CSE520: Computational Geometry
Lecture 10
Introduction to Randomized Incremental Algorithms

Antoine Vigneron

Ulsan National Institute of Science and Technology

June 15, 2020

Introduction

- Randomized incremental constructions are introduced.
- Simple example: a modified QUICKSORT.
- New techniques:
  - Random permutation.
  - Backwards analysis.
- Data structures:
  - Conflict list, history graph.
  - Random binary search trees.

Reference (this lecture differs substantially) :

- Textbook Chapter 6.
- Dave Mount's lecture notes, lectures 14, 15 and 18.

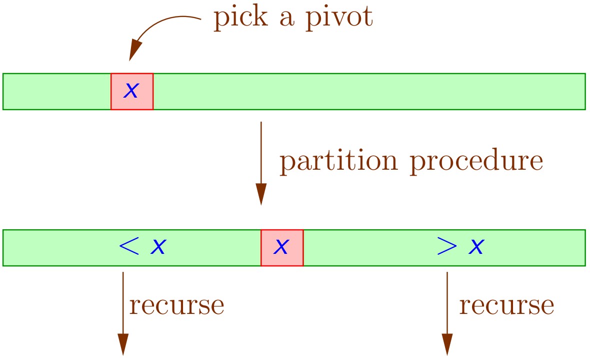
Closer reference: Mulmuley's textbook Chapter 1.

Introduction

- A different way of looking at QUICKSORT.
- We present it as a randomized incremental construction.
- Also called RIC (Randomized Incremental Construction).
- It will generalize to important geometric problems:
  (Next lectures)
  - Point location.
  - Linear programming.
  - Voronoi diagrams
  - Delaunay triangulation.
  - Convex hull in $\mathbb{R}^d$... (not in CSE520)
- Simpler than known deterministic algorithms for these problems, and fast in practice.

Quicksort

- Input: a set S of n real numbers.
- Output: S, sorted.
- Idea:



Randomization

- Usual QUICKSORT: Pick a pivot at random at each step.
- Here: First compute a *random permutation* of S.
  - Input: A set S of n real numbers.
  - Output: A permutation $(x_1, x_2 \dots x_n)$ of S, chosen uniformly at random.

Random permutation:

- First step of the RIC.
- This is the only random aspect of the RIC.
- Expected running time: The expectation is the average over the $n!$ possible permutations.
  - It does *not* depend on the input.

Computing a Random Permutation

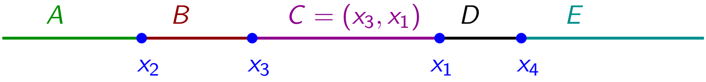
Random permutation

Algorithm *Permute*(A)
Input: An array $A[1 \dots n]$ of numbers.
Output: $A[1 \dots n]$, randomly permuted.
1. **for** $i = n$ **downto** 2
2. **do** $j \leftarrow$ random integer in $\{1, \dots, i\}$
3. Swap(A[i],A[j])

- Runs in $\Theta(n)$ time.
- Generates each permutation with probability $1/n!$.
- After computing *Permute*(S):
  - $\forall x \in S \, \forall i \quad \Pr(x = x_i) = 1/n$.
  - We denote $S_i = \{x_1, x_2 \dots x_i\}$.

Idea

- Insert x_1 first, then x_2 … and finally x_n.
- At step i:
  - S_i partitions $\mathbb{R}$ into $i+1$ intervals.



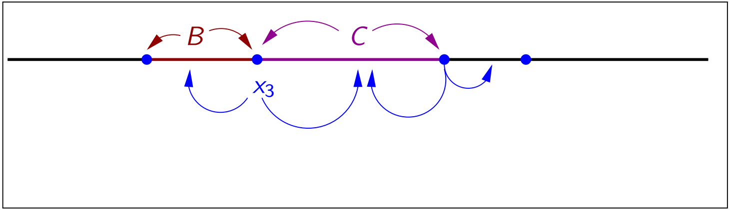
$S_4 = \{x_1, x_2, x_3, x_4\}$

$\mathbb{R} \setminus S_4 = A \cup B \cup C \cup D \cup E$

  - This partition is stored in an appropriate data structure.
  - Insert x_{i+1}, update the partition.
  - The final partition gives $S = S_n$ in sorted order.

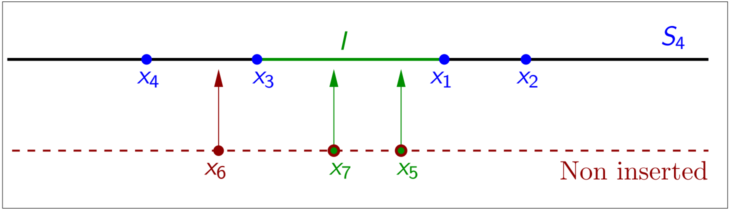
Data Structure: Preliminary

- At step i:
  - Each interval stores pointers to its two endpoints.
  - For each $j \leqslant i$, we store pointers from x_j to its two adjacent intervals.



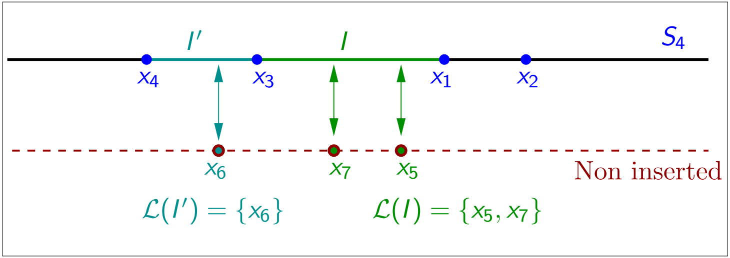
Conflict

- A non-inserted number x_j, $j > i$ *conflicts* with interval I if $x_j \in I$.
- Each non inserted x_j, $j > i$ has a pointer to its conflicting interval.



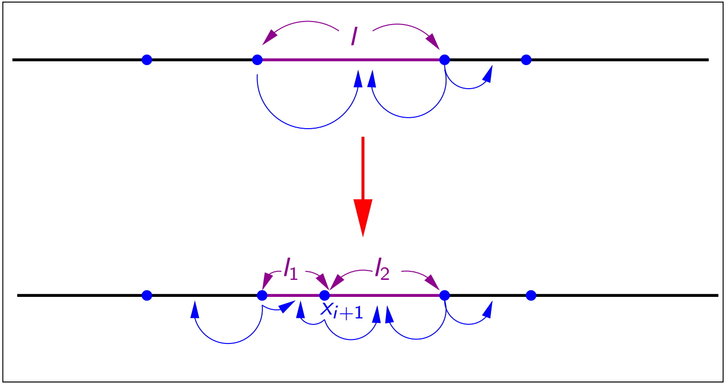
Conflict List

- Conversely, each interval I is associated with an unordered list $\mathcal{L}(I)$ of all its conflicting numbers x_j, $j > i$.
- These lists are called *conflict lists*.



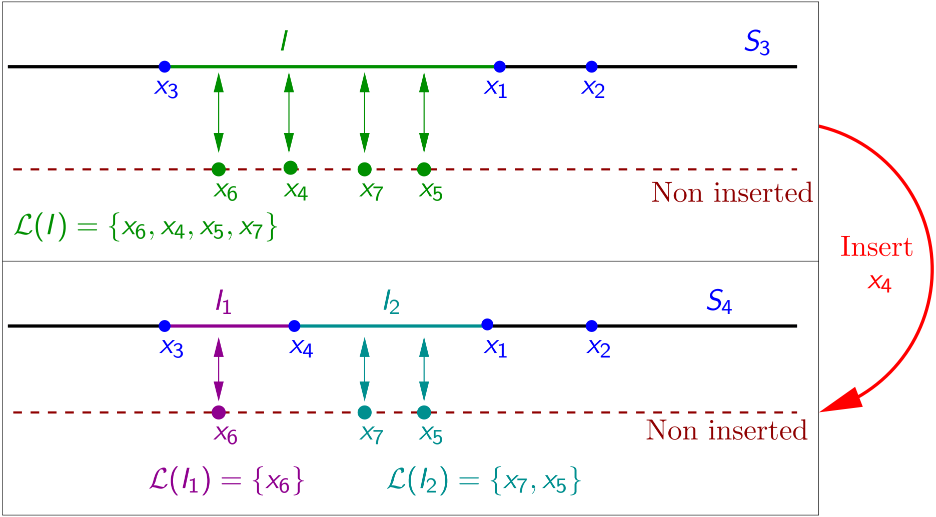
Inserting x_{i+1}

- Find the interval I that contains x_{i+1} by following the pointer at x_{i+1}.
- I is split into I_1 and I_2 by x_{i+1}.



- Update the adjacency information: $O(1)$ time.

Inserting x_{i+1}



Inserting x_{i+1}

Build $\mathcal{L}(I_1)$ and $\mathcal{L}(I_2)$:

- Traverse $\mathcal{L}(I)$, for each element check if it lies in I_1 or I_2, and insert it in the appropriate conflict list.
- It takes $O(|\mathcal{L}(I)|)$ time.
- At the same time, we update the pointer from each x_j in $\mathcal{L}(I)$ to the new conflicting interval I_1 or I_2.

Analysis

- Let I be the interval in $\mathbb{R} \setminus S_{i-1}$ that contains x_i.
- We denote $k_i = |\mathcal{L}(I)|$.
- In other words, k_i is the number of elements of $S \setminus S_{i-1}$ that lie in the segment that we break while inserting x_i.
- Let $T(n)$ denote the running time of our algorithm.
- Then

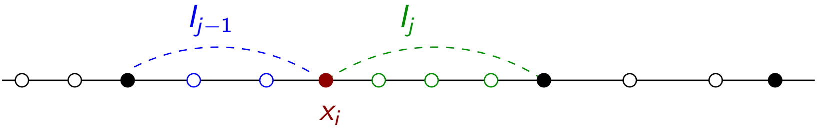
$$T(n) = O\left(\sum_{i=1}^{n} k_i\right).$$

- By linearity of expectation

$$E[T(n)] = O\left(\sum_{i=1}^{n} E[k_i]\right).$$

Backwards Analysis

- We need to bound $E[k_i]$.
- Remember that this expectation is the average over all the possible permutations of S.
- We use *backwards analysis*.
  - We assume that S_i is fixed, but not the order in which its elements have been inserted.
  - Then x_i is an element of S_i chosen uniformly at random.
- Let I_{j-1} and I_j be the two intervals adjacent to x_i.



- Then $k_i = |\mathcal{L}(I_{j-1})| + |\mathcal{L}(I_j)| + 1$.

Backwards Analysis

- Let $I_0, I_1 \dots I_i$ be the i intervals defined by S_i.
- Then the expected value of k_i is

$$E[k_i] = \frac{1}{i} \sum_{j=1}^{i} |\mathcal{L}(I_{j-1})| + |\mathcal{L}(I_j)| + 1.$$

- So

$$E[k_i] \leqslant 1 + \frac{2}{i} \sum_{j=0}^{i} |\mathcal{L}(I_j)|.$$

- Finally

$$E[k_i] \leqslant 1 + \frac{2(n-i)}{i} < \frac{2n}{i}.$$

Backwards Analysis

- We proved that, if S_i is fixed, then

$$E[k_i] < \frac{2n}{i}.$$

- Since it holds for any fixed S_i, then it also holds when we don't assume that S_i is fixed. (See Slide 22 for more details.)
- So

$$E[T(n)] = O\left(\sum_{i=1}^{n} E[k_i]\right) = O\left(\sum_{i=1}^{n} \frac{2n}{i}\right).$$

$$E[T(n)] = O\left(n \sum_{i=1}^{n} \frac{1}{i}\right) = O(n \log n).$$

Backwards Analysis: Example

- Let $S = \{1, 2, 3, 4, 5, 6, 7\}$.
- First case: assume $S_3 = \{2, 4, 7\}$.
  - So (x_1, x_2, x_3) is a random permutation of $\{2, 4, 7\}$, and $(x_4 \dots x_7)$ is a random permutation of $\{1, 3, 5, 6\}$.
  - What is $E[k_3]$ under this assumption?
  - Three possibilities are equally likely: $x_3 = 2$, or $x_3 = 4$ or $x_3 = 7$.
    - ★ If $x_3 = 2$, then $k_3 = 3$.
    - ★ If $x_3 = 4$, then $k_3 = 4$.
    - ★ If $x_3 = 7$, then $k_3 = 3$
  - So

$$
\begin{aligned}
E[k_3] &= \frac{10}{3} \\
&\leqslant 1 + \frac{2(n-i)}{i} \\
&= \frac{11}{3}.
\end{aligned}
$$

Backwards Analysis: Example

- We still consider $S = \{1, 2, 3, 4, 5, 6, 7\}$.
- Second case: assume $S_3 = \{1, 3, 6\}$.
  - So (x_1, x_2, x_3) is a random permutation of $\{1, 3, 6\}$, and $(x_4 \dots x_7)$ is a random permutation of $\{2, 4, 5, 7\}$.
  - What is $E[k_3]$ under this assumption?
  - Three possibilities are equally likely: $x_3 = 1$, or $x_3 = 3$ or $x_3 = 6$.
    - if $x_3 = 1$ then $k_3 = 2$
    - if $x_3 = 3$ then $k_3 = 4$
    - if $x_3 = 6$ then $k_3 = 4$
  - So

$$
\begin{aligned}
E[k_3] &= \frac{10}{3} \\
&\leqslant 1 + \frac{2(n-i)}{i} \\
&= \frac{11}{3}.
\end{aligned}
$$

Backwards Analysis: Example

- We still consider $S = \{1, 2, 3, 4, 5, 6, 7\}$
- For any particular choice of S_3 I will find that

$$
\begin{aligned}
E[k_3] &\leqslant 1 + \frac{2(n-i)}{i} \\
&= \frac{11}{3}.
\end{aligned}
$$

- So it is true in general (without specifying S_3) that

$$
\begin{aligned}
E[k_3] &\leqslant 1 + \frac{2(n-i)}{i} \\
&= \frac{11}{3}.
\end{aligned}
$$

Backwards Analysis: More Detailed Proof

In Slide 18, we showed that for any $F \subset S$ such that $|F| = i$, we have

$$E[k_i \mid S_i = F] < \frac{2n}{i}.$$

We can rewrite it

$$\sum_k k.P[k_i = k \mid S_i = F] < \frac{2n}{i},$$

and thus

$$\sum_k k \frac{P[k_i = k, S_i = F]}{P[S_i = F]} < \frac{2n}{i}.$$

Backwards Analysis: More Detailed Proof

The event $S_i = F$ has probability $1/\binom{n}{i}$, therefore

$$\binom{n}{i} \sum_k k.P[k_i = k, S_i = F] < \frac{2n}{i}.$$

Summing up over all $F \subset S$ such that $|F| = i$, we get

$$\binom{n}{i} \sum_k k \sum_{F \subset S, |F|=i} P[k_i = k, S_i = F] < \binom{n}{i} \frac{2n}{i},$$

so after cancelling the $\binom{n}{i}$ factors, we obtain

$$\sum_k k \sum_{F \subset S, |F|=i} P[k_i = k, S_i = F] < \frac{2n}{i}.$$

Backwards Analysis: More Detailed Proof

As the events $S_i = F$ are mutually exclusive, we can rewrite it

$$\sum_k k.P[k_i = k] < \frac{2n}{i},$$

and thus

$$E[k_i] < \frac{2n}{i}.$$

Concluding Remarks

The expected running time is $O(n \log n)$.

- It works on any input.
- So this is an expected running time on worst case input.
- Expectation is over the random choices of the algorithm.

On the other hand, deterministic QUICKSORT (with fixed pivot):

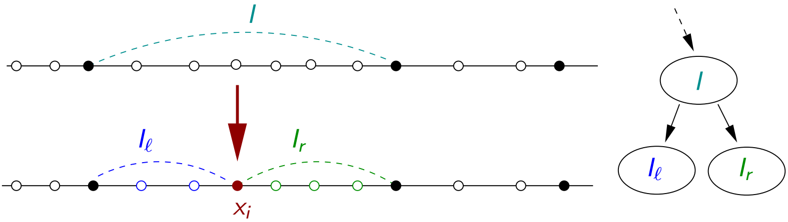
- Sorts random input in expected $O(n \log n)$ time.
- But is quadratic on worst case input.

Motivation

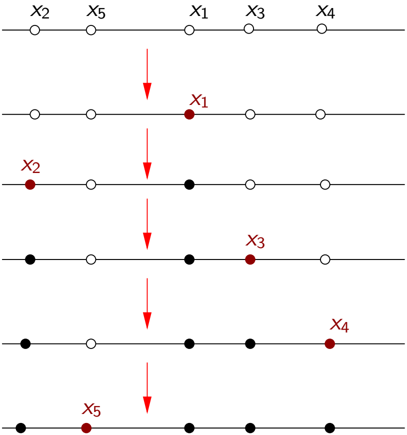
- We solved sorting using RIC.
- Is there a related query problem?
- Yes, searching.
- Formulation:
  - S defines a partition of $\mathbb{R}$ into intervals.
  - Given a query number q, which interval does it belong to?
  - General position assumption: $q \notin S$.
- Can be solved simply by binary search.
- Here we give a method that can be generalized to solve more difficult problems. (See next lecture).

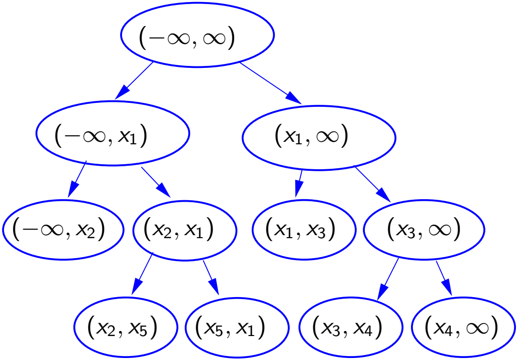
Idea

- During the RIC, after we split an interval, we forget it.
- Instead of forgetting this information, we will record it in a *history graph*.
- In the case of QUICKSORT, this graph is a binary tree.
- So if we split I into I_ℓ and I_r:
  - Make two nodes containing I_ℓ and I_r respectively.
  - Draw edges (I, I_ℓ) and (I, I_r).



Example





History graph

Answering queries

- q denotes the query number.
- Start at the root.
  - If the current node is a leaf, we are done.
  - Otherwise, recurse on the child whose interval contains q.
- We have just build a randomized binary search tree.
- Query time?
  - Worst case: $\Omega(n)$.
  - Best case: $O(1)$.
  - Average: $O(\log n)$. (See next slides.)

Analysis

- Expectation will be over the random choice of the permutation of S.
- We consider that q is fixed and the data structure is random.
- We denote by J_i the interval defined by S_i that contains q.
- That is, the interval that contains q at the i–th step of the RIC.
- The number of different intervals J_i gives the number of nodes in the search path, hence the query time.
- What is the probability that $J_i \neq J_{i-1}$?
  - It is the probability that J_{i-1} is split while inserting x_i.

Backwards Analysis

- We use backwards analysis: S_i is considered as fixed.
- The probability that J_{i-1} is split by x_i is the probability that x_i is an endpoint of J_i.
- J_i has only two endpoints, so this probability is $\frac{2}{i}$.
  - Except for first and last interval $(-\infty, x_k)$ and (x_ℓ, ∞) where it is $\frac{1}{i}$.
  - So when S_i is fixed the probability is $\leqslant \frac{2}{i}$.
- So it is also true in general, when we don't impose that S_i is fixed.
- Then the expected query time is

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

Concluding Remarks

- We have shown that, for a fixed query point, the search path has expected length $O(\log n)$.
- It is also true that the expected length of the longest search path is $O(\log n)$, and that it holds with high probability.
- In other words the height of a random BST is $O(\log n)$, like a balanced BST.
- Problem: we cannot insert or delete in a random BST and keep this bound. Why?
  - In this case, the insertion order is not random.
  - So insertion can take $\Omega(n)$ time and height can be $\Omega(n)$.