A Linear Time Algorithm for the k Maximum Sums Problem

By Gerth S. Brodal and

Allan G. Jørgensen

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

AARHUS UNIVERSITET

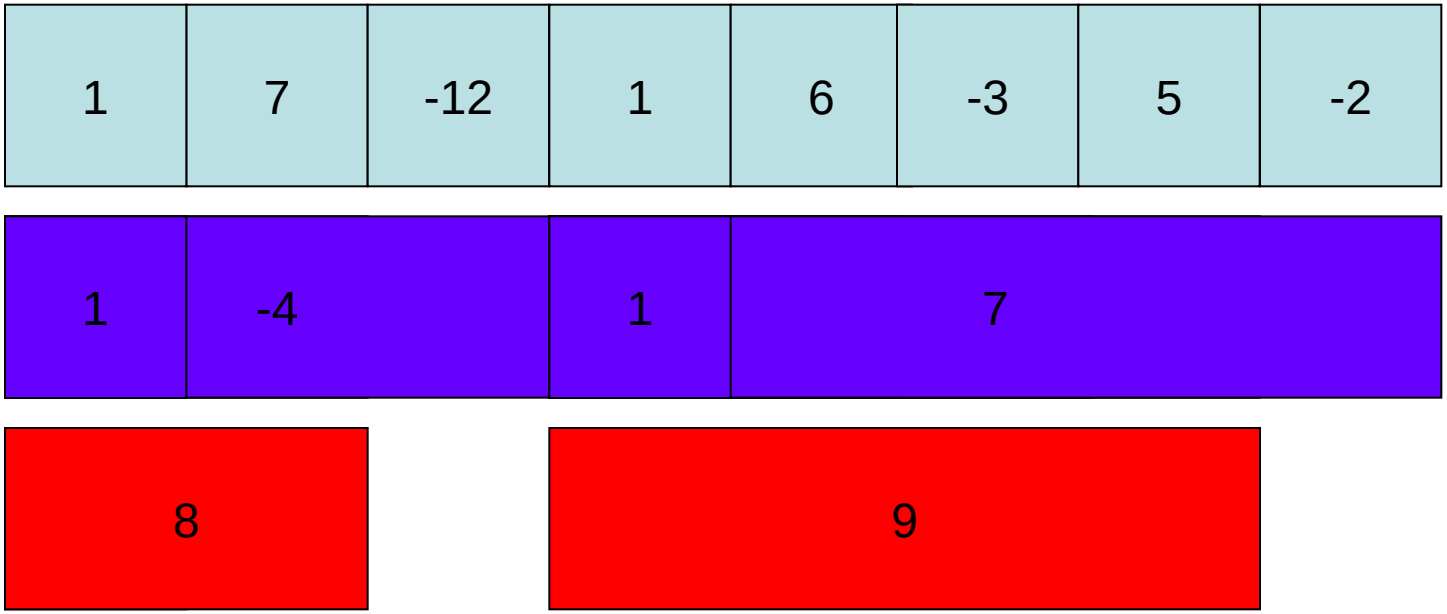
The Maximum Sum Problem

- Given array of numbers, find the subarray(vector) with the largest aggregate

-3	7	-12	1	6	-3	5	-2

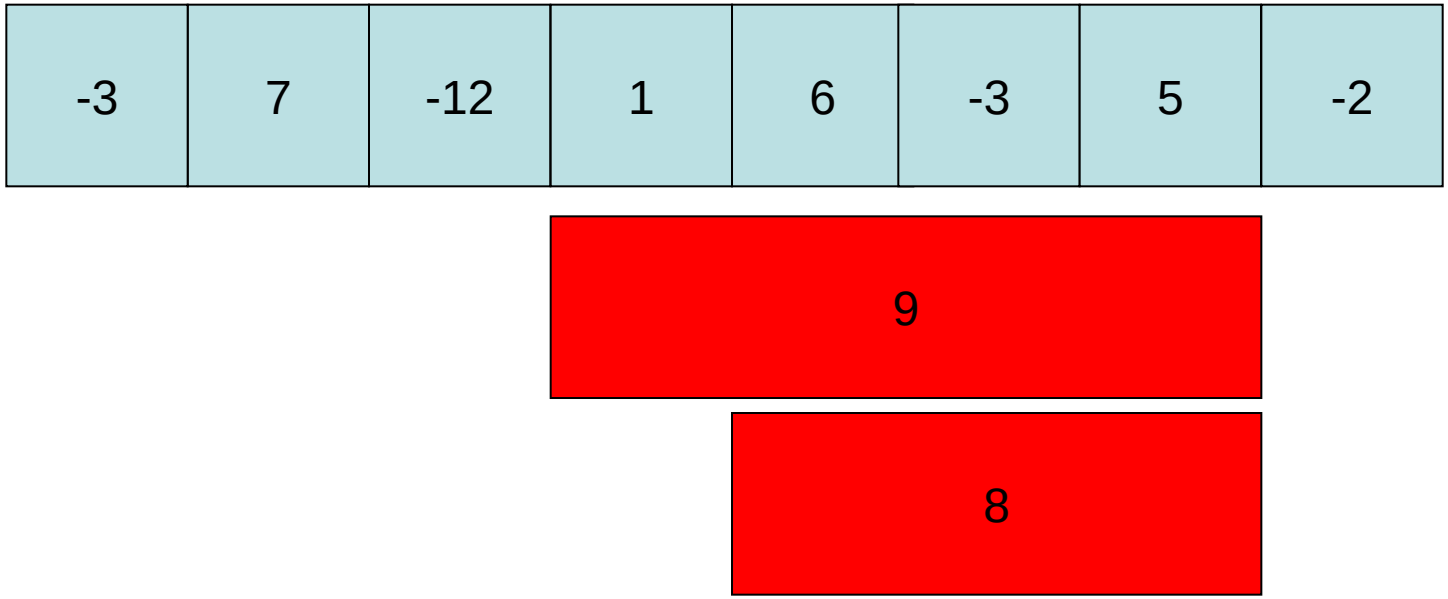
Kadanes Algorithm

- Scan array and update
- Best current suffix
- Best sum so far



The k Maximal Sums Problem

- Given array of numbers, find the k subarrays with the largest aggregates. They may overlap
- Example with $k=2$



Main Idea (Intuition)

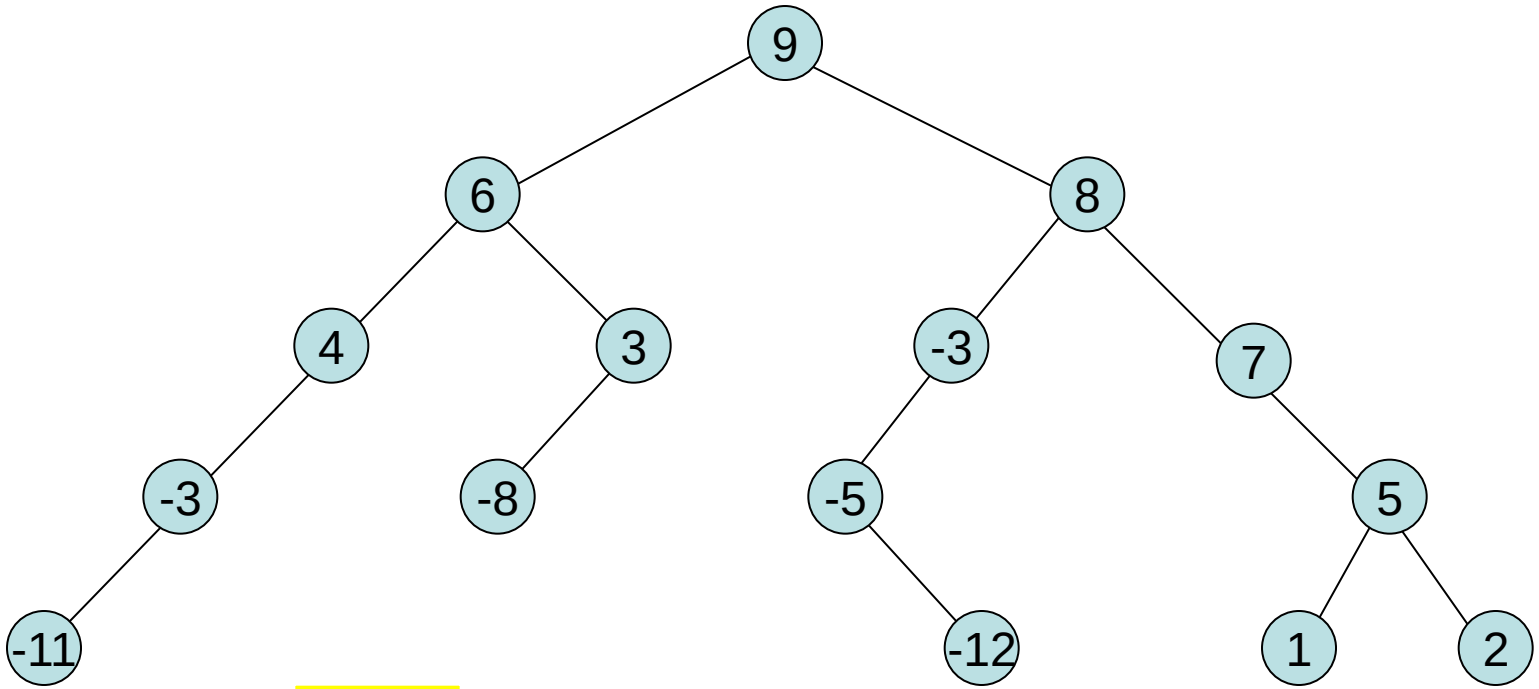
- Build all sums and insert them into a heap ordered binary tree. There are

$$\binom{n}{2} + n = O(n^2)$$

- Find the k largest using Frederickson's heap selection algorithm

Example(k=4)

-12	1	6	-3	5	-2



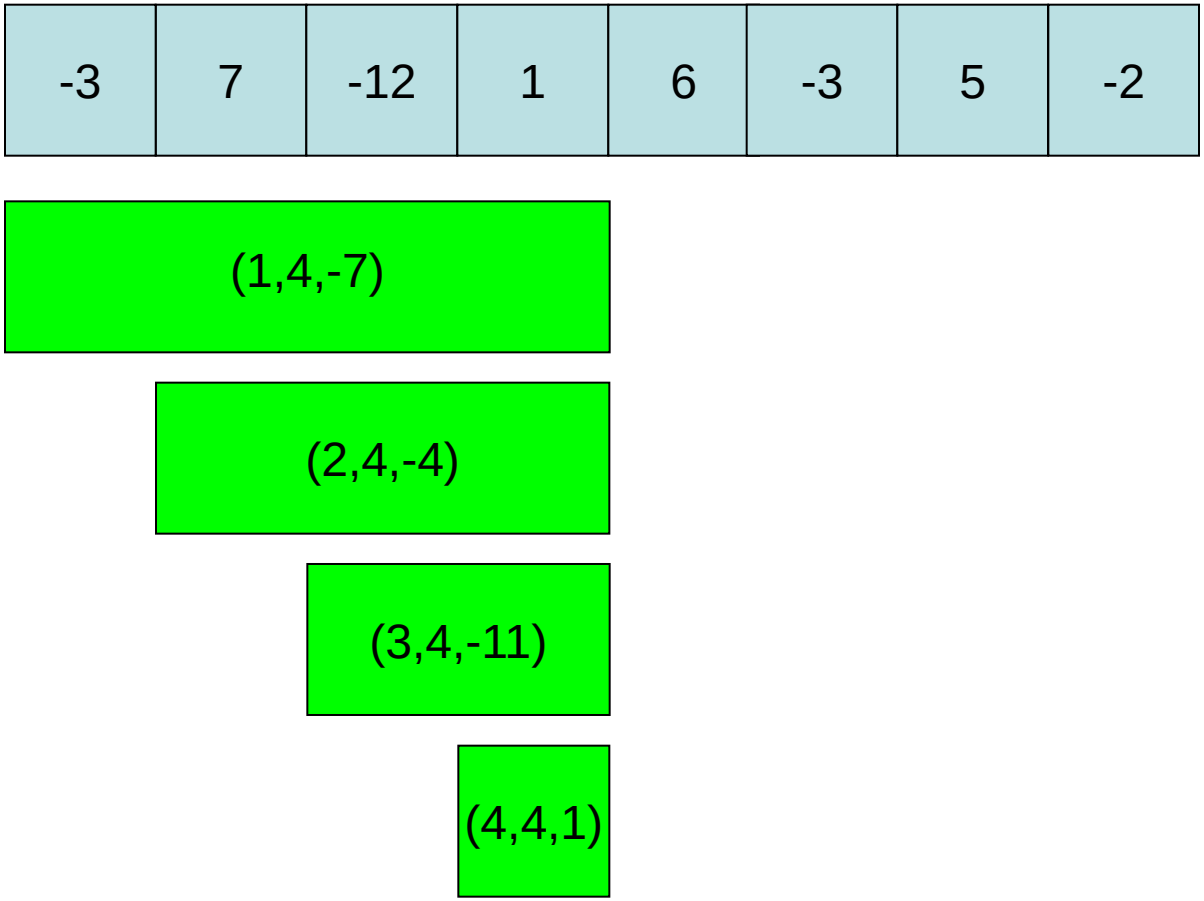
$= O(k)$

Representing the Sums

- The partial sums are grouped by their endpoint in the array
- The partial sums ending at index j is

$$Q_{\text{suf}}^{j} = \left\{ (i, j, sum) \,|\, 1 \le i \le j \wedge sum = \sum_{s=i}^{j} A[s] \right\}$$

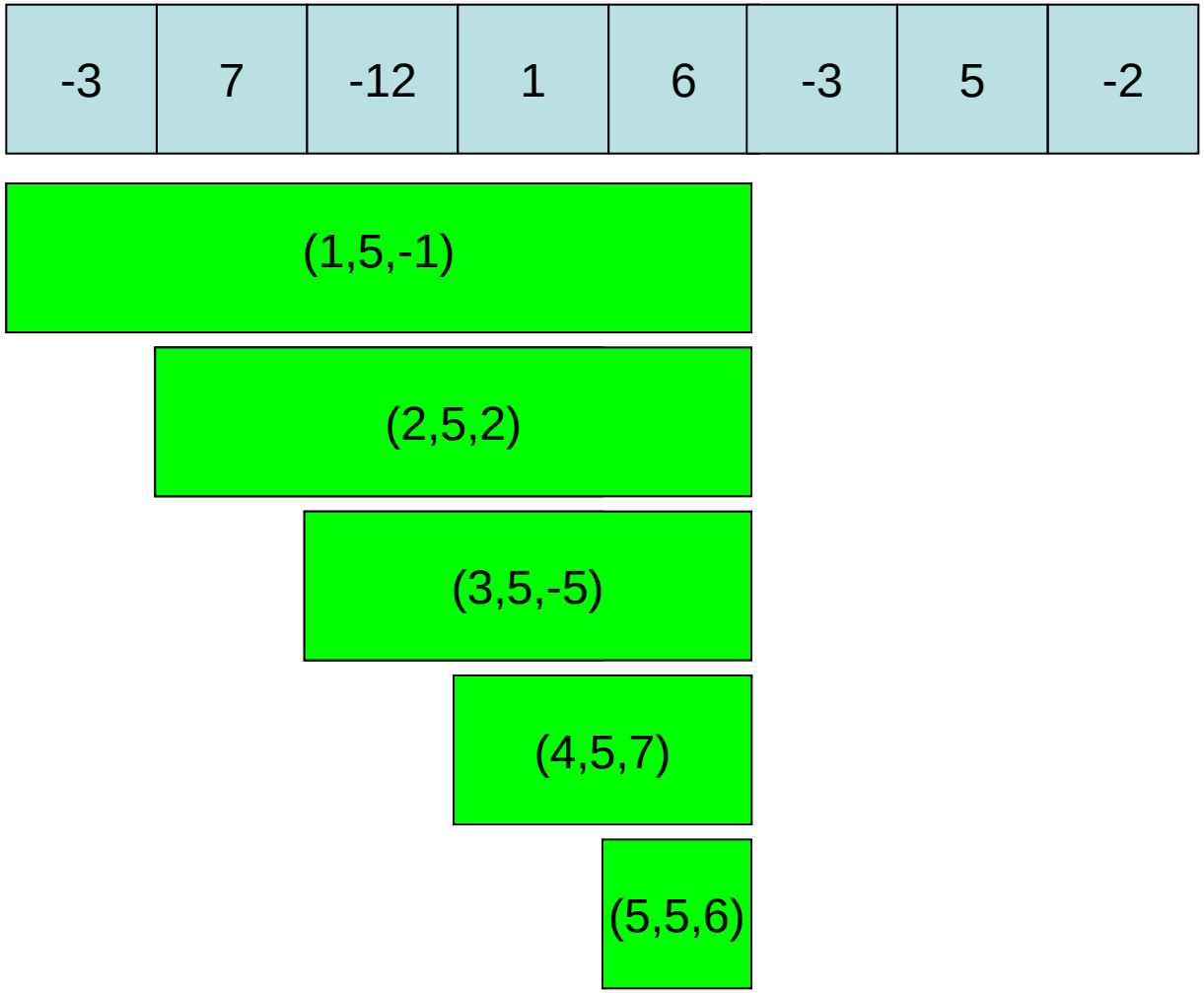
Q^4_{suf}



Construction Equation

$$Q_{\text{suf}}^{j} = \{(j, j, A[j])\} \cup$$
$$\{(i, j, s + A[j]) \mid (i, j-1, s) \in Q_{\text{suf}}^{j-1}\}$$

Constructing Q^5_{suf} from Q^4_{suf}

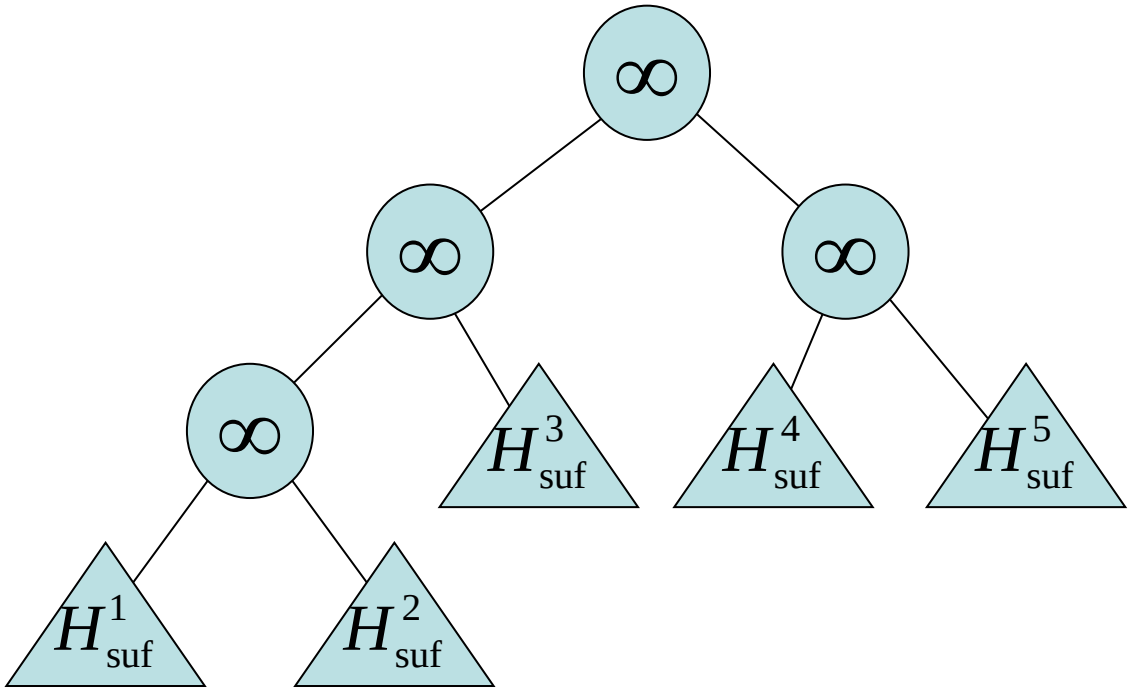


Main Idea Continued

- Represent the suffix sets as heap ordered binary trees

- Combine to a single heap by assembling them into one big heap using dummy infinity keys.

The Assembled Heap

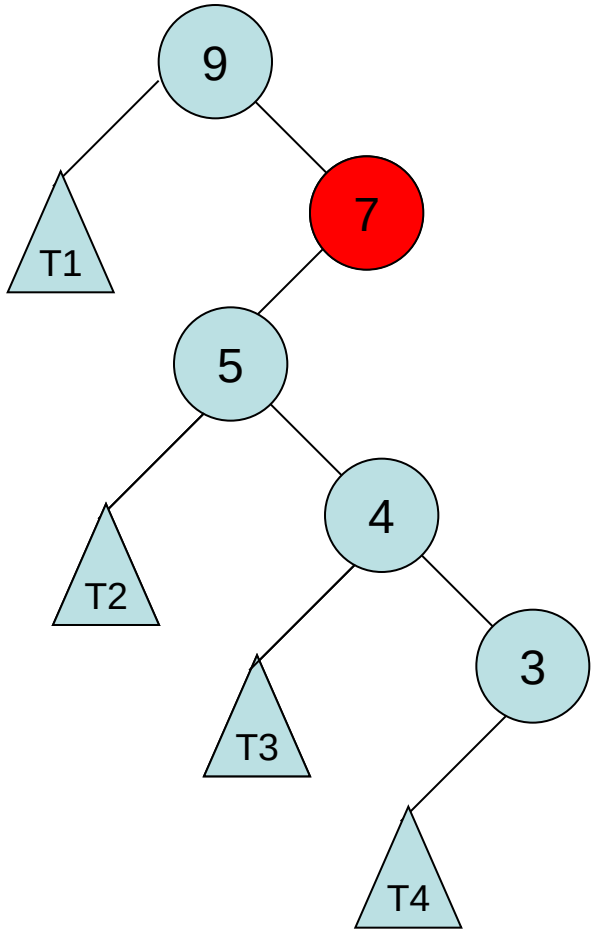


The Iheap

- It is a heap ordered binary tree

- Supports insertions in amortized constant time

Inserting 7 in an Iheap



Representing suffix sets:

- Each set Q_{suf}^{j} is represent by a tuple $<\delta_j, H_{\text{suf}}^{j}>$
- H is an Iheap containg all the elements
- δ_j is number must be added to all elements.
- We get the following construction equation.

$$<\delta_0, H_{\text{suf}}^{0}>=<0,\{\}>$$

$$<\delta_{j+1}, H_{\text{suf}}^{\text{j+1}}>=<\delta_j + A[j+1], H_{\text{suf}}^{j} \bigcup \{-\delta_j\}>$$

Example

-3	7	-12

Insert - $\delta_0 (= 0)$

Set $\delta_1 = \delta_0 - 3 (= 3)$

$\delta_1 = -3$

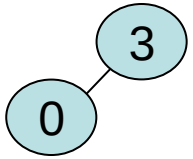
{-3}

Insert $- \delta_1 (= 3)$

Set $\delta_2 = \delta_1 + 7 (= 4)$

$\delta_2 = 4$

{4,7}

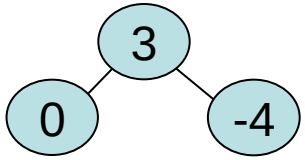


Insert $- \delta_2 (= -4)$

Set $\delta_3 = \delta_2 - 12 (= -8)$

$\delta_3 = -8$

{-8,-5,-12}



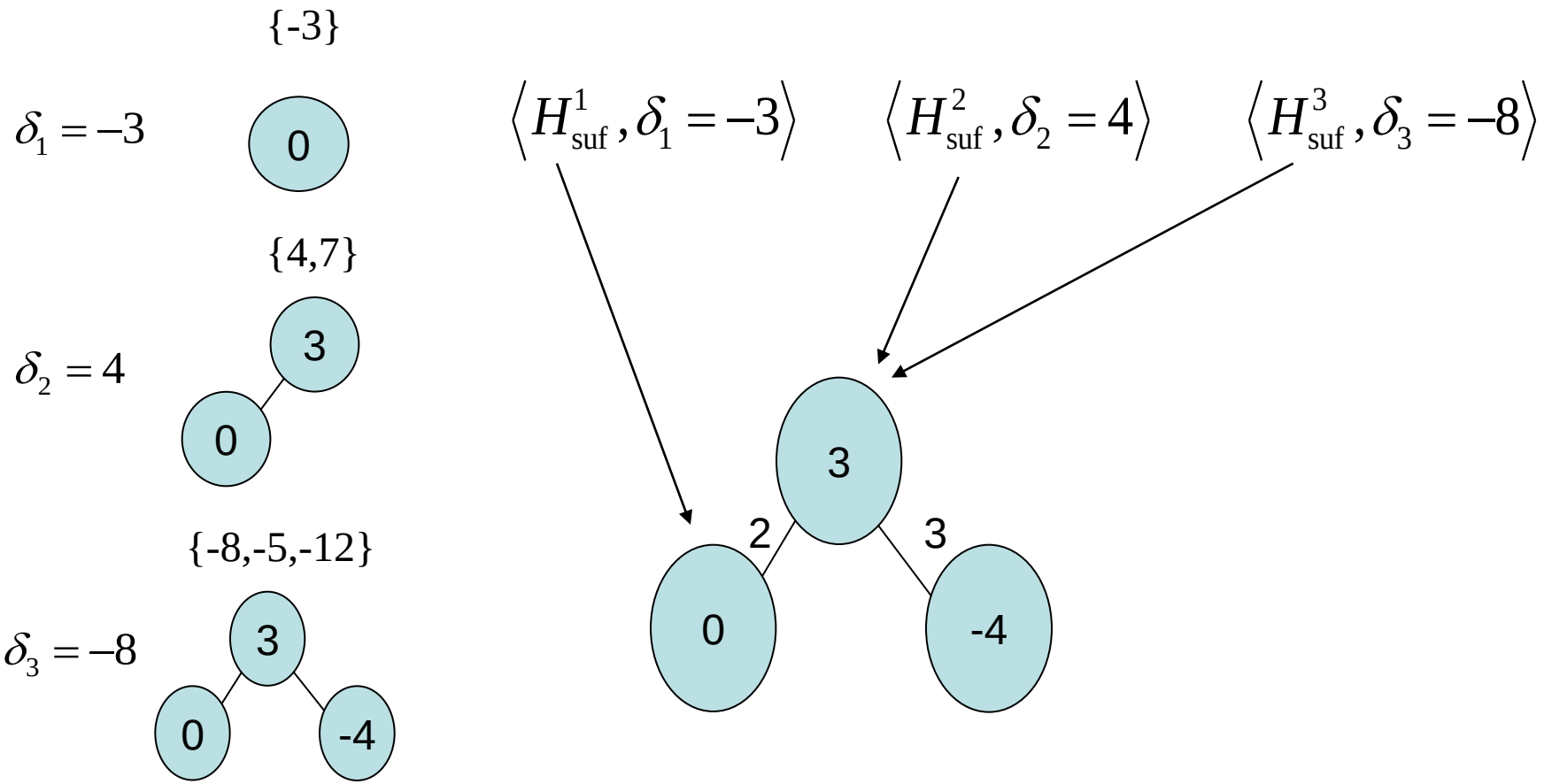
Pair Construction

- Builiding each pair takes amortized constant time.(One insertion into Iheap)
- !! But the old version disapears
- Solution: Partial Persistence.

Partial Persistence

- **Driscoll, Sarnak, Sleator, and Tarjan. Making Data Structures Persistent**
- Allows queris to any version.
- Using node copying, the extra cost per update is amortized the cost of copying O(1) original nodes. Query time is the same.

Before and After



Last Problem

- The resulting data structure is no longer a heap ordered binary tree
- Fix by incremental construction following Fredericksons algorithm, which works top down

Resume

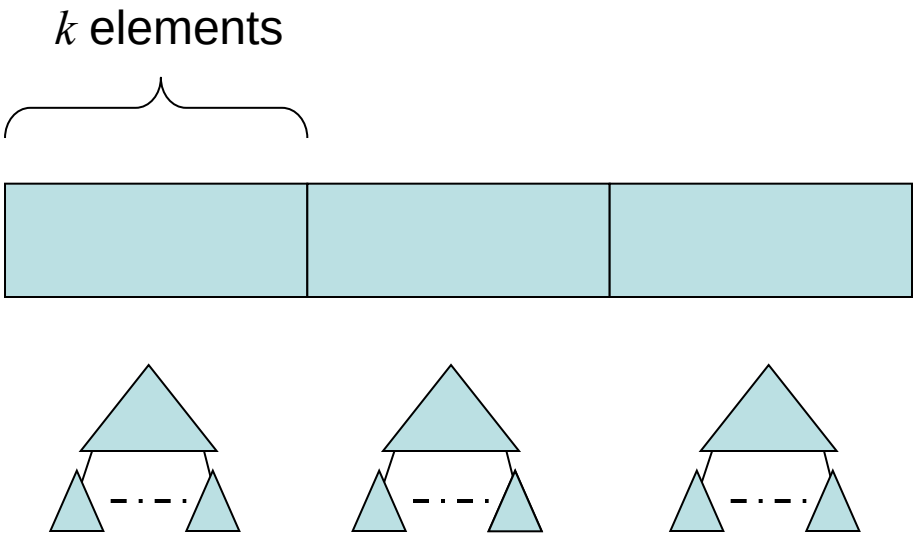
- Build all suffix heaps in $O(n)$ time
- Join them into a single heap
- Use incremental construction while performing Fredericksons algorithm
- All in all $O(n+k)$

Space reduction

- Current algorithm uses $O(n+k)$ time and additional space. The input array is considered read only.
- Kadanes algorithm uses $O(1)$ additional space.
- We want to reduce the additional space usage to $O(k)$

Algorithm

- Build the k heaps, and find the k largest as before
- Repeat on the next k, using the last built heap. The rest can be discarded.
- Merge the two sets using selection.
- ! The size of the last heap is growing
- Only the k best suffixes are usefull. Find these from the last heap and build a new heap with these.
- Repeat on next k …

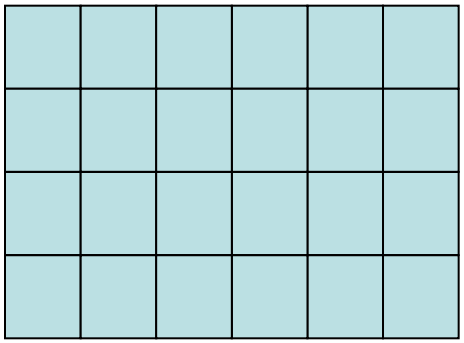


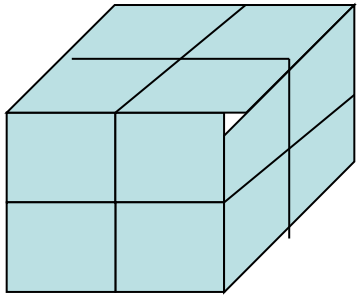
$$\left|H^{k}_{\text{suf}}\right| = k \quad \left|\overline{H}^{2k}_{\text{suf}}\right| = k$$

Flash-Back

- For $k=1$ this algorithm and Kadanes algorithm from the introduction are the same.

Higher Dimensions





.....

Can be reduced to 1D case.

- For an m x n matrix, we get
  $O(m^2 n + k)$ time, $O(n + k)$ space
- In general we get
  $O(n_1 \prod_{i=2}^{d} n_i^2 + k)$ time, $O(\prod_{i=1}^{d-1} n_i + k)$ space

Thank You!