Note Title 12-06-2019

Given a set of points $P = \{p_1, p_2 \cdots p_n\}$

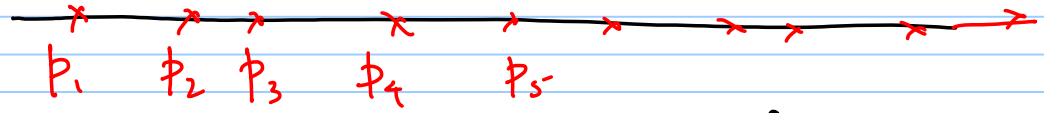
find the closest pair among $P$ $\quad \underset{i,j \in P}{\operatorname{argmin}}\ dist(p_i, p_j)$

Naive approaches Try all possible pairs $1 \le i, j \le n \Rightarrow \binom{n}{2}$ distance computation

$O(n^2)$

dist function can be computed in $O(1)$ time

Can we do better




(points are NOT given in sorted order)

Number of candidate pairs are $(p_i, p_{i+1})$ where $p_i$ are sorted

So first sort points and then try $n-1$ consecutive pairs

Time complexity is dominated by sorting : $O(n \log n)$

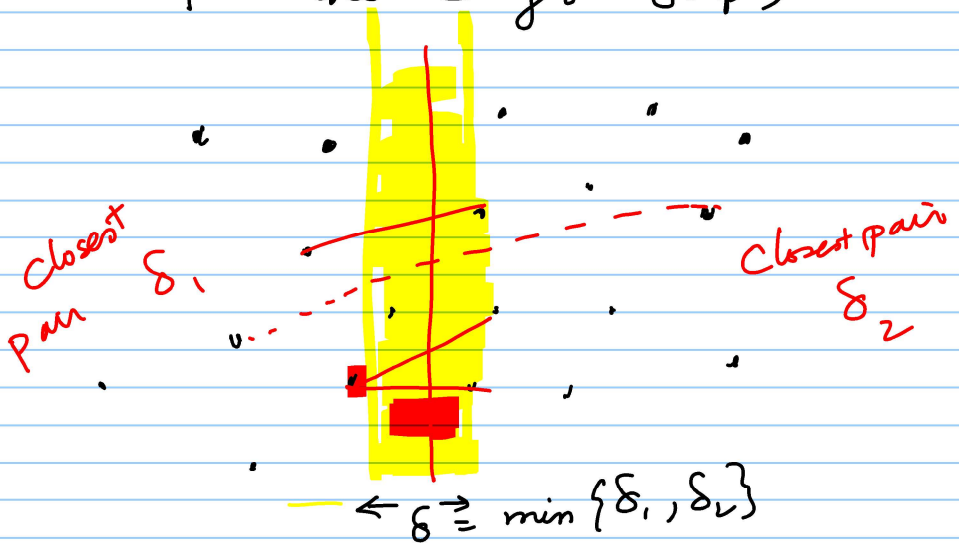
Bad News : $\Omega(n \log n)$ is a provable lower bound for Element Uniqueness / Distinctness

Given $n$ elements $x_1, x_2 \cdots x_n$ are all points distinct, $x_i \neq x_j \; \forall \; i, j$

Since Element Distinctness $\propto$ Closest pair, the lower bound also applies to closest pair

(Closest pair for non-distinct pair is 0 distance)

(Note: There is a simple $O(n)$ algorithm for the largest Gap)

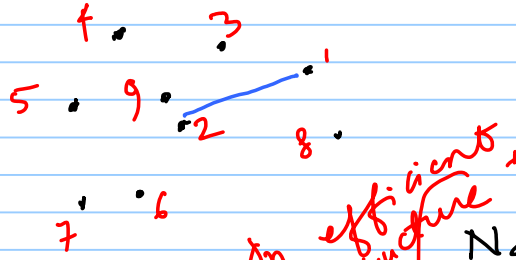



Divide and Conquer algorithm yields an $O(n \log n)$

# Another method to compute Closest pair

Given: $P$

We "add" the points in some order and maintain the closest pair among the points added till now.




$P_j$ = the subset of the first $j$ points

$\delta_j$ = closest pair distance within $P_j$

for $2 \le j \le n$

Naive analysis : when we add the point $p_{j+1}$ to $P_j$ $P_{j+1} = P_j \cup \{p_{j+1}\}$

we can check all the distances $dist(p_{j+1}, q)$ $q \in P_j$ and compare with $\delta_j$

If any of the newly computed distances is smaller then update $\delta_j$ to $\delta_{j+1}$

An efficient data structure to compute $\delta_{j+1}$

Running time:

$$\sum_{j=3}^{n} O(j) = O(n^2)$$

The "twist" to the previous Incremental Algorithm

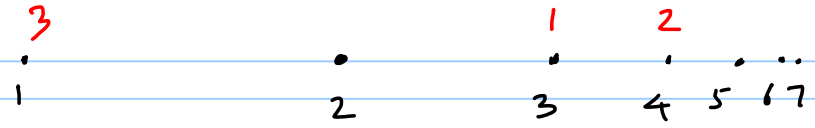
We will add the points in a "random" order

Random order in this context is a random permutation of $\{1, 2, \ldots n\}$ : each permutation is equally likely

1. We can start the algorithm by generating a random permutation $(\tilde{p}_1, \tilde{p}_2, \tilde{p}_3 \cdots \tilde{p}_n)$ and follow this order for insertion
2. Among the remaining points in stage $j$ $P_j$ we pick a point uniformly at random from $P - P_j$

This results in a randomized algorithm and the framework is called Randomized Incremental Algorithm (RIC)

1. There is no assumption on the input set of points : worst case (input points are not chosen randomly)
2. The running would depend on the order of insertion



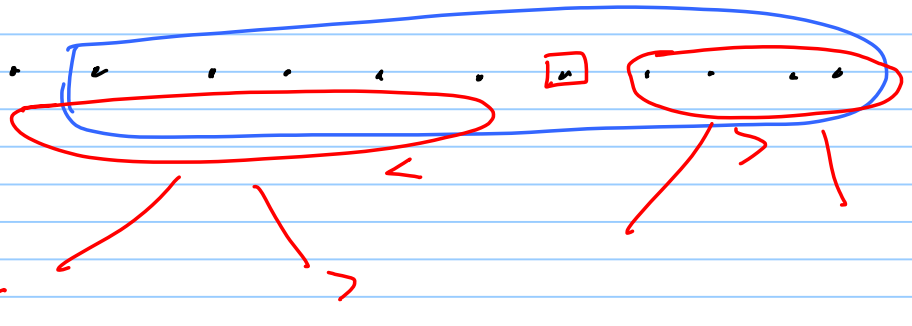

Obs: Running-time $T(n)$ will depend on the permutation

$T^{\sigma}(n)$ : is some fixed quantity if $\sigma$ is fixed

$\sigma \in \Pi(n)$ : all perm.

We want to analyse the quantity $\frac{1}{n!} \sum_{\sigma \in \Pi(n)} T^{\sigma}(n)$ : Expected running time

Recall Quicksort



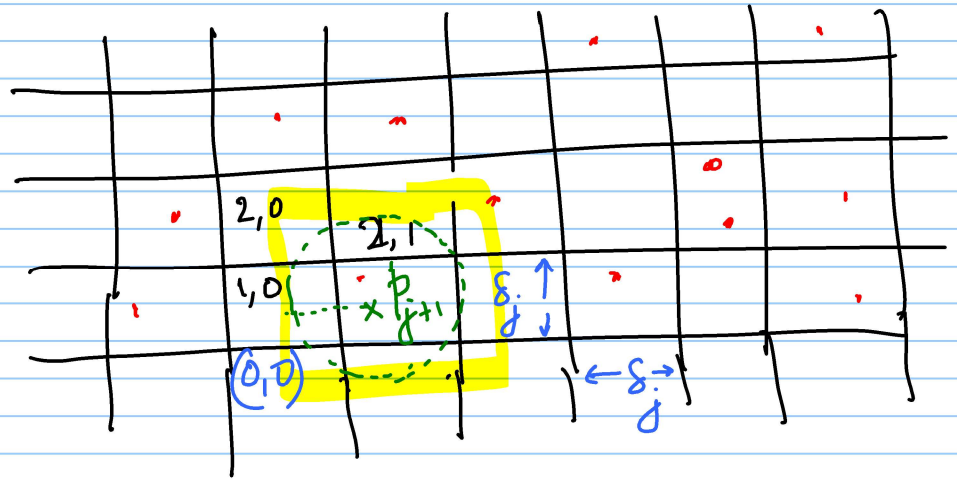
Randomized Quicksort

The sequence of pivots can be thought of a random permutation

The expected running-time of quicksort is averaging over all sequences of pivots

# A data structure to recalculate $\delta_{j+1}$

Inductive we have computed $\delta_j$ for $P_j$ (the first $j$ random points)




the grid cells can be indexed by $s, k$
$s, k$ are integers

The data structure will only store the non-empty cells : at most $j$

1. We can easily compute the cell containing $p_{j+1}$ using $(x_{j+1}, y_{j+1})$ and normalize w.r.t. $\delta_j$

Data structure should support

1. Finding non-empty neighboring cells
   Can be done in $O(\log n)$ time using BST
   or in $O(1)$ using universal hashing

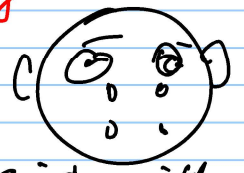
**Observation**: In $O(\log n)$ time, we can verify if

$$\delta_{j+1} = \delta_j \quad \text{or} \quad \delta_{j+1} \neq \delta_j$$

Good case

No change in request

So the new point is simply stored in the data structure : $O(\log n)$ time



Recompute the entire grid with $\delta_{j+1}$ and store cells

Time $O(j \log j) \leq O(j \log n)$

$\sum_{j=1}^{n} j \log n \sim O(n^2 \log n)$

In the good case, total bound is $O(n \log n)$

What is the probability of Case II (disaster)

What is the expected cost of the $j+1$ insertion

Suppose the probability of incurring Case II is $q_{j+1}$

$\Rightarrow$ Expected cost of $j^{th}$ step is $j \log n \times q_{j+1} + (1 - q_{j+1}) \log n$

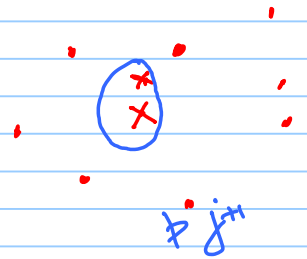
ignore since $\log n$ cost will occur anyways

with $q_{j+1} = \frac{2}{j+1}$ we get expected cost as $\leq j/j+1 \cdot \log n$ $\leq j \log n \cdot q_{j+1} + \log n$

Consider a fixed subset of $j+1$ points $Q_{j+1}$

Note: When we are generating a random permutation all initial $j+1$ element subsets are equally likely (from symmetry of random perm)

$Q_{j+1}$



Within $Q_{j+1}$ all orderings are equally likely

Suppose the closest pair of points is $(q_a, q_b) \in Q_{j+1}$

Obs: The closest pair distance $\delta_{j+1} \neq \delta_j$ iff $q_a$ or $q_b$ is the $j+1^{st}$ point

This occurs with probability $q_{j+1} = \frac{2}{j+1}$

This is conditioned on $Q_{j+1}$ being chosen as the subset. However this calculation holds for any choice of $Q_{j+1}$. Therefore unconditioned prob $= \frac{2}{j+1}$

$$\sum_{Q^{j+1}} \text{Prob}(Q_{j+1}) \cdot \frac{2}{j+1} = \frac{2}{j+1} \left( \sum_{Q_{j+1}} \text{Prob}(Q_{j+1}) \right) = 1$$

Total running-time of the algorithm $T_n$

$$T_n = \sum t_i$$ where $t_i$ is the cost of $i^{th}$ step

$$E[T_n] = E\left[\sum t_i\right] = \sum E[t_i]$$ using linearity of expectation

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

$E[T_n]$: all permutations being equally likely

Note: By using hashing based data structure, expected cost of $j^{th}$ step is $O(1) \Rightarrow O(n)$ overall

Caveat: The lower bound of $\Omega(n \log n)$ doesn't apply to algorithms that uses "truncation operation"

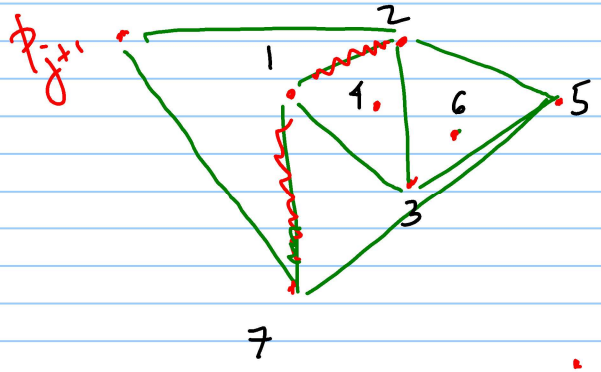
RIC is extremely versatile and extends to efficient/optimal algorithms for numerous geometric problem

Seriously influenced the entire field of Computational Geometry since mid 80's

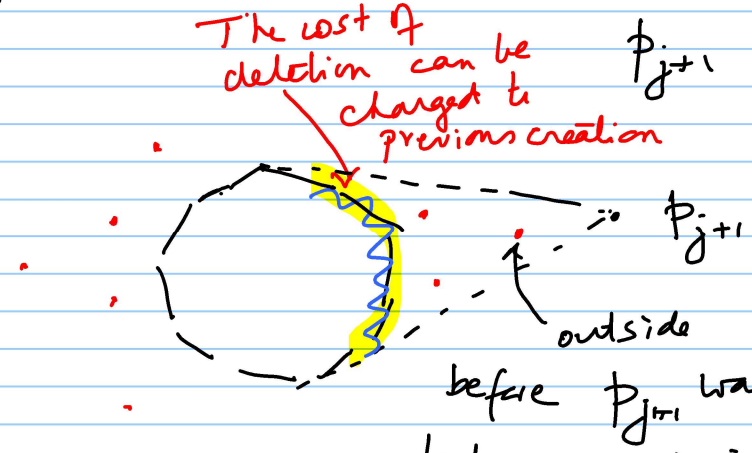
There is a generalized analysis of RIC that provides running times dependent on a few structural parameters of the respective problems

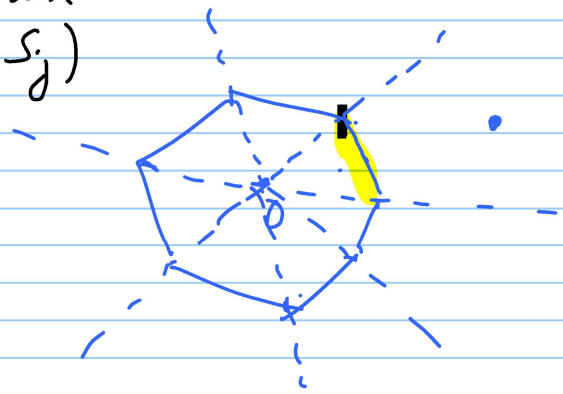
# Another example of RIC : planar hulls

Maintain convex hull of the currently added points




- Good case : the next point is inside the $CH(S_j)$ $S_j$ : the first $j$ points
- More difficult when $p_{j+1} \notin CH(S_j)$



How do we account for the cost of (unadded) points that move from outside to inside

Backward Analysis [Seidel]


As we walk from the intersection point of $l_{j+1}$ ($\nearrow p_{j+1}$) we will update all the other $l_i'$ that we come across

- either they disappear because the status changes or
- the new intersection with the newly inserted edges have to computed

So each update of the unadded points can be done in $O(1)$ time per point (i.e. for $k$ points, it is $O(k)$)

What is the probability that an unadded point incurs a cost in the $j+1^{st}$ step

$$I_k^j = \begin{cases} 1 & \text{if } k^{th} \text{ point incurs cost in } j^{th} \text{ addition} \\ 0 & \text{otherwise} \end{cases}$$

$$E[\text{Total cost of construction}] = E\left[\sum_k \sum_j I_k^j\right] \leq n \cdot E\left[\sum_j I_k^j\right] = n \cdot \sum_j E\left[I_k^j\right]$$

$$E[I_k^j] = \text{Prob}[I_k^j = 1] = \frac{2}{j+1}$$

$$\leq n \cdot \sum_j \frac{2}{j+1} \leq O(n \log n)$$