Chih-Hung Liu

Taiwan

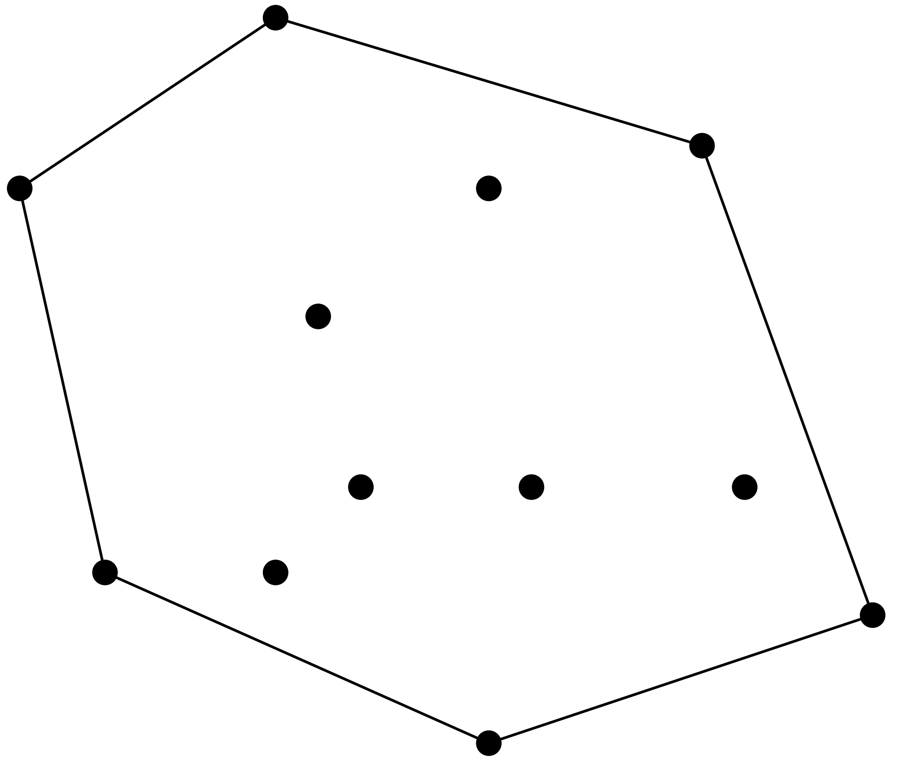
B. S., Computer Science, National Taiwan University

Ph. D, Electronics Engineering, National Taiwan University

chliu@uni-bonn.de

Selected Topics in Algorithmics

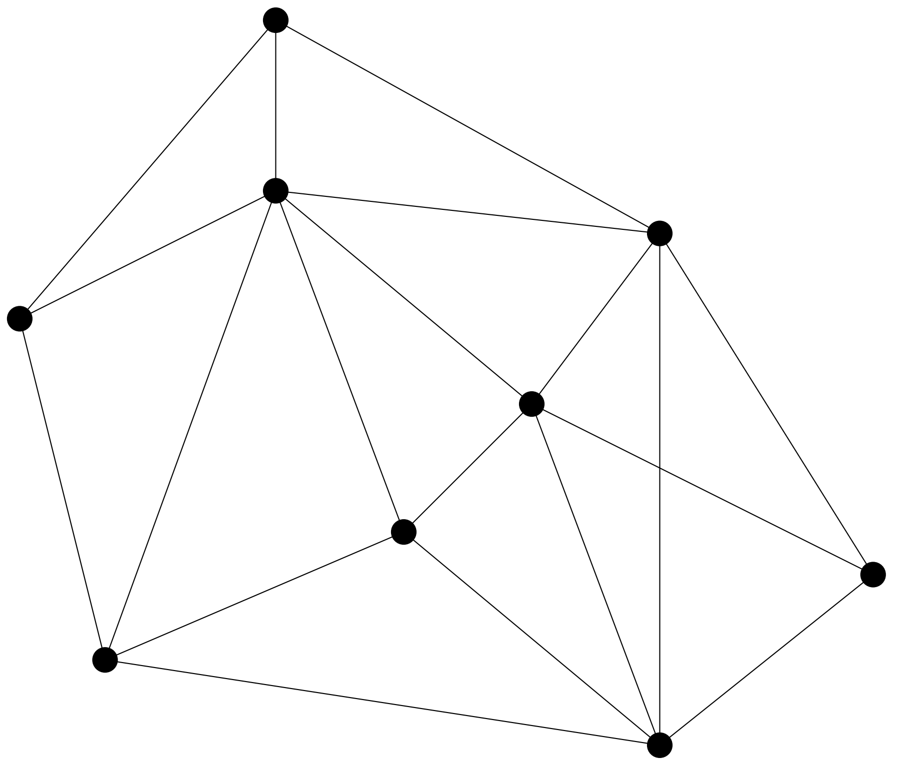
Randomized Algorithms for **Geometric Structures**



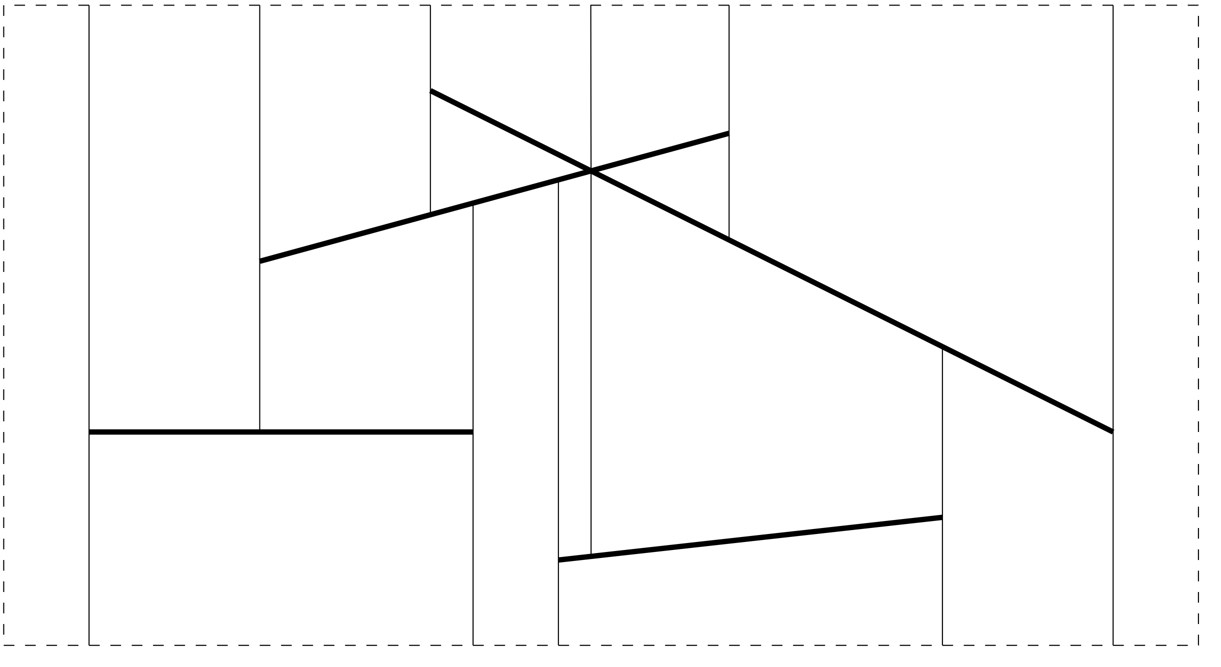
Convex Hull

A set $C \subseteq \mathbb{R}^2$ is **convex** if for any two points $p, q \in C$, $\overline{pq} \subseteq C$.

For a set S of points, the convex hull of S is the minimum convex set containing S

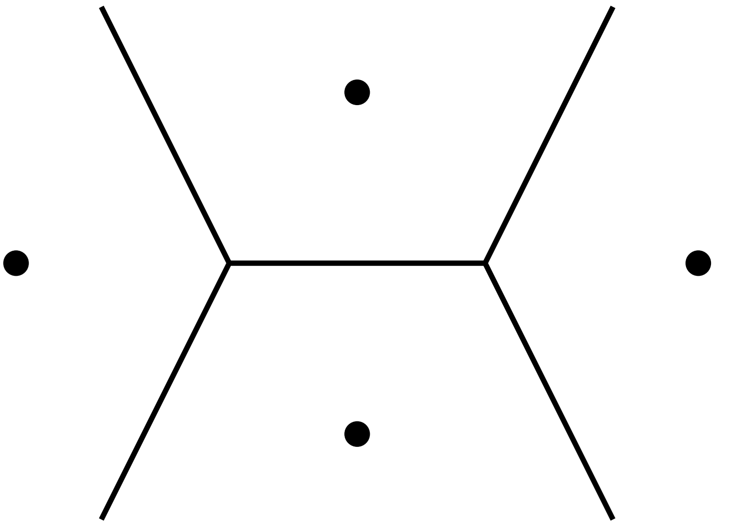


For a set S of points, a **triangulation** of S is a maximal collection of edges among S without any edge crossing



For a set S of line segments, the vertical trapezoidal decomposition of S is constructed as follows:

- Pass a vertical attachment through every endpoint or point of intersection
- Each vertical attachement exteneds upwards and downwards until it hit another segment or if no such segment exist, it extends to infinity



For a set S of point sites, the Voronoi diagram of S is a planar subdivision such that all points in a region share the same nearest site among S

A **randomized algorithms** we are interested in this lecture is an algorithm which will make **random choices** during the computation. For example, Quick sort can be viewed as a randomized algorithm if the pivot is selected randomly.

Advantages

- Simpler Structure
  - Easy for implementation
  - Constant inside the Big-O is small
- Worst-case hardly happen
  - more efficient in practice
  - Quick-sort is the most efficient sorting algorithm in practice.

Main topics

- Randomized Incrmental Construction
- Randomized Divided and Conquer
- Their Applications

Referance Book:
Ketan Mulmuley,
Computational Geometry: An Introduction Through Randomized Algorithms,
Prentice Hall, 1993

- Lecture notes depict the main ideas
- For more details, please refer to the books and related papers.

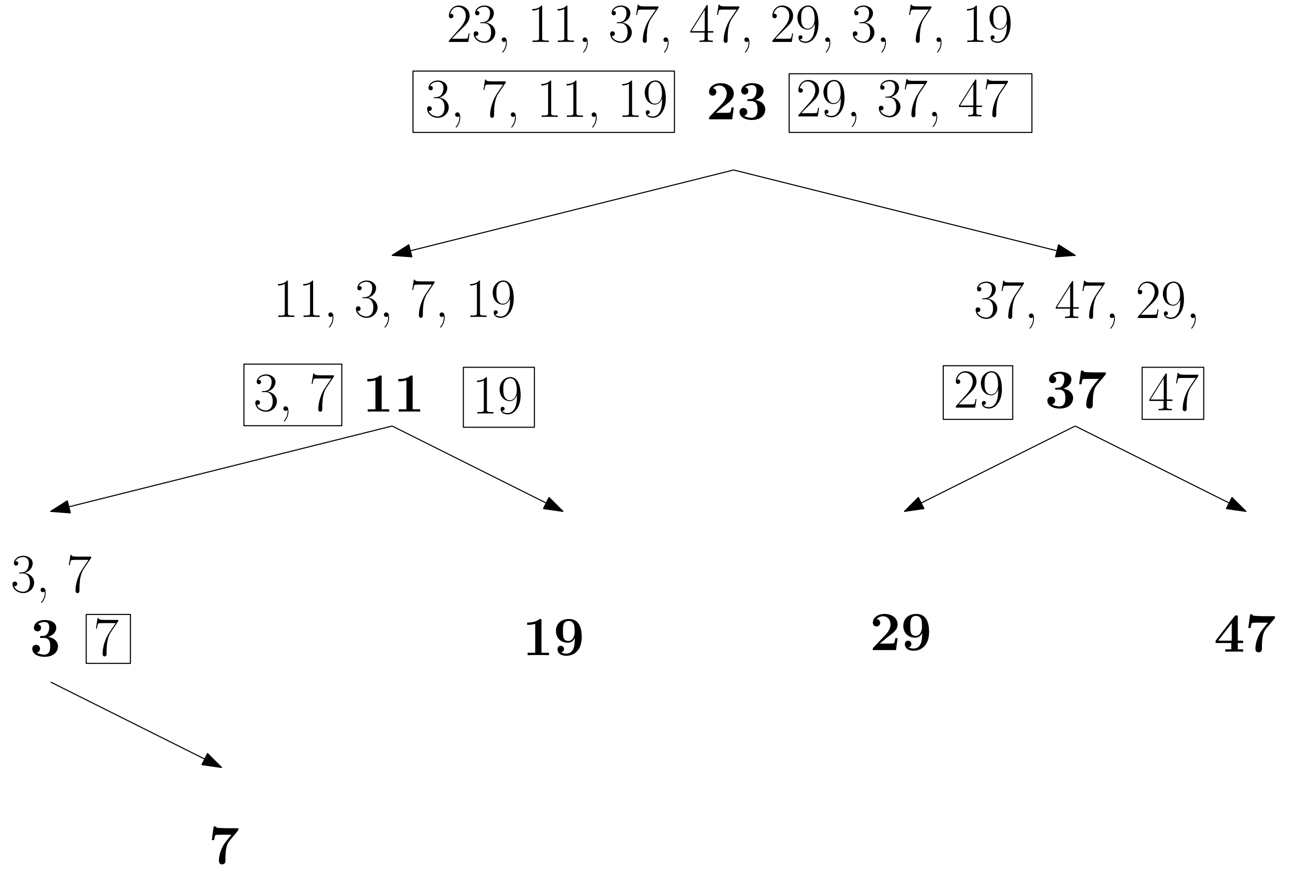
1. Quick Sort And Search

Input: a set N of n real numbers (distinct)

Output: an ordered sequence of N

Qucik-Sort(N)

1. If $|N| = 1$, return N.
2. Select a number p from N
3. Let N_L be $\{l \mid l \in N \text{ and } l < p\}$
   Let N_R be $\{r \mid r \in N \text{ and } r > p\}$
4. If $|N_L| > 0$, $L =$ Quick-Sort(N_L); else $L = \emptyset$
5. If $|N_R| > 0$, $L =$ Quick-Sort(N_R); else $R = \emptyset$
6. return a seqeuence L, p, R

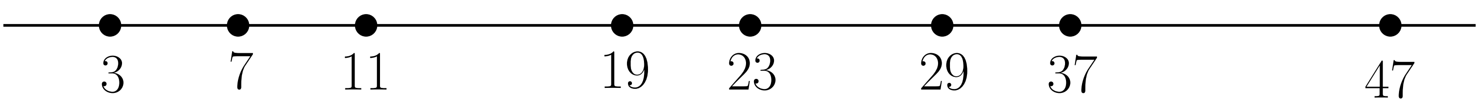


Expected Time Complexity

- If a subset has k elements, it takes $O(k)$ comparisons.
- If a level has m subsets, N_1, N_2, ..., N_m, since they are distinct, a level needs $\sum_{i=1}^{m} O(|N_i|) = O(n)$.
- Expected size of N_L (or N_R)$= \frac{n}{2}$, expected depth of recursion $= O(logn)$
- $O(n \log n)$ expected time

Sorting ⟷ Geometric Structure

An Ordered Seqeuence = A Partition of Real Line R



- **Sorting Problem:**

  Find the partition $\boldsymbol{H(N)}$ of R formed by the given set N of n points.

- **Search Problem:**

Associate a search structure $\widetilde{H}(N)$ with $H(N)$ so that, given any point $q \in R$, one can locate the interval in $H(N)$ containing q quickly, e.g., in logarithmic time.

1.1 Randomized Incremental Version of Quick Sort

$S_1, S_2, \ldots, S_n$: a **random seqeuence** of N

$N^0 = \emptyset$ $N^i = \{S_1, S_2, \ldots, S_i\}$

$H(N^0)$ is R

$H(N^i)$ is the partition of R by N^i

Randomized Incremental Construction:
$H(N^0)$, $H(N^1)$, $H(N^2)$, ..., $H(N^n) = H(N)$.

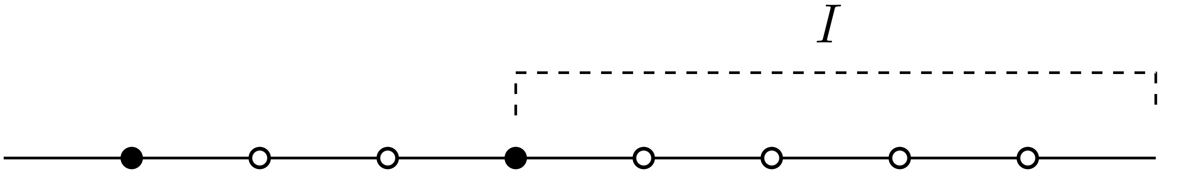


Fig 2. $H(N^2)$ ● points in N^2 ○ points in $N \setminus N^2$

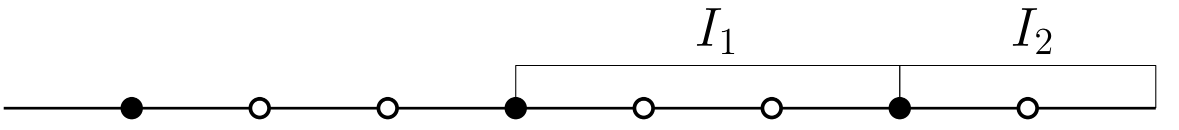


Fig 3. Addition of the third point S^3

Conflict List:
For each interval I in $H(N^i)$, conflict list $\boldsymbol{L(I)}$ is an unsorted list of points in $N \setminus N^i$ contained by I, and $l(I)$ is the size of $L(I)$

E.g., in Fig. 2, $L(I)$ has four points.

Fact
Each point in $N \setminus N^i$ is related to a unique interval in $H(N^i)$.

There is a unique edge between a point in $N \setminus N^i$ and its conflicted interval in $H(N^i)$.

Adding a point $S = S^{i+1}$ into N^i

1. Find a interval I in $H(N^i)$ which contains S.
2. Separate I by S into I_L and I_R.
3. Compute $L(I_L)$ and $L(I_R)$ by $L(I)$

Adding S takes $\boldsymbol{O(l(I_L) + l(I_R) + 1)}$

1. Finding I takes $O(1)$ due to the unique edge between S and I in the conflict list.
2. Separtating I takes $O(1)$ time
3. Computing $L(I_L)$ and $L(I_R)$ takes $O(l(L)) = O(l(I_L)+l(I_R)+1)$ time.

Backward Time Analysis

Inserting S^{i+1} into $H(N^i)$ $=$ Deleting S^{i+1} from $H(N^{i+1})$

Each point S in N^{i+1} is equally likely to be S^{i+1}.
$I_L(S)$: Interval left to S
$I_R(S)$: Interval right to S

Expected Time of Adding S:

$$\frac{1}{i+1}\sum_{S\in N^{i+1}} O(l(I_L(S)) + l(I_R(S)) + 1)$$

$$\leq \quad \frac{2}{i+1}\sum_{J\in H(N^{i+1})} O(l(J)+1)$$

Each interval are adjacent to at most two points

$$= \quad O\left(\frac{n}{i+1}\right)$$

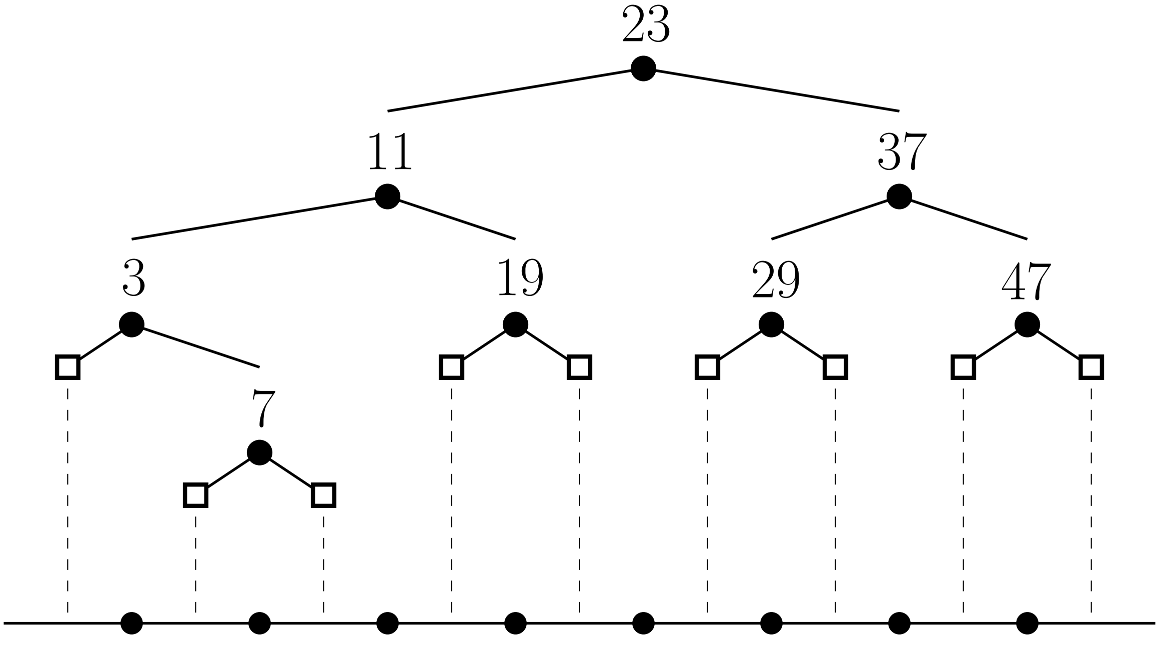
Expected Time Complexity of Randomized Incremental Version:

$$\sum_{i=1}^{n} O\left(\frac{n}{i+1}\right) = O(n \log n)$$

1.2 Randomized Binary Tree

$$N = \{\ 23,\ 11,\ 37,\ 47,\ 29,\ 3,\ 7,\ 19\ \}$$
$$S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8$$

Divide-and-Conquer Quick-Sort



Random Binary Tree $\widetilde{H}(N)$ is defined as follows:

- If $N = \emptyset$, $\widetilde{H}(N)$ is a node corresponding to the whole real line R
- otherwise,
  - the root of $\widetilde{H}(N)$ is a randomly chosen point $S \in N$
  - $\widetilde{H}(N_L)$ and $\widetilde{H}(N_R)$ are defined recursively for the havles of R on the two sides of S, where N_L and N_R are the sets of points in $N \setminus S$ left to and right to S, respectively.

Search Problem:
Given a point $q \in R$, we locate the invertval in $H(N)$ containing q by applying a binary search on $\widetilde{H}(N)$.

Expected search time = expected depth of $\widetilde{H}(N) = O(\log n)$

1.3 History (On-Line)

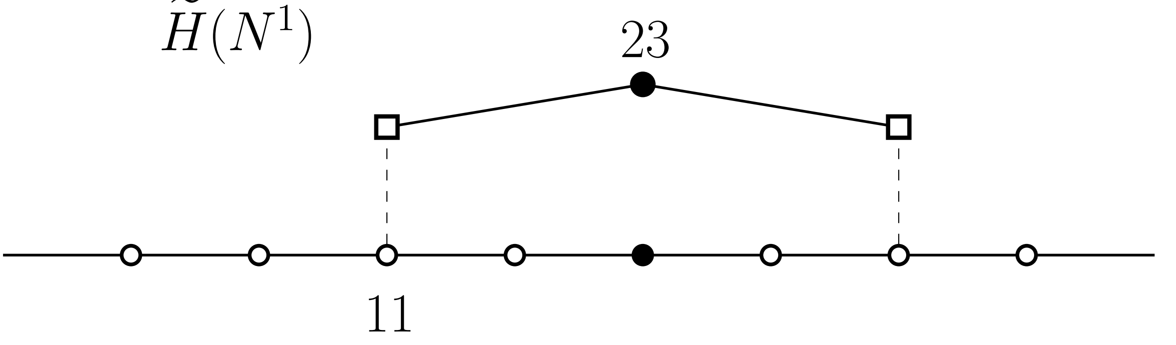
Randomized Incremental Version of Quick-Sort
through the Random Binary Tree

- Locating the interval using the binary tree

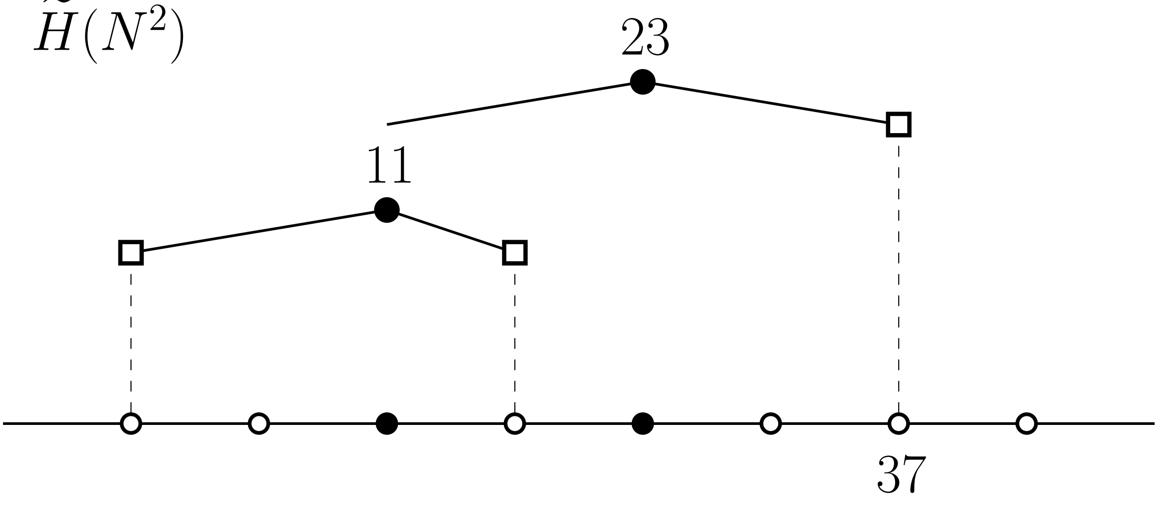
$S_1, S_2, \ldots, S_n$ is a random seqeuence of N

(23, 11, 37, 47, 29, 3, 7, 19)

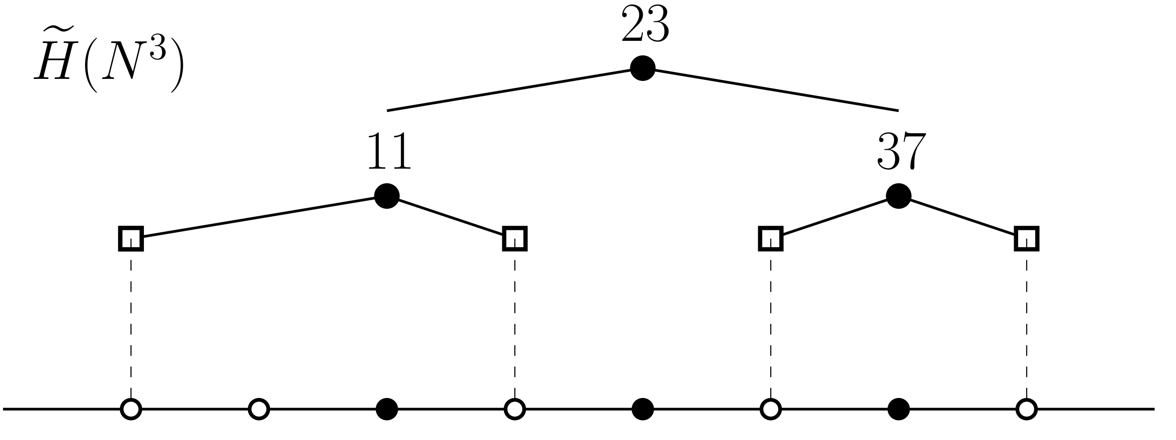
$\widetilde{H}(N^1)$

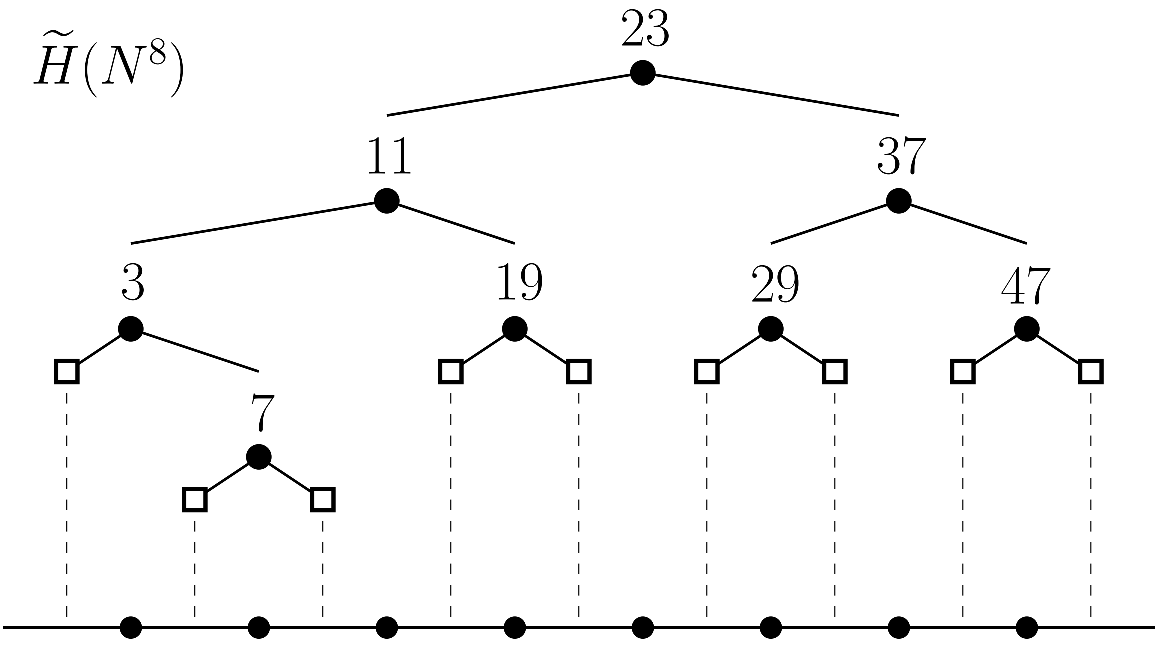


$\widetilde{H}(N^2)$



$\widetilde{H}(N^3)$





Property: If S_j is the left child of S_i, S_j must belong to the left Interval of S_i in $H(N^i)$.

Cost of Inserting S_j = Searching which interval S_j is located in

= Length of Search Path

Backward Analysis

For a query pint q, the search cost is analyzed as follows:

- If the search tests S_i,
  q must belong to the left or right interval of S_i in $H(N^i)$
  $\rightarrow$ probability of testing S_i is $2/i$
- Expected length of search path is $\sum_{i=1}^{n} 2/i = O(\log n)$
- Similarly, inserting S_i takes $O(\log i)$ time

Total Time of Constructing $\widetilde{H}(N)$:

$$\sum_{i=1}^{n} O(\log i) = O(n \log n)$$

This randomized incremental construction through a random binary tree does not require conflict lists:

An on-line algorithm

history(i)

- $\widetilde{H}(N^i)$
- Auxiliary Information
  - Each internal node of $\widetilde{H}(N^i)$ records the left and right intervals when it was created.
  - Each interval records the creation and the deletion time (if it is dead).

history(i)

- Contains the entire history of construction, $\widetilde{H}(N^0), \widetilde{H}(N^1), \ldots, \widetilde{H}(N^n)$.
- Allow searching in $\widetilde{H}(N^i)$ by the auxiliary information.