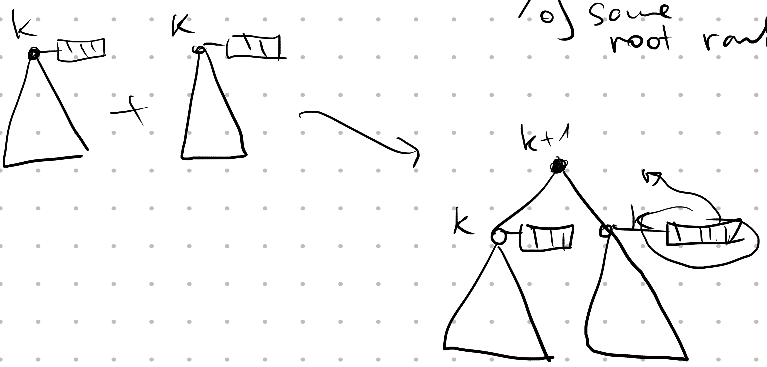
- insert



- soft-heap
- list of trees, s.t. root ranks are all distinct
- maintain pointer to root with smallest current-key

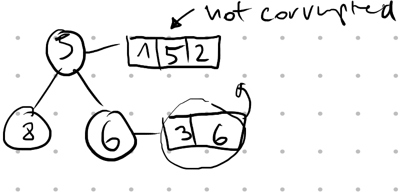
insert x

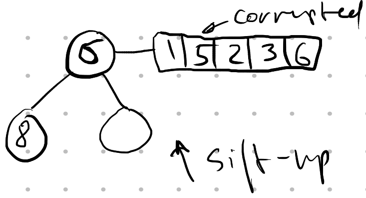
- create new tree, singleton root 
- merge trees until all root ranks are ~~unique~~ distinct of same root rank



- fix list sizes by doing sift-up
- update min-pointer

e.g





"current key was increased"
$=$
corruption

Analysis

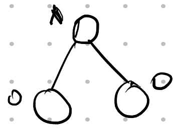
Claim 1) # corrupted items is $\le \varepsilon \cdot n$
inserted

Proof

$s_k - 1$ items corrupted

Obs. # nodes of rank k is at most $\frac{n}{2^k}$

Proof. Induction $k=0$



nodes of rank $0 \le n$

nodes of rank $1 \le \frac{n}{2}$

...

nodes of rank $k \le \frac{n}{2^k}$ ∎

$\boxed{\text{\# corrupted items}}$

$$\le \sum_{\substack{\text{rank } k\\ k>r}} \left(\begin{matrix}\text{\# nodes of}\\ \text{rank } k\end{matrix}\right) \cdot s_k$$

$$\le \sum_{k>r} \frac{n}{2^k} \cdot (1.5)^{k-r}$$

$$\le n \cdot \sum_{k>r} \left(\frac{3}{2}\right)^{k-r} \cdot \frac{1}{2^{k-r} \cdot 2^r}$$

$$\le n \cdot \sum_{k>r} \left(\frac{3}{4}\right)^{k-r} \cdot \frac{1}{2^r}$$

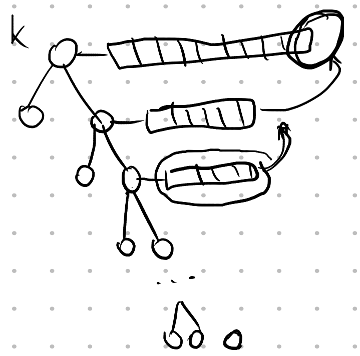
$r = \log \frac{1}{\varepsilon} + \overset{2}{\underset{=}{c}}$

$$\le n \cdot \sum_{k>r} \left(\frac{3}{4}\right)^{k-r} \cdot \underbrace{\frac{1}{2^{\log 1/\varepsilon + 2}}}_{= \frac{\varepsilon}{4}}$$

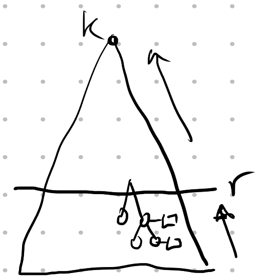
$$\le \frac{\varepsilon n}{\cancel{4}} \cdot \underbrace{\sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^{i}}_{\cancel{4}}$$

$$\boxed{\le \varepsilon \cdot n}$$

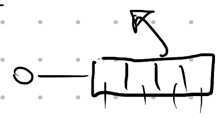
Claim 2) Amortized cost of ~~extract-min~~ siftup is $O(\log 1/\varepsilon)$



• moving one list up and concatenating w. parent list cost $\underline{1}$

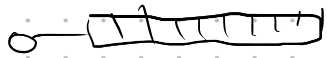


cost $= r$



"car-pooling"

cost of moving from rank $k \to k+1$



shared among s_k items

Cost of a single element moving from rank 0 to rank k.

• First r steps $\to$ cost r

• For steps $r \ldots k$ $\to$ cost $\sum_{k>r} \frac{1}{s_k}$

$$\text{Total cost} = r + \sum_{k>r} \frac{1}{s_k} \le \log\frac{1}{\varepsilon} + \sum_{k>r} \left(\frac{2}{3}\right)^{k-r}$$

$$\sum_{k>r} \left(\frac{2}{3}\right)^{k-r} \le \sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^{i} \in O(1)$$

$$\le \log\frac{1}{\varepsilon} + c$$

$$\in O\left(\log\frac{1}{\varepsilon}\right)$$

Loose ends (see [Kaplan-Zwick] for details)

• $\underline{s_k}$ is "ideal" size of a list

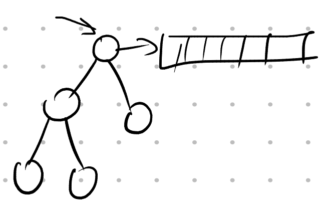
only fix when size is $\le \frac{s_k}{2}$

true size of a list $\ge \frac{s_k}{2}$

$\le \underline{3 s_k}$

• other operations? (delete / meld)

min current key



• where is min?

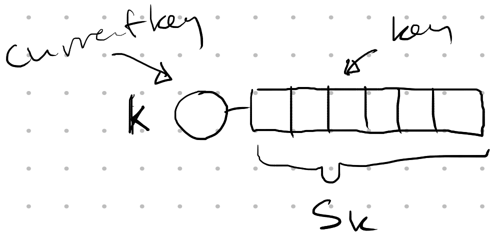
Cost of updating min-pointer?

• analysis of insert cost

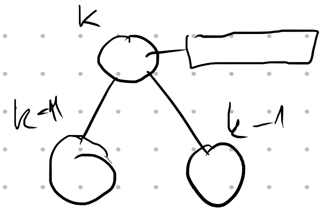
Soft heap analysis recap

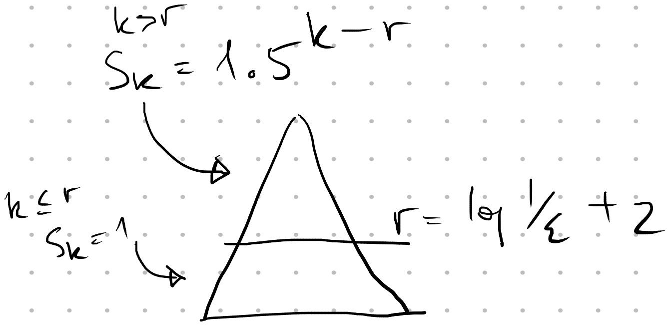
make-heap $O(1)$
insert $O(\log 1/\varepsilon)$
extract_min $O(1)$
} amortized

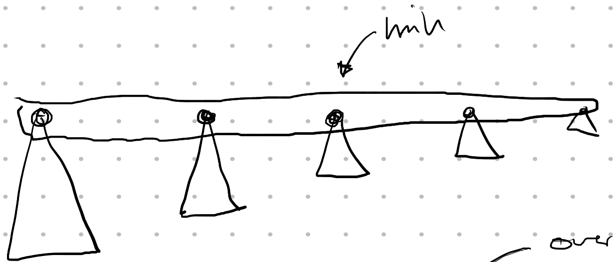
ε - error parameter



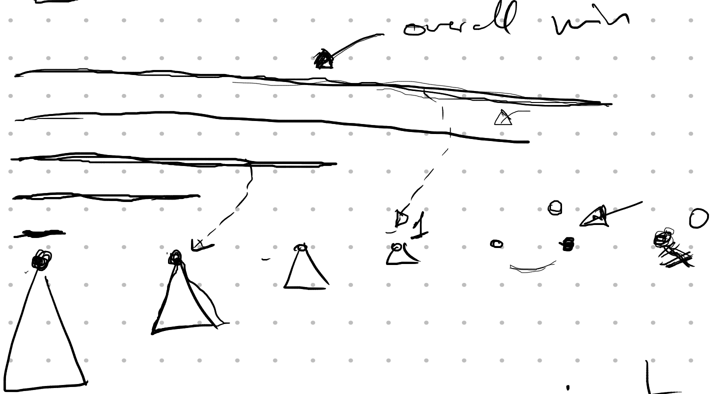
nodekey is max key of list
nodekeys are in heap order





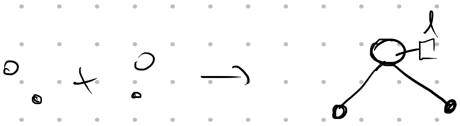


each root rank is unique

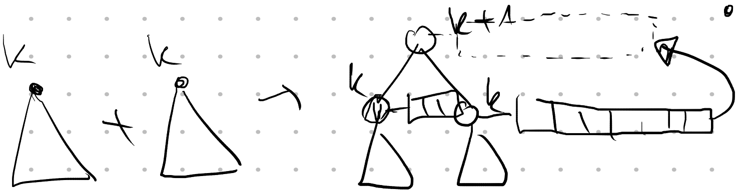


insert: x

- create a new tree with singleton root of rank 0



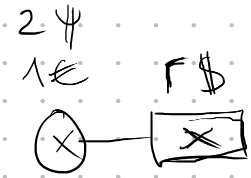
- merge equal rank ~~roots~~ into new tree



- sift up where necessary
- update min

(merge up to rank k → update k minima

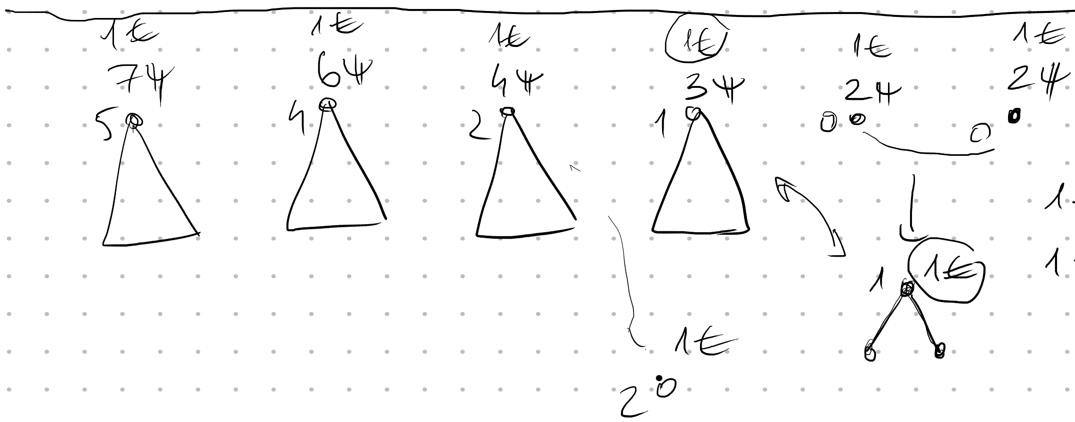
insert costs



- $r^{(+3)}$ \$ to the element in list (will pay for all sift-ups to x in the future)
- 1 € on tree node (will pay for all tree merges)
- 2 Ψ on tree node (will pay for updating minimum)

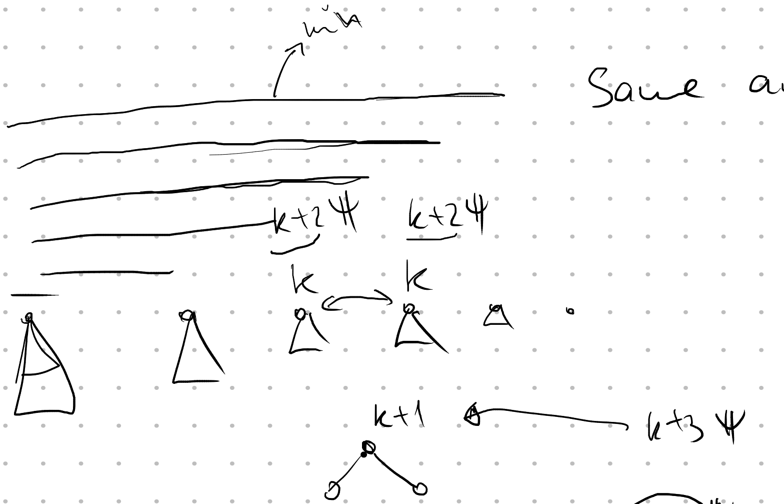
cost:
$\underline{r+3} = O(\log \frac{1}{\varepsilon})$
$\overset{''}{\log \frac{1}{\varepsilon}}$

<u>Invariant</u>: each tree root has 1€
each tree root of rank k has k+2 Ψ



1€ paid for merge
1€ put back to restore invariant

Same analysis as for binary counter



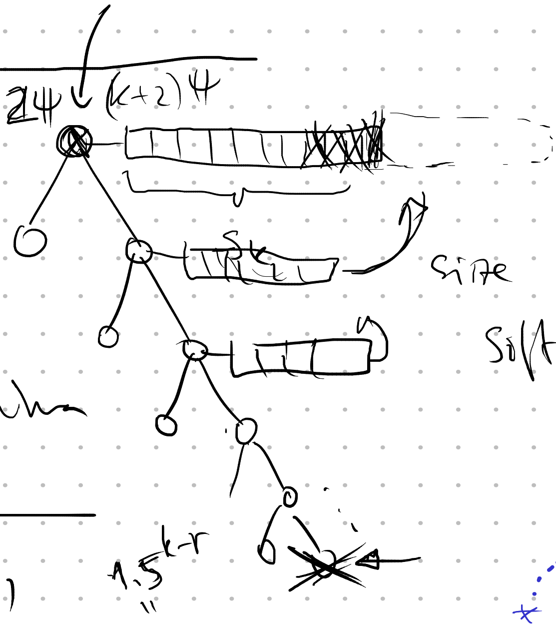
(k+1)Ψ available to pay for update min
→ pays for updating last k minima

insert → everything accounted for (except sift-up)

extract-min:

cost: 1 ¥

if root node-key changes need to update min



size drops below $\frac{s_k}{2}$ sift-up from children

ineqality true when k>=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

$\boxed{3r¥}$ .. this can pay for .. update-min

Obs. when I sift-up, I had already done $\frac{s_k}{2}$ extracting $\geq k$

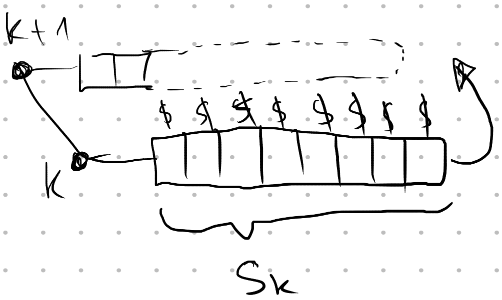
1.5^{k-r} "

these k ¥ pay for updating min

cost of extract-min $O(1)$

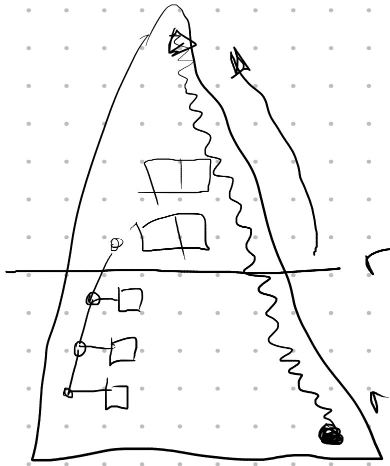
(sift-up to be analyzed)

Claim: $'s pay for all sift-up



actual cost of moving list up ≤ 1

charge each element $\frac{1}{s_k}$ $



r

rank 0 , r\$

$$\frac{1}{s_k} = \left(\frac{2}{3}\right)^{k-r}$$

$$\sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^{i} = 3$$

r\$

Applications of Soft-heaps