Space-Efficient Geometric Divide-and-Conquer Algorithms

Jan Vahrenhold

Department of Computer Science
WWU Münster

Joint work with:

Jit Bose, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid (Carleton)

Overview

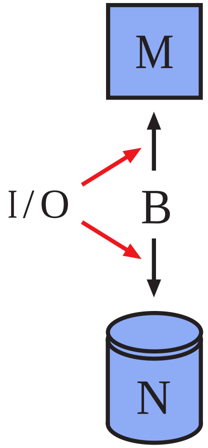
1. Introduction
2. Closest Pair & All Nearest Neighbors
3. Bichromatic Closest Pair
4. Conclusions

Does size really matter?

Cell Phone  $\longleftrightarrow$  Network Attached Storage

Theory (I/O-Model):

- Limited fast memory.



Practice:

- Limited fast memory, e.g. in car navigation systems.

Core Issue:

- Utilize (fast) memory in the best possible way, i.e., use as little memory as possible. (put aside compression...)

Definition 1.1

An algorithm A is called in-place iff during its execution A occupies $\mathcal{O}(\log_2 n)$ bits in addition to the space required by the input.

Consequences:

- Classic recursive algorithms are not in-place.
  - Need to maintain a call stack of size $\Omega(\log n)$ addresses, i.e., occupies $\Omega\left(\log^2 n\right)$ bits.
- Algorithms using auxiliary pointer-based data structures (such as balanced binary trees) are not in-place.
  - Need to resort to implicit data structures.

Example:

- Heapsort is an in-place algorithm.
  - Implicit data structure, needs $\mathcal{O}(1)$ indices of size $\mathcal{O}(\log n)$ bits each.

Sorting and Related Problems:

- Heapsort [Floyd, 1964].
- Stackless quicksort [Huang & Knuth, 1986, Wegner, 1987].
- Stable (multiset) sorting [Katajainen & Pasanen, 1994].
- Linear-time merging [Geffert et al., 2000].
- Linear-time stable partitioning [Katajainen & Pasanen, 1992].
- Linear-time k-selection [Carlsson & Sundström, 1995].

Computational Geometry:

- Planar Convex Hull [Brönnimann et al., 2002]:
  - $\mathcal{O}(n \log n)$ time (modification of [Graham, 1972]).
  - $\mathcal{O}(n \log h)$ time, h points on convex hull (modification of [Chan, 1996]).

Problems that can be solved in-place:

- Diameter of a Planar Point Set: $\mathcal{O}(n \log h)$ time.
- Convex Hull of a Simple Polygon: $\mathcal{O}(n)$ time.
- Minimum Enclosing Circle: $\mathcal{O}(n)$ expected time.

Problems that can be solved in-place:

- Diameter of a Planar Point Set: $\mathcal{O}(n \log h)$ time.
- Convex Hull of a Simple Polygon: $\mathcal{O}(n)$ time.
- Minimum Enclosing Circle: $\mathcal{O}(n)$ expected time.

Definition 1.2

An algorithm A is called *in situ* iff during its execution A occupies $\mathcal{O}\left(\log_2^2 n\right)$ bits in addition to the space required by the input.

Problems that can be solved in-place:

- Diameter of a Planar Point Set: $\mathcal{O}(n \log h)$ time.
- Convex Hull of a Simple Polygon: $\mathcal{O}(n)$ time.
- Minimum Enclosing Circle: $\mathcal{O}(n)$ expected time.

Definition 1.2
An algorithm A is called *in situ* iff during its execution A occupies $\mathcal{O}\left(\log_2^2 n\right)$ bits in addition to the space required by the input.

Powerful Tools:

- Implicit dictionaries [Munro, 1986]: $\mathcal{O}\left(\log_2^2 n\right)$ updates/queries.
- Recursion...

"In Situ" Geometry Results:

- Line Segment Intersection: $\mathcal{O}\left((n+k)\log_2^2 n\right)$ time and $\mathcal{O}\left(\log_2^2 n\right)$ extra bits [Chen & Chan, 2003].
- $3d$-convex hull and related: $\mathcal{O}\left(n\log_2^3 n\right)$ time and $\mathcal{O}\left(\log_2^2 n\right)$ extra bits [Brönnimann et al., 2004].

Data Structures:

- Dictionaries [Brodnik & Munro, 1999, Francheschini et al., 2002].
- Deque with random access [Brodnik et al., 1999].
- Dynamic arrays [Raman et al., 2001].
- . . . more results . . .

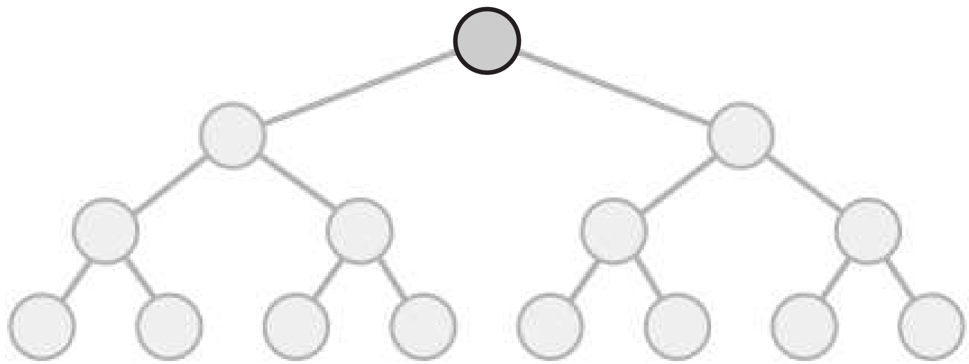
Overview

1. Introduction
2. Closest Pair & All Nearest Neighbors
3. Bichromatic Closest Pair
4. Conclusions

Basic Scheme:

1. Divide problem instance in two roughly equally sized parts.
2. Recurse on first subproblem.
3. Recurse on second subproblem.
4. Combine results.

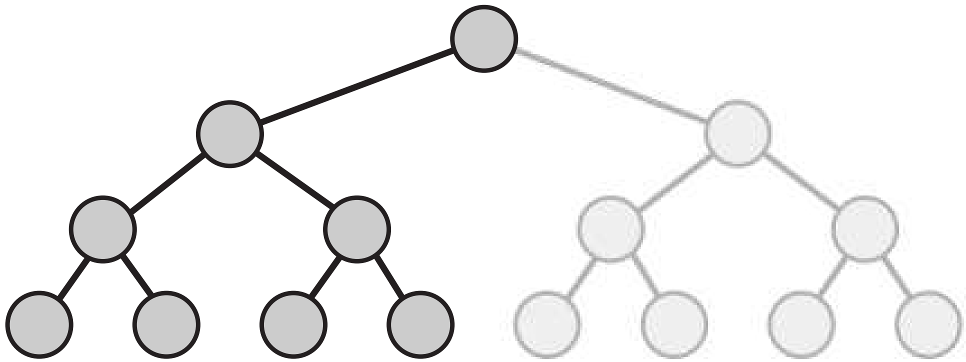
- Consider recursion tree induced by divide-and-conquer scheme.
- Recurse on left subtree, then recurse on right subtree.



Basic Scheme:

1. Divide problem instance in two roughly equally sized parts.
2. Recurse on first subproblem.
3. Recurse on second subproblem.
4. Combine results.

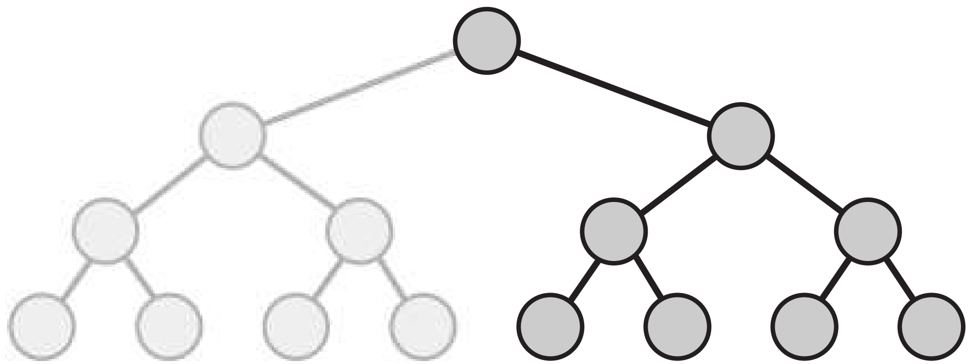
- Consider recursion tree induced by divide-and-conquer scheme.
- Recurse on left subtree, then recurse on right subtree.



Basic Scheme:

1. Divide problem instance in two roughly equally sized parts.
2. Recurse on first subproblem.
3. Recurse on second subproblem.
4. Combine results.

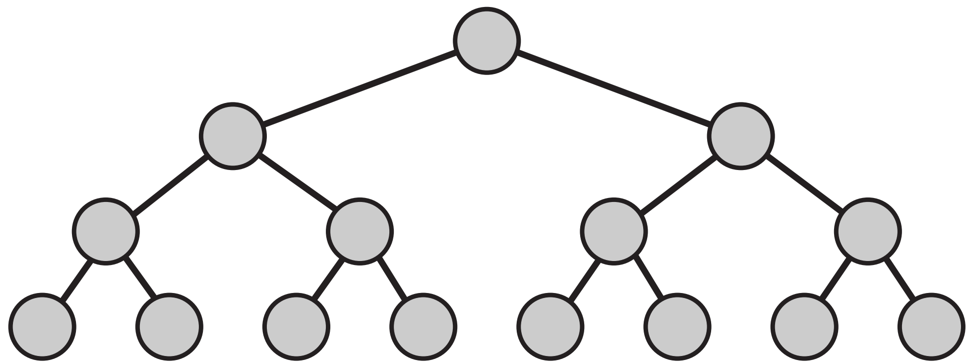
- Consider recursion tree induced by divide-and-conquer scheme.
- Recurse on left subtree, then recurse on right subtree.



Basic Scheme:

1. Divide problem instance in two roughly equally sized parts.
2. Recurse on first subproblem.
3. Recurse on second subproblem.
4. Combine results.

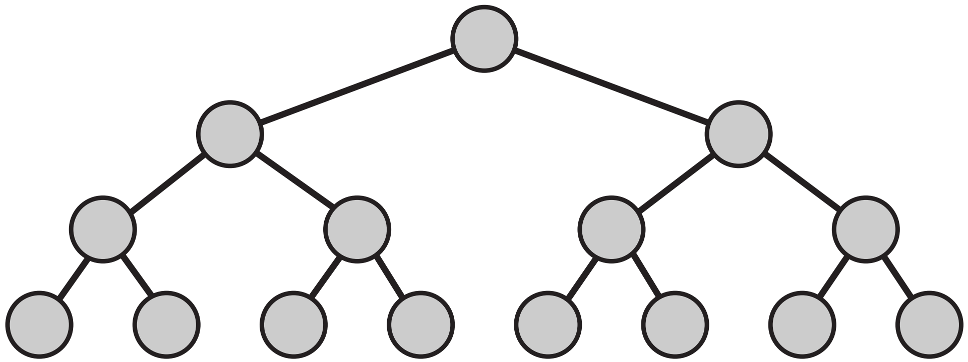
- Consider recursion tree induced by divide-and-conquer scheme.
- Recurse on left subtree, then recurse on right subtree.



Basic Scheme:

1. Divide problem instance in two roughly equally sized parts.
2. Recurse on first subproblem.
3. Recurse on second subproblem.
4. Combine results.

- Consider recursion tree induced by divide-and-conquer scheme.
- Recurse on left subtree, then recurse on right subtree.

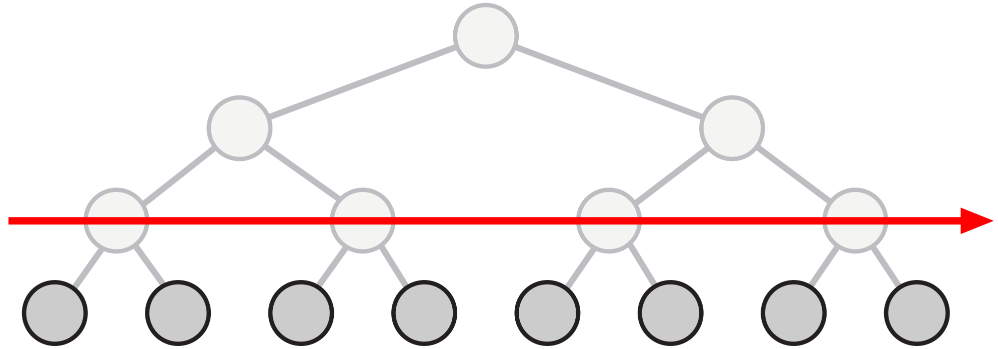


- In-place algorithm needs to traverse tree without recursion.

A First Approach: Bottom-Up

- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

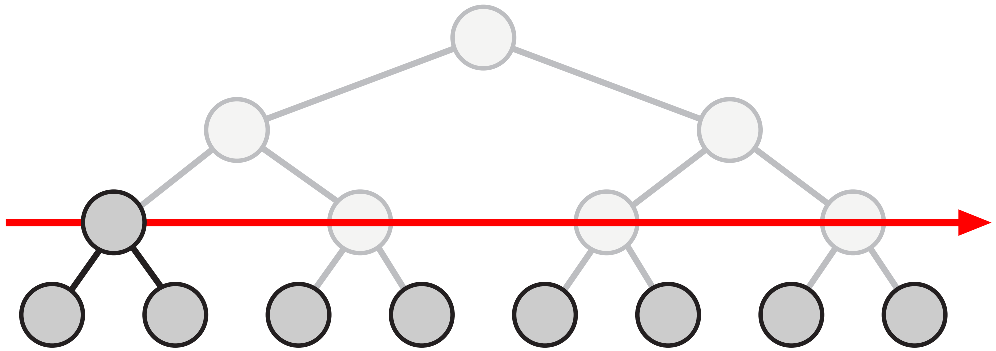
for level $= 1$ to $k - 1$ **do**
    width $:= 2^{\text{level}}$
    for $j = 1$ to $2^{k-\text{level}}$ step 2 **do**
        Combine elements in $A[(j-1) \cdot \text{width} + 1 \ldots (j+1) \cdot \text{width}]$.
    end for
end for



A First Approach: Bottom-Up

- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

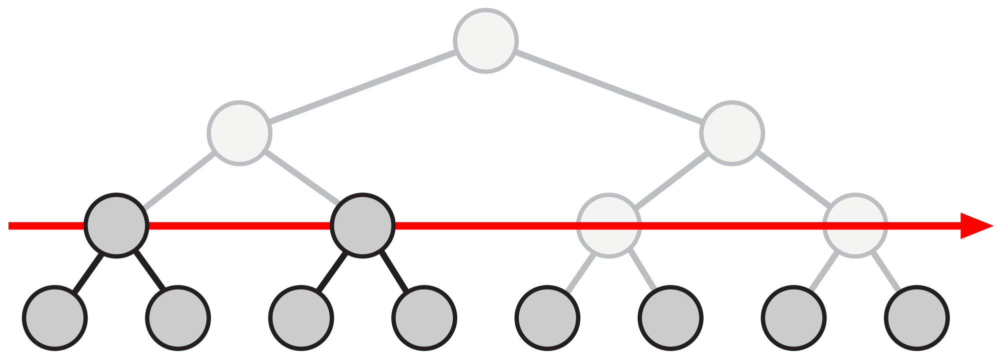
for level $= 1$ to $k - 1$ **do**
  width $:= 2^{\text{level}}$
  for $j = 1$ to $2^{k-\text{level}}$ step 2 **do**
    Combine elements in $A[(j-1)\cdot\text{width}+1 \ldots (j+1)\cdot\text{width}]$.
  end for
end for



A First Approach: Bottom-Up

- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

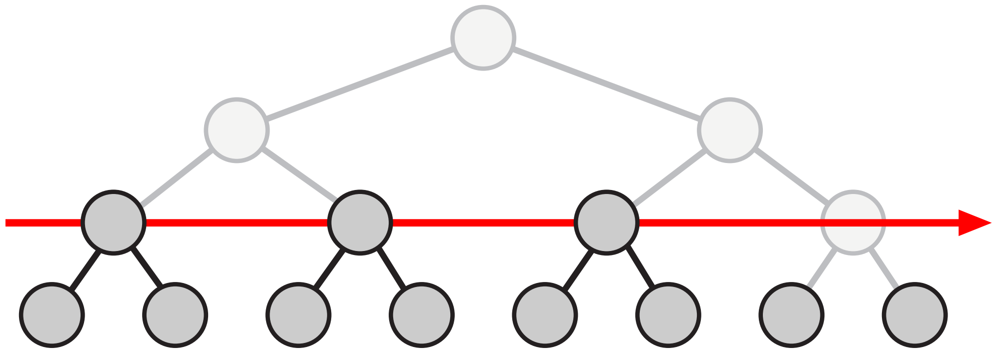
for level $=$ 1 to $k-1$ **do**
  width := 2^{level}
  for $j = 1$ to $2^{k-\text{level}}$ step 2 **do**
    Combine elements in $A[(j-1)\cdot\text{width}+1 \ldots (j+1)\cdot\text{width}]$.
  end for
end for



A First Approach: Bottom-Up

- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

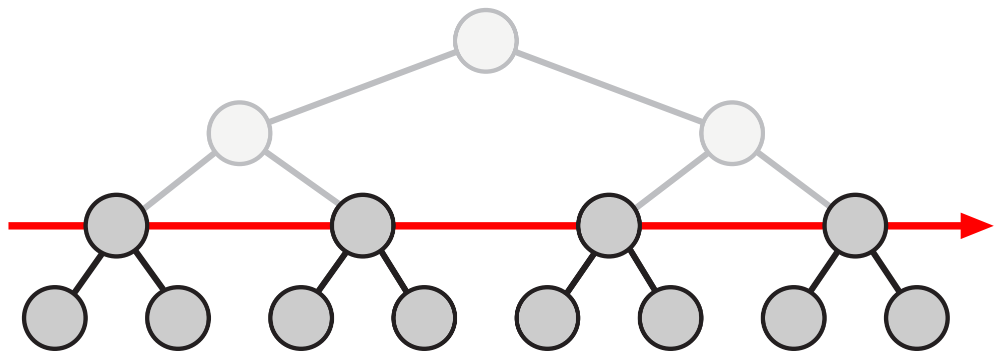
for level $=$ 1 to $k - 1$ **do**
  width $:= 2^{\text{level}}$
  for $j = 1$ to $2^{k-\text{level}}$ step 2 **do**
    Combine elements in $A[(j-1)\cdot\text{width}+1 \ldots (j+1)\cdot\text{width}]$.
  end for
end for



A First Approach: Bottom-Up

- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

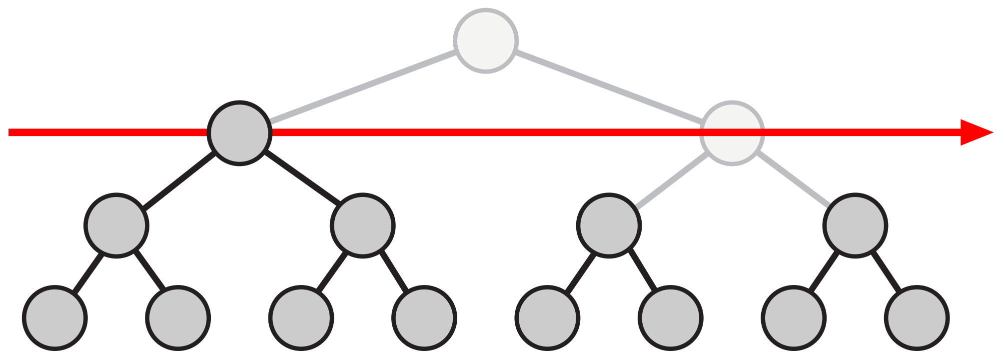
for level $= 1$ to $k - 1$ **do**
  width $:= 2^{\text{level}}$
  for $j = 1$ to $2^{k-\text{level}}$ step 2 **do**
    Combine elements in $A[(j-1)\cdot\text{width}+1 \ldots (j+1)\cdot\text{width}]$.
  end for
end for



A First Approach: Bottom-Up

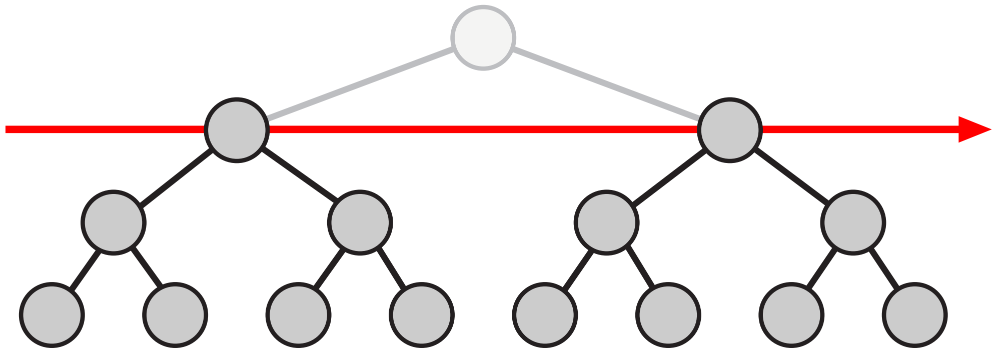
- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

for level $=$ 1 to $k-1$ **do**
    width $:= 2^{\text{level}}$
    for $j = 1$ to $2^{k-\text{level}}$ step 2 **do**
        Combine elements in $A[(j-1)\cdot\text{width}+1 \ldots (j+1)\cdot\text{width}]$.
    end for
end for



- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

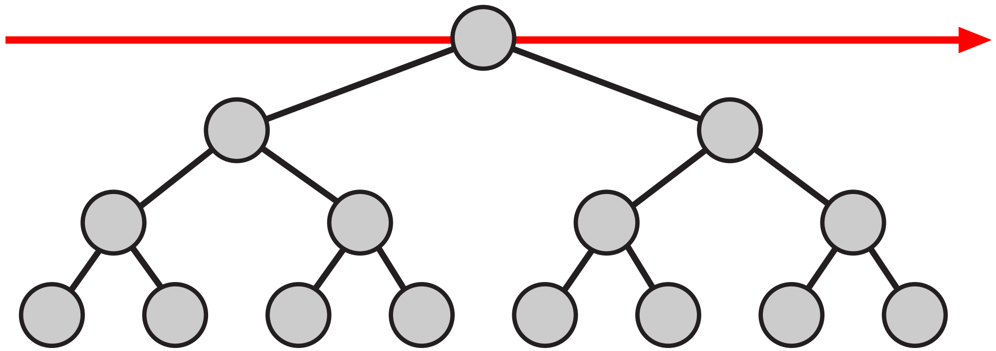
for level $=$ 1 to $k-1$ **do**
    width $:= 2^{\text{level}}$
    for $j = 1$ to $2^{k-\text{level}}$ step 2 **do**
        Combine elements in $A[(j-1)\cdot\text{width}+1 \ldots (j+1)\cdot\text{width}]$.
    end for
end for



A First Approach: Bottom-Up

- Assume w.l.o.g. $n = 2^k$ for some $k \in \mathbb{N}$.

```
for level = 1 to k − 1 do
    width := 2^level
    for j = 1 to 2^(k−level) step 2 do
        Combine elements in A[(j−1)·width+1 ... (j+1)·width].
    end for
end for
```



Two Drawbacks:

- Bad (memory) locality of data accesses ($\rightarrow$ cache efficiency?)
- Bad (spatial) locality of data accesses for geometric data.

Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

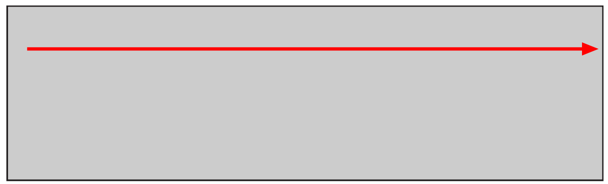
- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

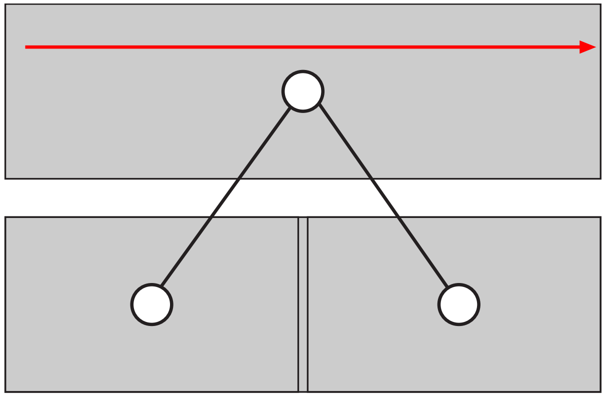


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

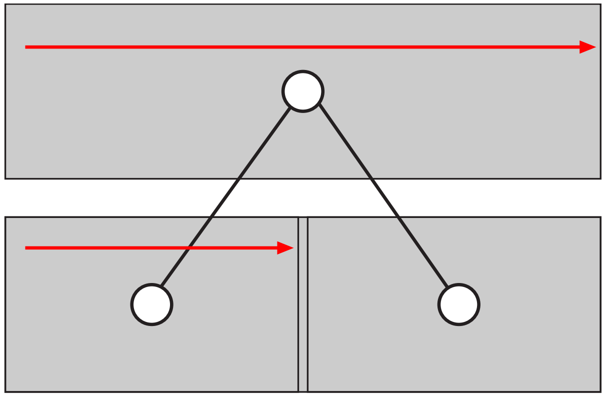


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

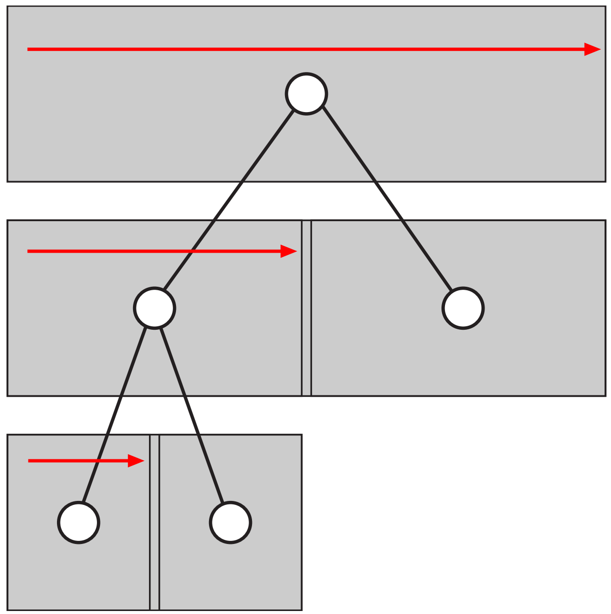


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

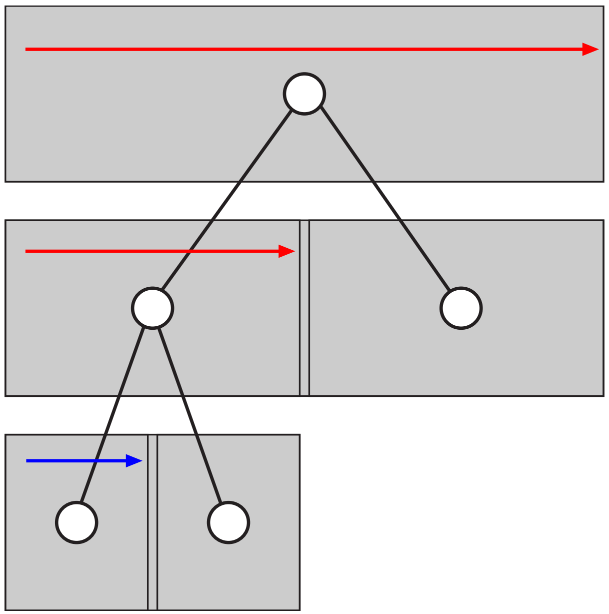


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

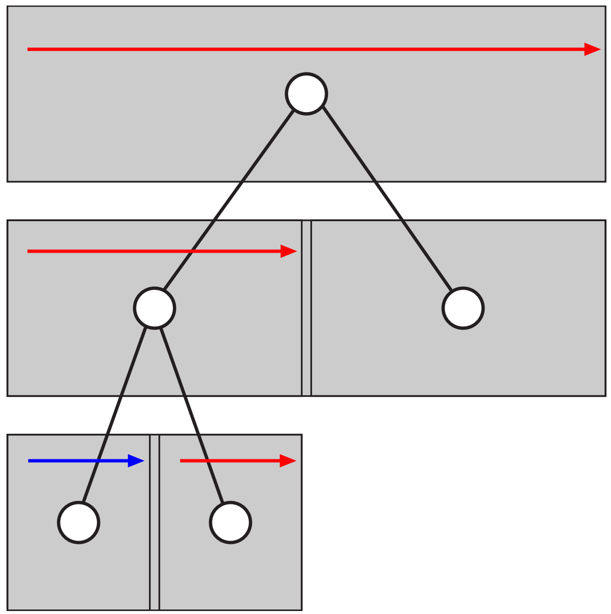


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

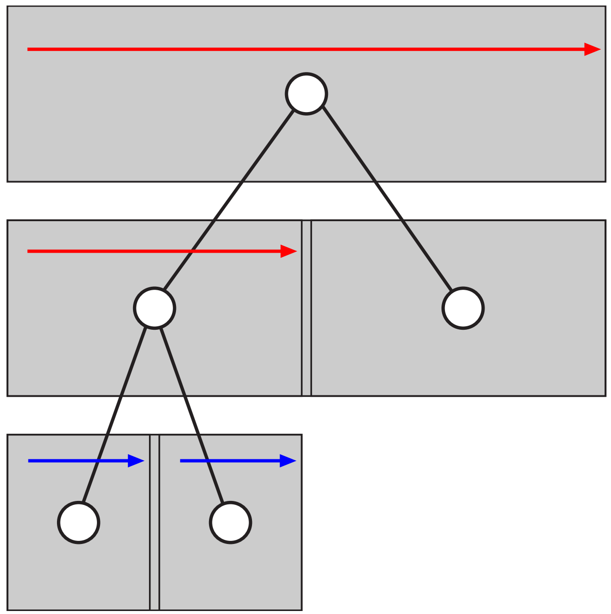


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

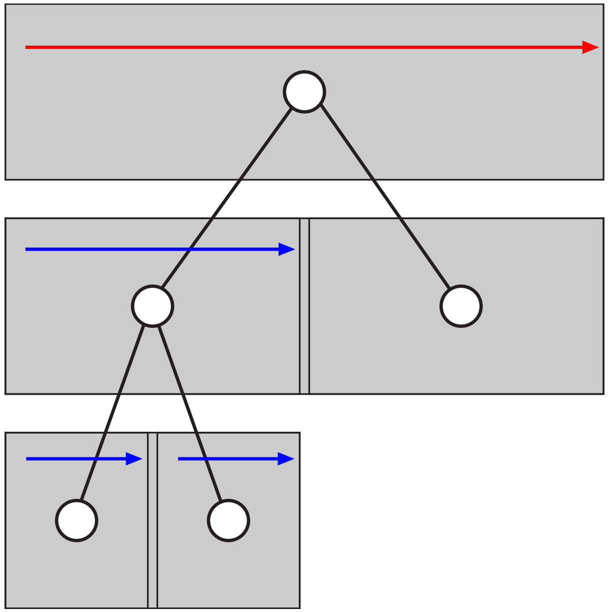


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

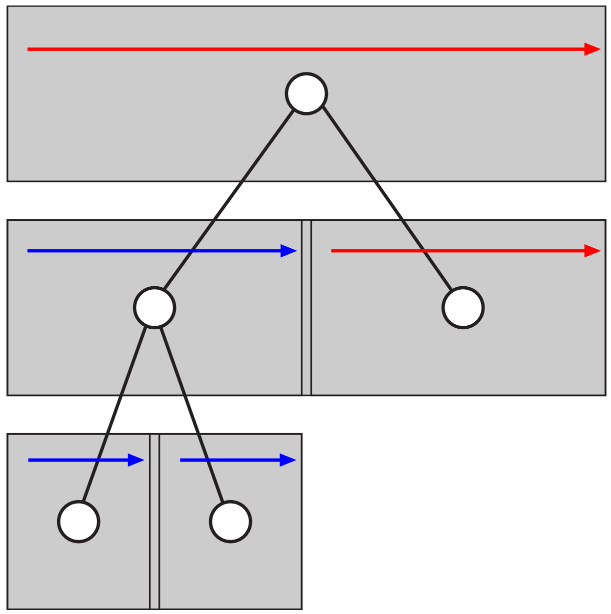


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

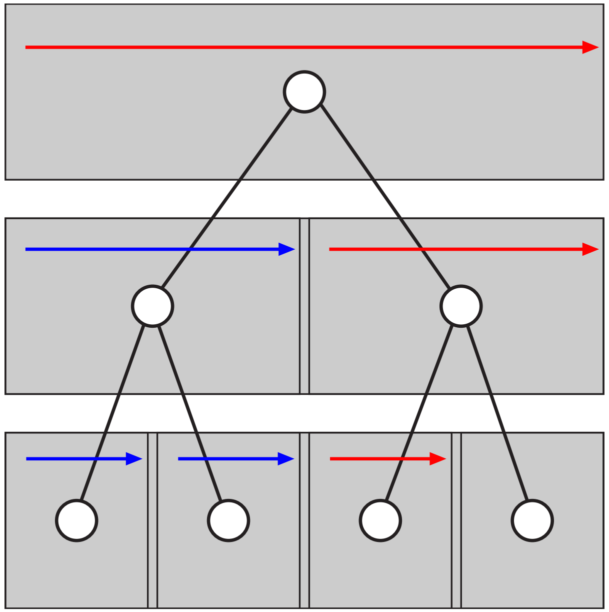


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

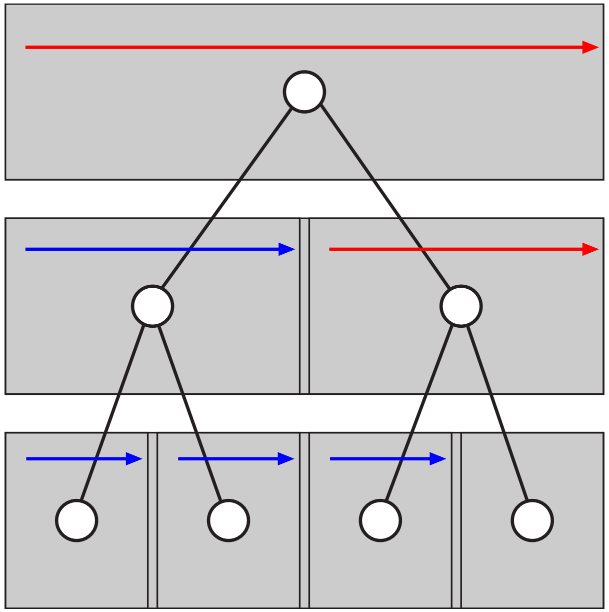


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

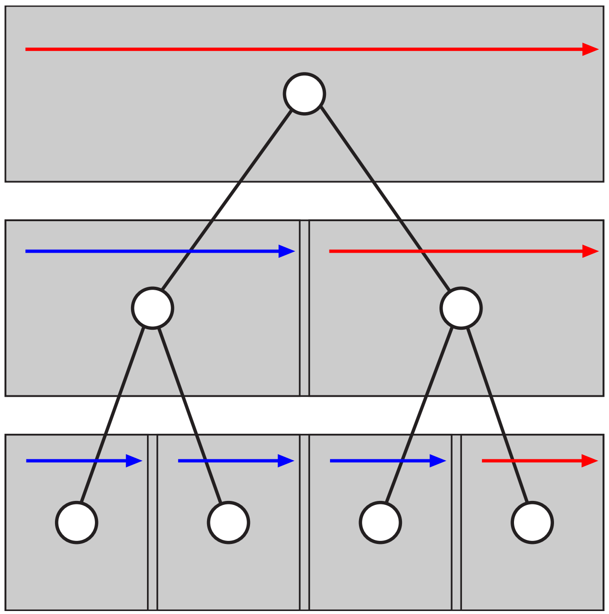


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

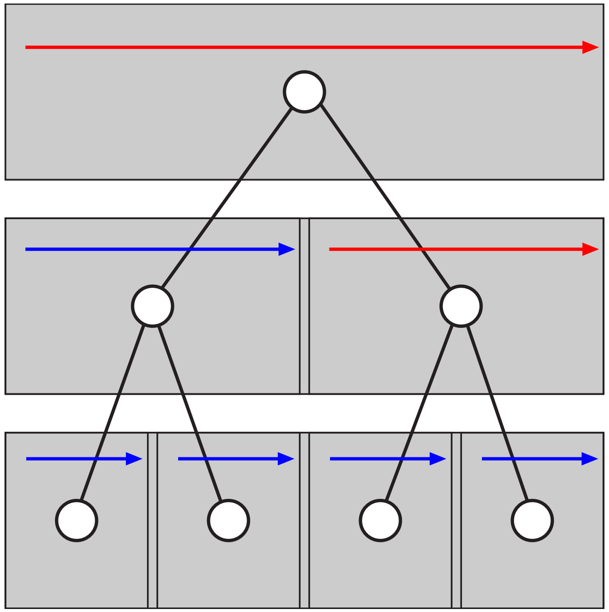


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

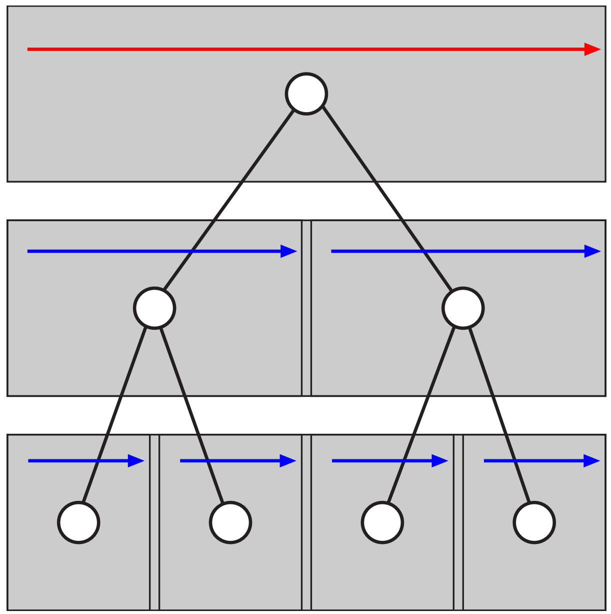


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

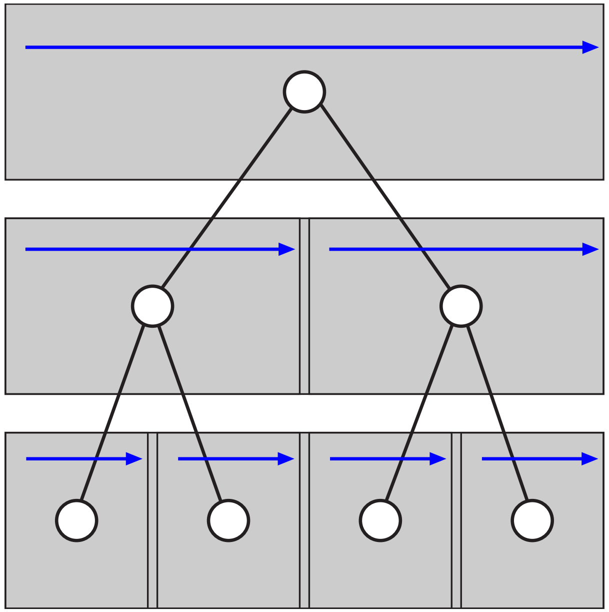


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.

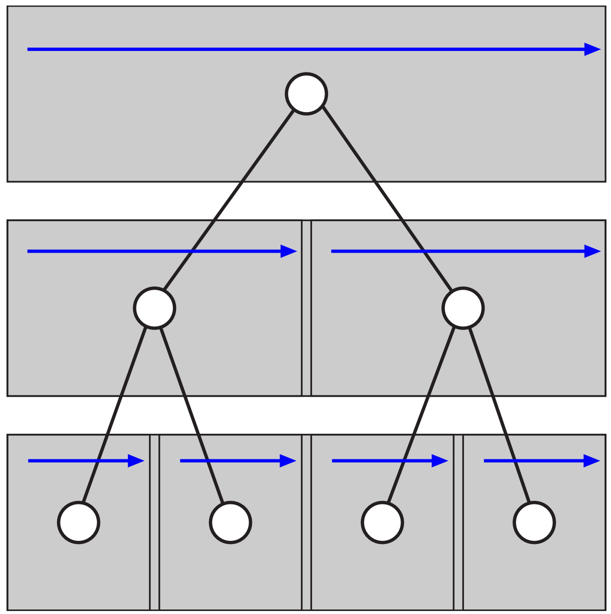


Example [Balaban, 1995]:

- Solve line segment intersection problem in optimal time and space.
- Combine divide-and-conquer and plane-sweeping technique.

Approach:

- Hierarchically subdivide plane in vertical slabs.
- Input: Segments crossing left slab boundary.
- Output: Segments crossing right slab boundary.
- Compute intersections while sweeping.



- Observation: Algorithm performs Euler tour of recursion tree.

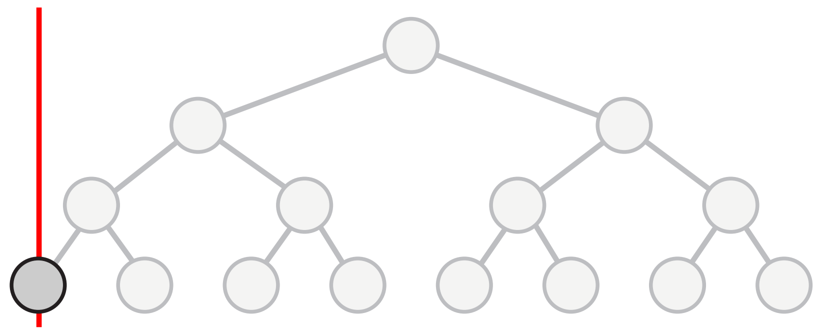
Space-Efficient Divide-and-Conquer

- Use “folklore” approach for Euler-tour like postorder traversal:

   Let $b = 0$ and $e = 1$.
   while $b \neq 0$ or $e \neq n$ **do**
      Let i be index of e's least significant bit (lowest index: 1).
      for $c := 1$ to $i - 1$ **do**
         Set $b := e - 2^c$.
         Merge two subarrays in $\mathtt{A}[b \ldots e-1]$.
      end for
      Set $e := e + 1$.
   end while

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



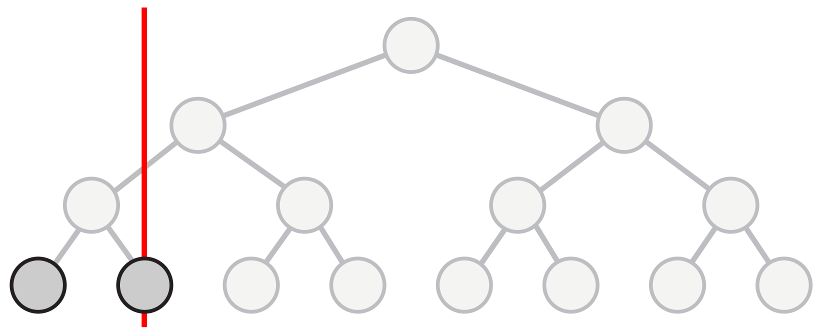
Space-Efficient Divide-and-Conquer

- Use “folklore” approach for Euler-tour like postorder traversal:

```
Let b = 0 and e = 1.
while b ≠ 0 or e ≠ n do
    Let i be index of e's least significant bit (lowest index: 1).
    for c := 1 to i − 1 do
        Set b := e − 2^c.
        Merge two subarrays in A[b . . . e − 1].
    end for
    Set e := e + 1.
end while
```

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.

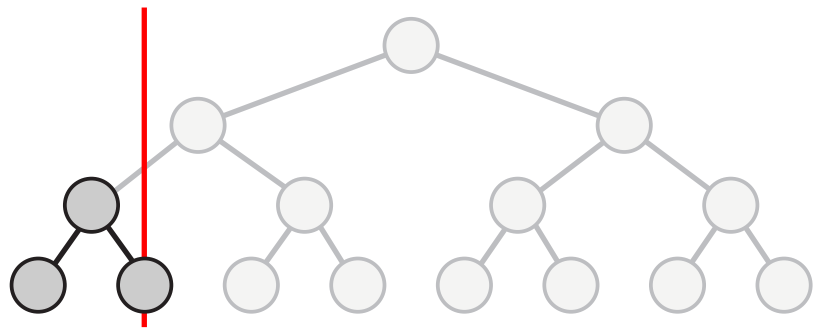


- Use “folklore” approach for Euler-tour like postorder traversal:

  Let $b = 0$ and $e = 1$.
  while $b \neq 0$ or $e \neq n$ **do**
      Let i be index of e’s least significant bit (lowest index: 1).
      for $c := 1$ to $i - 1$ **do**
          Set $b := e - 2^c$.
          Merge two subarrays in $\mathtt{A}[b \ldots e-1]$.
      end for
      Set $e := e + 1$.
  end while

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



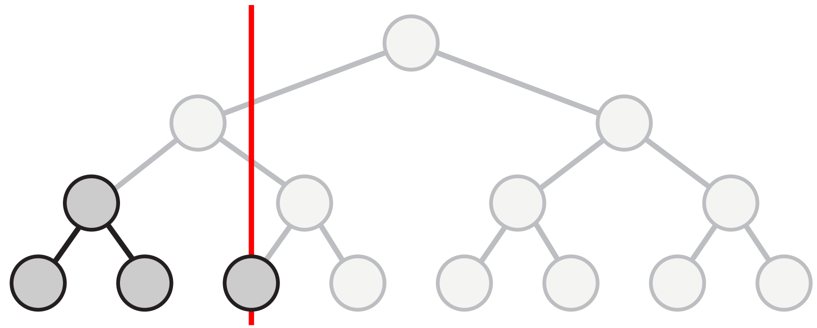
Space-Efficient Divide-and-Conquer

- Use “folklore” approach for Euler-tour like postorder traversal:

```
Let b = 0 and e = 1.
while b ≠ 0 or e ≠ n do
    Let i be index of e's least significant bit (lowest index: 1).
    for c := 1 to i − 1 do
        Set b := e − 2^c.
        Merge two subarrays in A[b...e − 1].
    end for
    Set e := e + 1.
end while
```

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



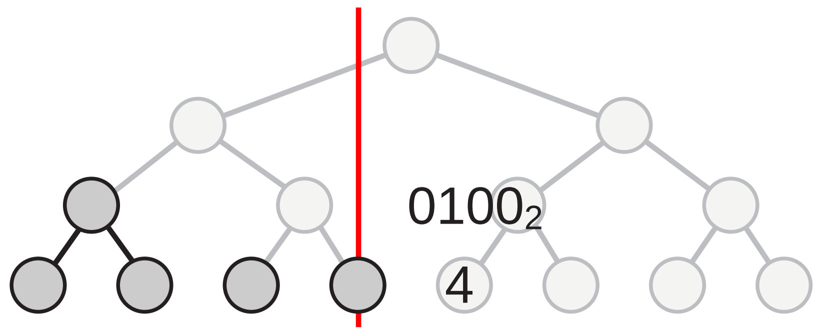
Space-Efficient Divide-and-Conquer

- Use "folklore" approach for Euler-tour like postorder traversal:

Let $b = 0$ and $e = 1$.
while $b \neq 0$ or $e \neq n$ **do**
    Let i be index of e's least significant bit (lowest index: 1).
    for $c := 1$ to $i - 1$ **do**
        Set $b := e - 2^c$.
        Merge two subarrays in $\texttt{A}[b \ldots e - 1]$.
    end for
    Set $e := e + 1$.
end while

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



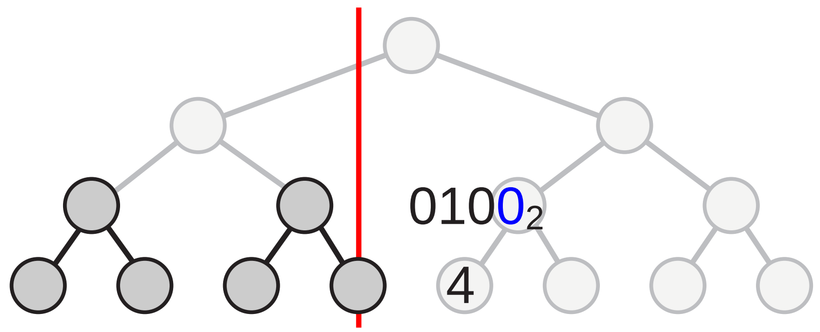
Space-Efficient Divide-and-Conquer

- Use "folklore" approach for Euler-tour like postorder traversal:

```
Let b = 0 and e = 1.
while b ≠ 0 or e ≠ n do
    Let i be index of e's least significant bit (lowest index: 1).
    for c := 1 to i − 1 do
        Set b := e − 2^c.
        Merge two subarrays in A[b ... e − 1].
    end for
    Set e := e + 1.
end while
```

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



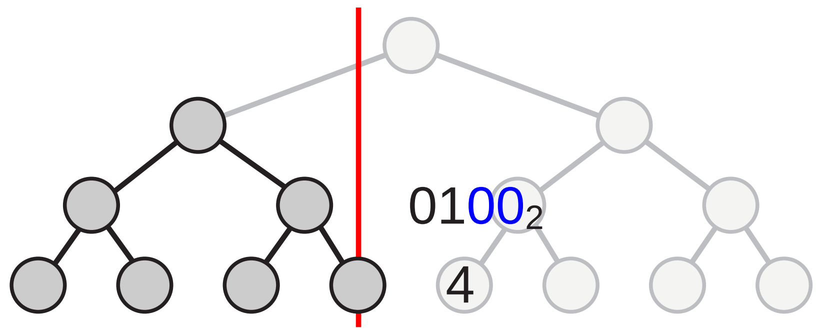
Space-Efficient Divide-and-Conquer

- Use "folklore" approach for Euler-tour like postorder traversal:

```
Let b = 0 and e = 1.
while b ≠ 0 or e ≠ n do
    Let i be index of e's least significant bit (lowest index: 1).
    for c := 1 to i − 1 do
        Set b := e − 2^c.
        Merge two subarrays in A[b ... e − 1].
    end for
    Set e := e + 1.
end while
```

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.

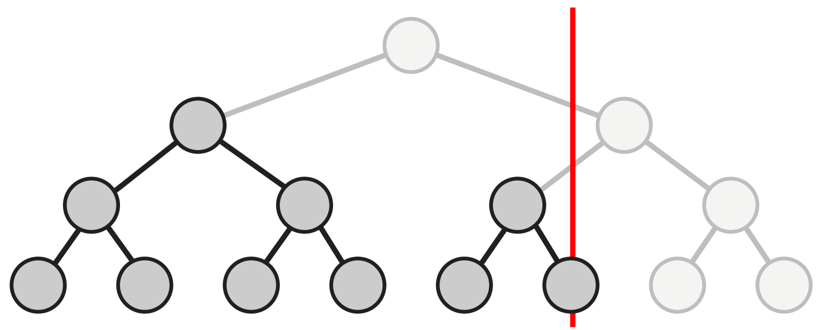


- Use "folklore" approach for Euler-tour like postorder traversal:

```
Let b = 0 and e = 1.
while b ≠ 0 or e ≠ n do
    Let i be index of e's least significant bit (lowest index: 1).
    for c := 1 to i − 1 do
        Set b := e − 2^c.
        Merge two subarrays in A[b ... e − 1].
    end for
    Set e := e + 1.
end while
```

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.

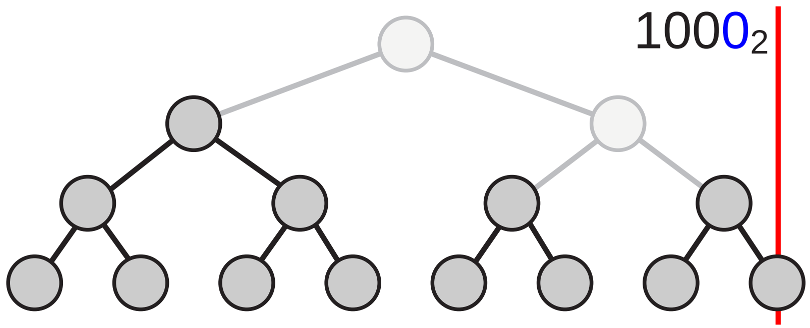


- Use “folklore” approach for Euler-tour like postorder traversal:

```
Let $b = 0$ and $e = 1$.
while $b \neq 0$ or $e \neq n$ do
    Let $i$ be index of $e$'s least significant bit (lowest index: 1).
    for $c := 1$ to $i - 1$ do
        Set $b := e - 2^c$.
        Merge two subarrays in $\mathtt{A}[b \ldots e-1]$.
    end for
    Set $e := e + 1$.
end while
```

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



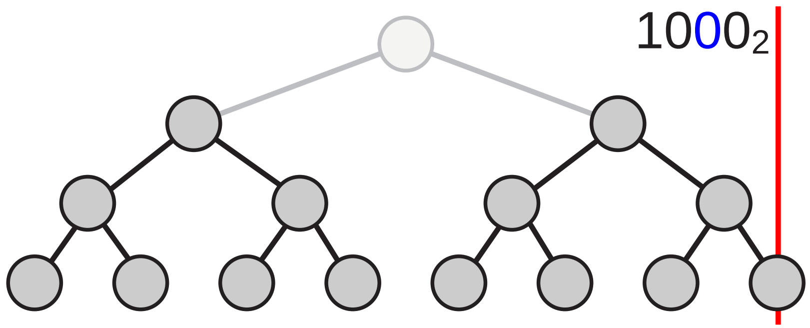
Space-Efficient Divide-and-Conquer

- Use "folklore" approach for Euler-tour like postorder traversal:

  Let $b = 0$ and $e = 1$.
  while $b \neq 0$ or $e \neq n$ **do**
      Let i be index of e's least significant bit (lowest index: 1).
      for $c := 1$ to $i - 1$ **do**
          Set $b := e - 2^c$.
          Merge two subarrays in $\mathtt{A}[b \ldots e - 1]$.
      end for
      Set $e := e + 1$.
  end while

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



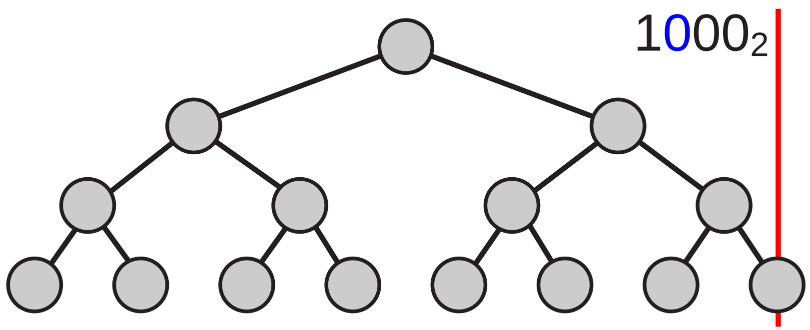
Space-Efficient Divide-and-Conquer

- Use "folklore" approach for Euler-tour like postorder traversal:

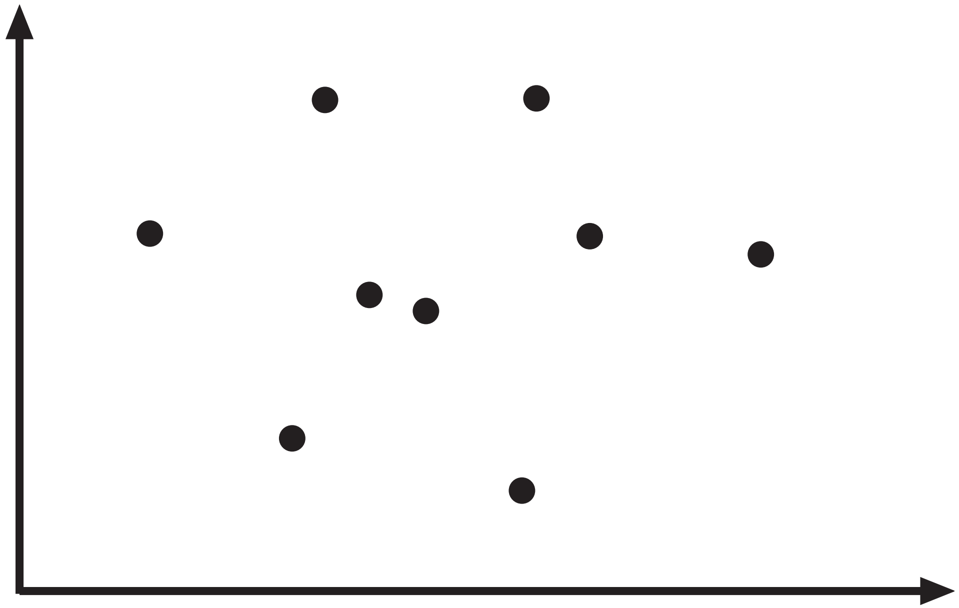
```
Let b = 0 and e = 1.
while b ≠ 0 or e ≠ n do
    Let i be index of e's least significant bit (lowest index: 1).
    for c := 1 to i − 1 do
        Set b := e − 2^c.
        Merge two subarrays in A[b ... e − 1].
    end for
    Set e := e + 1.
end while
```

Observations:

- Can compute LSB-index in amortized $\mathcal{O}(1)$ time ($\rightarrow$ binary counter).
- Only need to care about merging.



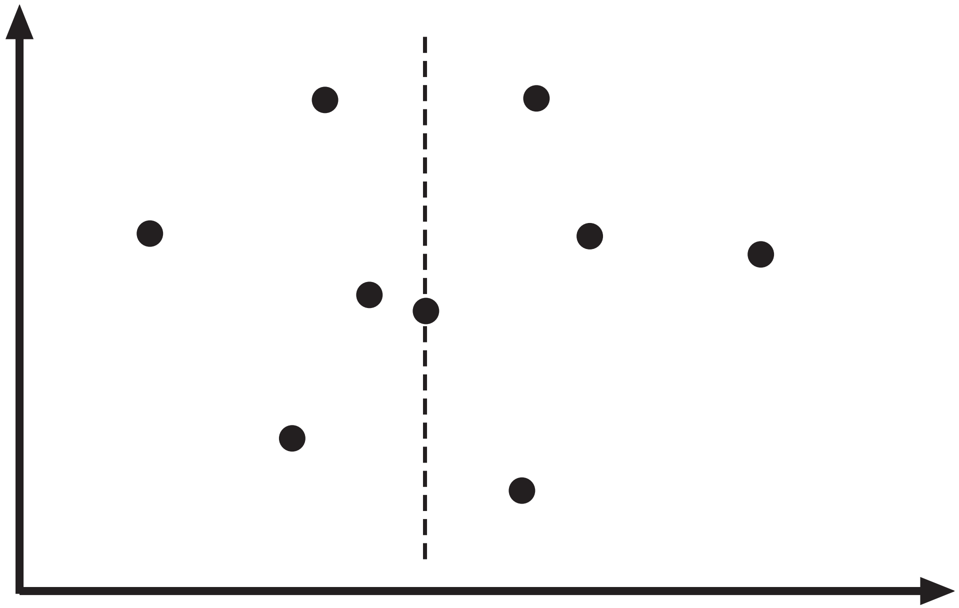
- Classic divide-and-conquer algorithm [Bentley & Shamos, 1976].



Application: Closest Pair

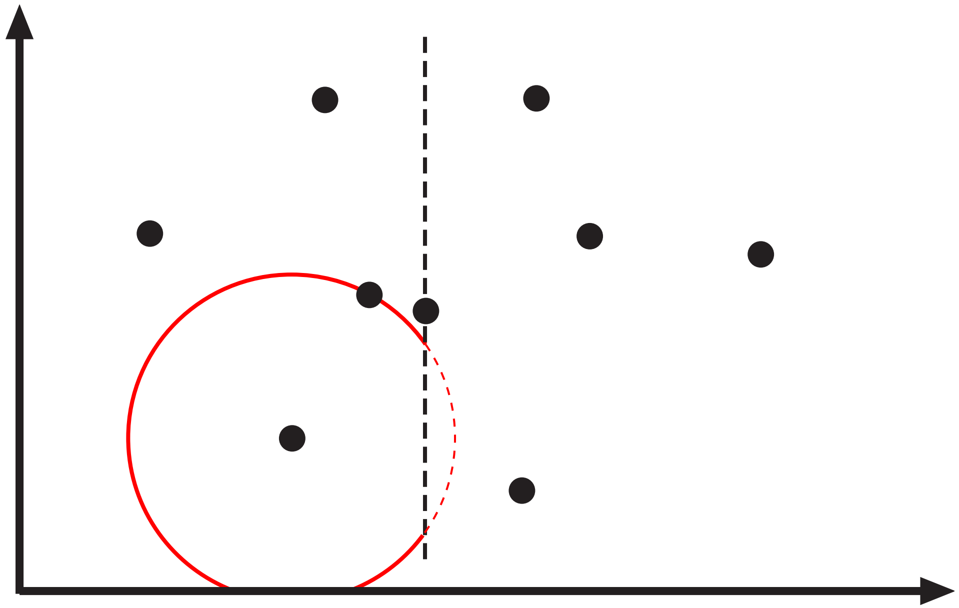
- Classic divide-and-conquer algorithm [Bentley & Shamos, 1976].

1. Partition points according to x-median x_{med}.



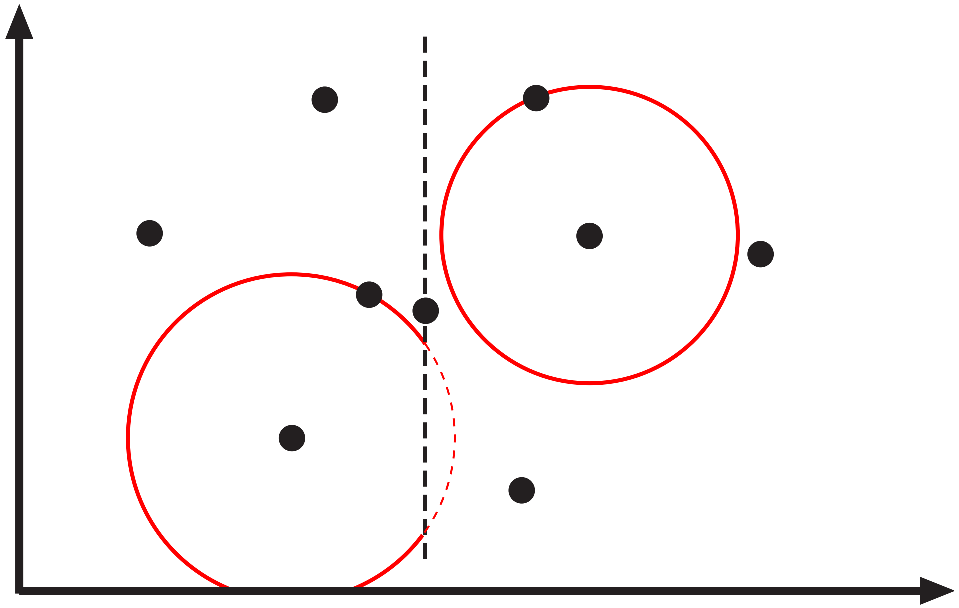
- Classic divide-and-conquer algorithm [Bentley & Shamos, 1976].

1. Partition points according to x-median x_{med}.
2. Recursively find closest pair in left subset.



- Classic divide-and-conquer algorithm [Bentley & Shamos, 1976].

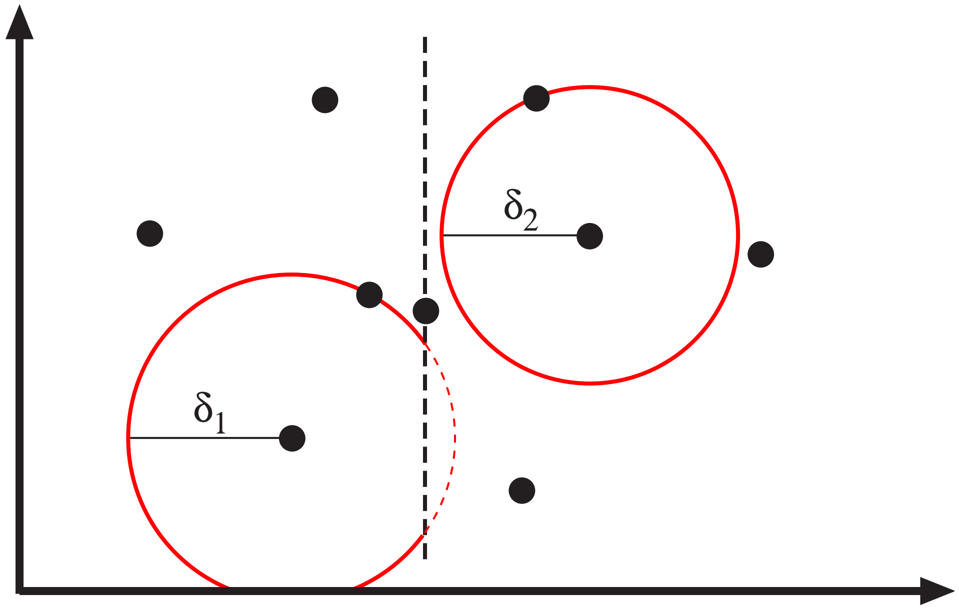
1. Partition points according to x-median x_{med}.
2. Recursively find closest pair in left subset.
3. Recursively find closest pair in right subset.



Application: Closest Pair

- Classic divide-and-conquer algorithm [Bentley & Shamos, 1976].

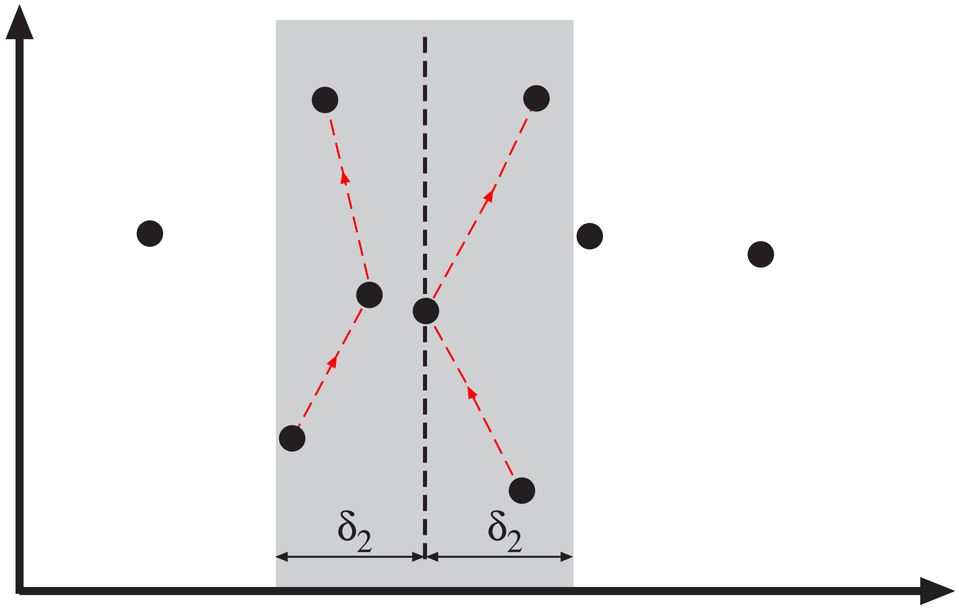
1. Partition points according to x-median x_{med}.
2. Recursively find closest pair in left subset.
3. Recursively find closest pair in right subset.
4. Merge by examining all points in strip of width $2 \cdot \min\{\delta_1, \delta_2\}$ centered at x_{med}.



Application: Closest Pair

- Classic divide-and-conquer algorithm [Bentley & Shamos, 1976].

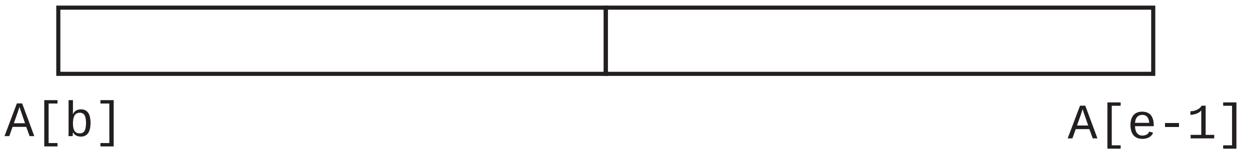
1. Partition points according to x-median x_{med}.
2. Recursively find closest pair in left subset.
3. Recursively find closest pair in right subset.
4. Merge by examining all points in strip of width $2 \cdot \min\{\delta_1, \delta_2\}$ centered at x_{med}.



Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:

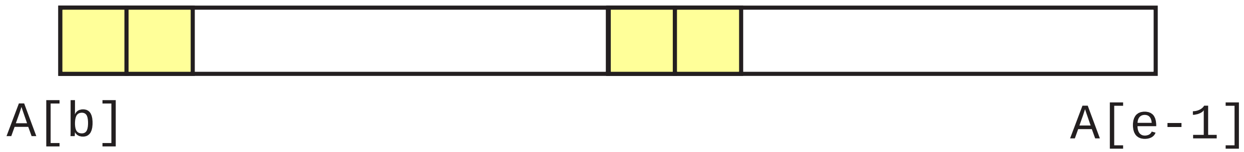
  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \dots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \dots e-1]$ is sorted according to $<_y$.



- Merging two subarrays in $A[b \dots e-1]$:

Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:
  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \dots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \dots e-1]$ is sorted according to $<_y$.

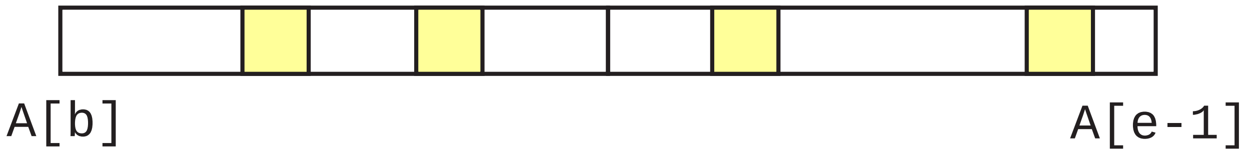


- Merging two subarrays in $A[b \dots e-1]$:
  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).

Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:

  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \dots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \dots e-1]$ is sorted according to $<_y$.

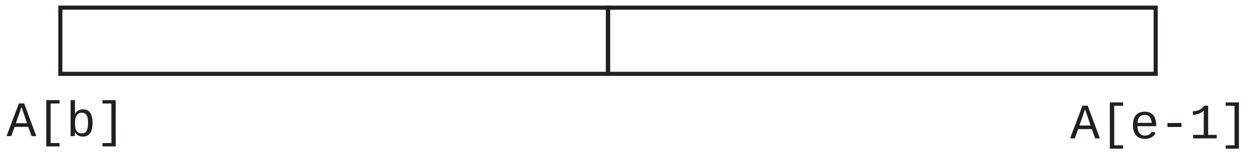


- Merging two subarrays in $A[b \dots e-1]$:

  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).
  2. Insert "closest points" into sorted order.

Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:
  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \ldots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \ldots e-1]$ is sorted according to $<_y$.



- Merging two subarrays in $A[b \ldots e-1]$:
  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).
  2. Insert "closest points" into sorted order.

Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:

  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \ldots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \ldots e-1]$ is sorted according to $<_y$.

- Merging two subarrays in $A[b \ldots e-1]$:

  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).
  2. Insert "closest points" into sorted order.
  3. In-place partition subarray, such that points inside the 2δ-strip are put together (maintain y-order) [Katajainen & Pasanen, 1992].
  4. Find closest pair inside the strip.

Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:

  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \ldots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \ldots e-1]$ is sorted according to $<_y$.

- Merging two subarrays in $A[b \ldots e-1]$:

  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).
  2. Insert "closest points" into sorted order.
  3. In-place partition subarray, such that points inside the 2δ-strip are put together (maintain y-order) [Katajainen & Pasanen, 1992].
  4. Find closest pair inside the strip.
  5. Merge partitions of each subarray according to $<_y$ [Geffert et al., 2000].

Space-efficiently Merging Two Subarrays

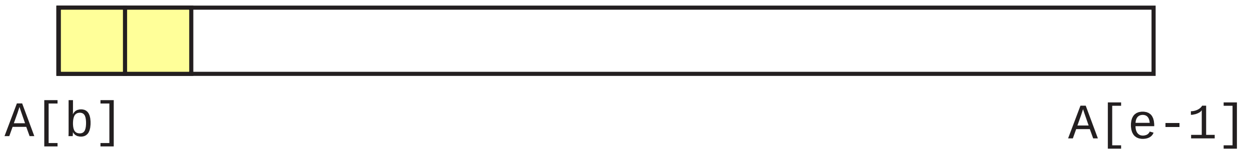
- Maintain two invariants as postconditions of merging:
  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \dots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \dots e-1]$ is sorted according to $<_y$.

- Merging two subarrays in $A[b \dots e-1]$:
  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).
  2. Insert "closest points" into sorted order.
  3. In-place partition subarray, such that points inside the 2δ-strip are put together (maintain y-order) [Katajainen & Pasanen, 1992].
  4. Find closest pair inside the strip.
  5. Merge partitions of each subarray according to $<_y$ [Geffert et al., 2000].

Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:

  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \ldots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \ldots e-1]$ is sorted according to $<_y$.



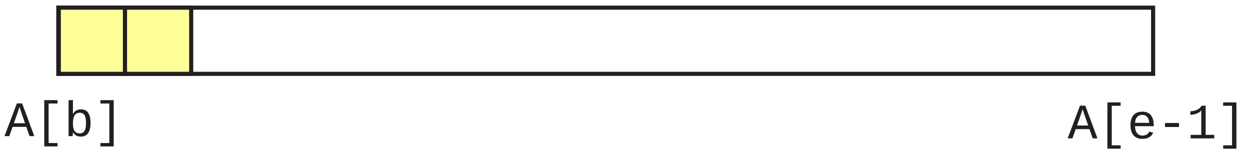
- Merging two subarrays in $A[b \ldots e-1]$:

  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).
  2. Insert "closest points" into sorted order.
  3. In-place partition subarray, such that points inside the 2δ-strip are put together (maintain y-order) [Katajainen & Pasanen, 1992].
  4. Find closest pair inside the strip.
  5. Merge partitions of each subarray according to $<_y$ [Geffert et al., 2000].
  6. Move "closest points" to the front ("reverse insertion sort")

Space-efficiently Merging Two Subarrays

- Maintain two invariants as postconditions of merging:

  1. $A[b]$ and $A[b+1]$ form a closest pair in $A[b \ldots e-1]$.
  2. If $e-1 > b+1$, then $A[b+2 \ldots e-1]$ is sorted according to $<_y$.



- Merging two subarrays in $A[b \ldots e-1]$:

  1. Using Invariant 1, compute δ_1 and δ_2 (hence $\delta := \min\{\delta_1, \delta_2\}$).
  2. Insert "closest points" into sorted order.
  3. In-place partition subarray, such that points inside the 2δ-strip are put together (maintain y-order) [Katajainen & Pasanen, 1992].
  4. Find closest pair inside the strip.
  5. Merge partitions of each subarray according to $<_y$ [Geffert et al., 2000].
  6. Move "closest points" to the front ("reverse insertion sort")

- Each step can be done in-place in linear time.

Theorem 2.1

The Closest-Pair problem can be solved optimally by an in-place algorithm using $\mathcal{O}(\log_2 n)$ extra bits.

Remark:

- Can give small almost tight upper bounds on the constants in both the space and time complexity.

Theorem 2.1
The Closest-Pair problem can be solved optimally by an in-place algorithm using $\mathcal{O}(\log_2 n)$ extra bits.

Remark:

- Can give small almost tight upper bounds on the constants in both the space and time complexity.

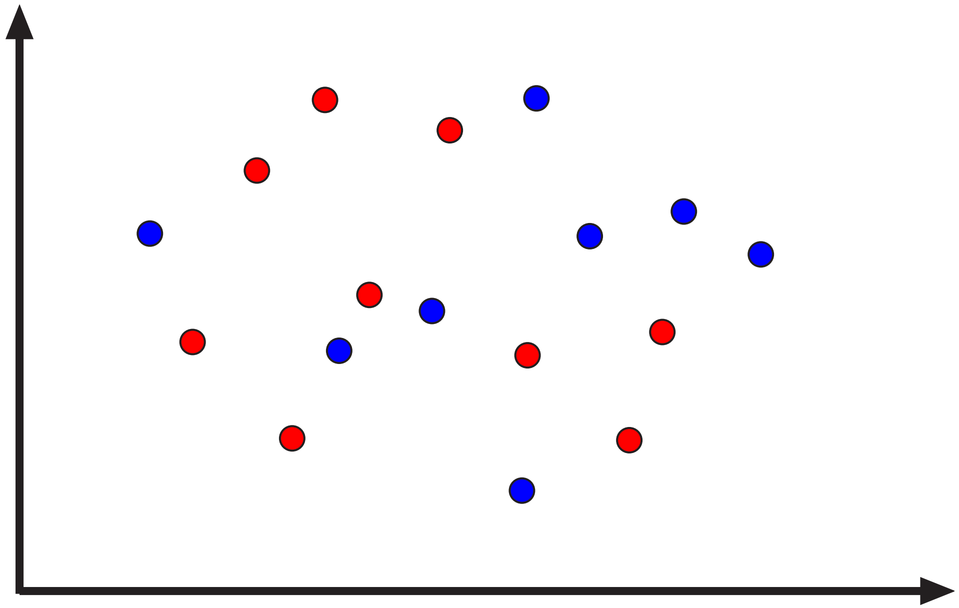
Theorem 2.2
The All-Nearest-Neighbor problem can be solved spending either

- $\mathcal{O}(n \log_2 n)$ time and $2 \cdot n \log_2 n + \mathcal{O}(\log_2 n)$ extra bits or
- $\mathcal{O}\left(n \log_2^2 n\right)$ time and $n \log_2 n + \mathcal{O}(\log_2 n)$ extra bits.

(Problem: Even $n \log_2 n$ extra bits not optimal.)

- Given set R of red points, set B of blue points, find

$$(r,b) \in R \times B \quad \text{s.t.} \quad d(r,b) = \min\{d(\rho,\beta) \mid \rho \in R, \beta \in B\}$$

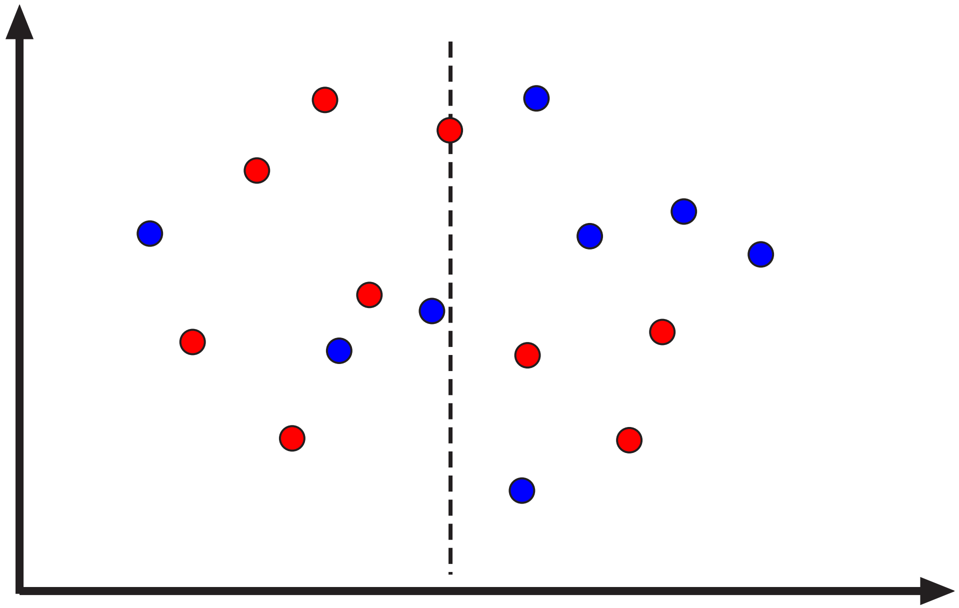


- Given set R of red points, set B of blue points, find

$$(r, b) \in R \times B \quad \text{s.t.} \quad d(r, b) = \min\{d(\rho, \beta) \mid \rho \in R, \beta \in B\}$$

Approach:

- Subdivide plane by line though x-median of $R \cup B$

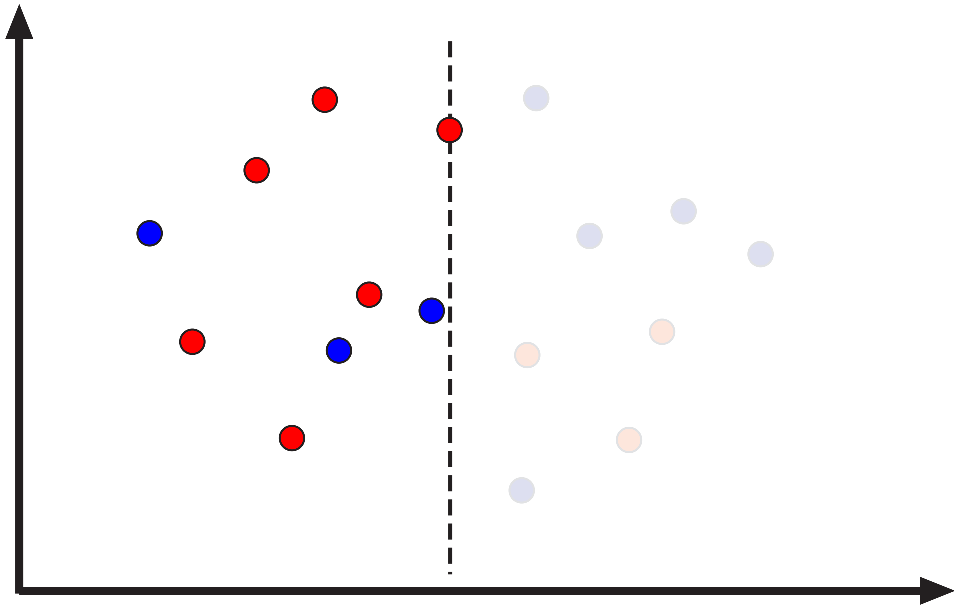


- Given set R of red points, set B of blue points, find

$$(r,b) \in R \times B \quad \text{s.t.} \quad d(r,b) = \min\{d(\rho,\beta) \mid \rho \in R, \beta \in B\}$$

Approach:

- Subdivide plane by line though x-median of $R \cup B$, recurse.

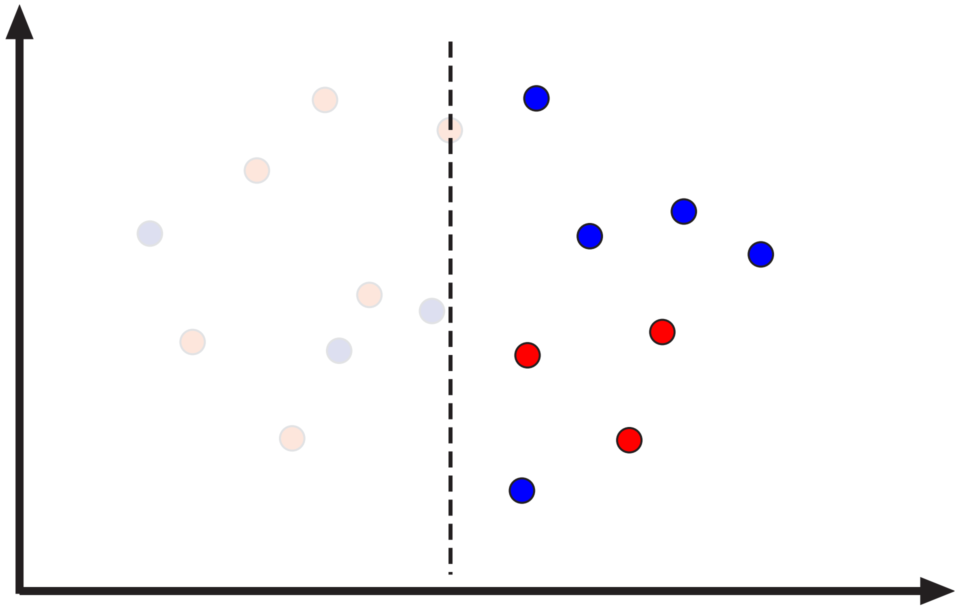


- Given set R of red points, set B of blue points, find

$$(r, b) \in R \times B \quad \text{s.t.} \quad d(r, b) = \min\{d(\rho, \beta) \mid \rho \in R, \beta \in B\}$$

Approach:

- Subdivide plane by line though x-median of $R \cup B$, recurse.

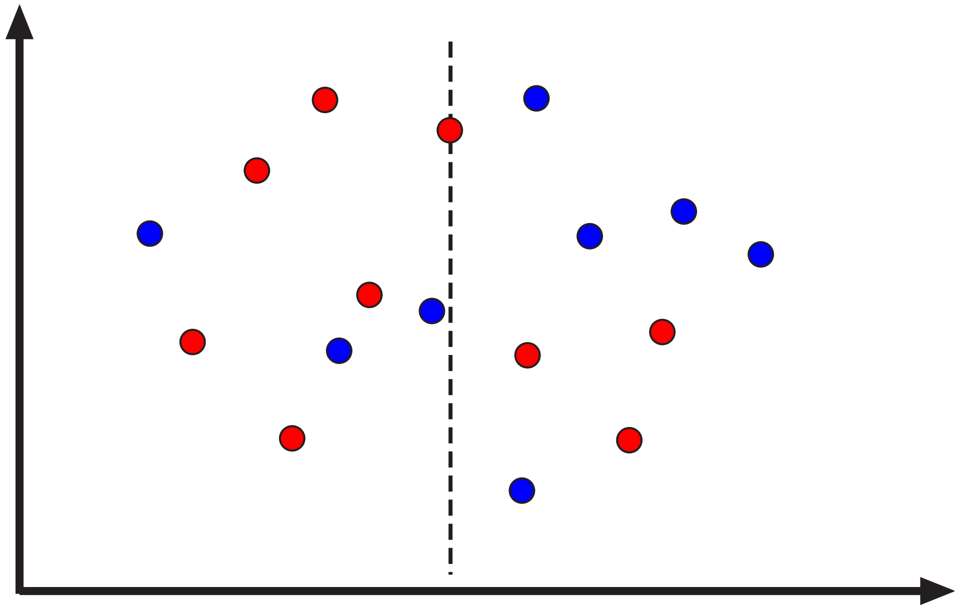


- Given set R of red points, set B of blue points, find

$$(r,b) \in R \times B \quad \text{s.t.} \quad d(r,b) = \min\{d(\rho,\beta) \mid \rho \in R, \beta \in B\}$$

Approach:

- Subdivide plane by line though x-median of $R \cup B$, recurse.
- Merge

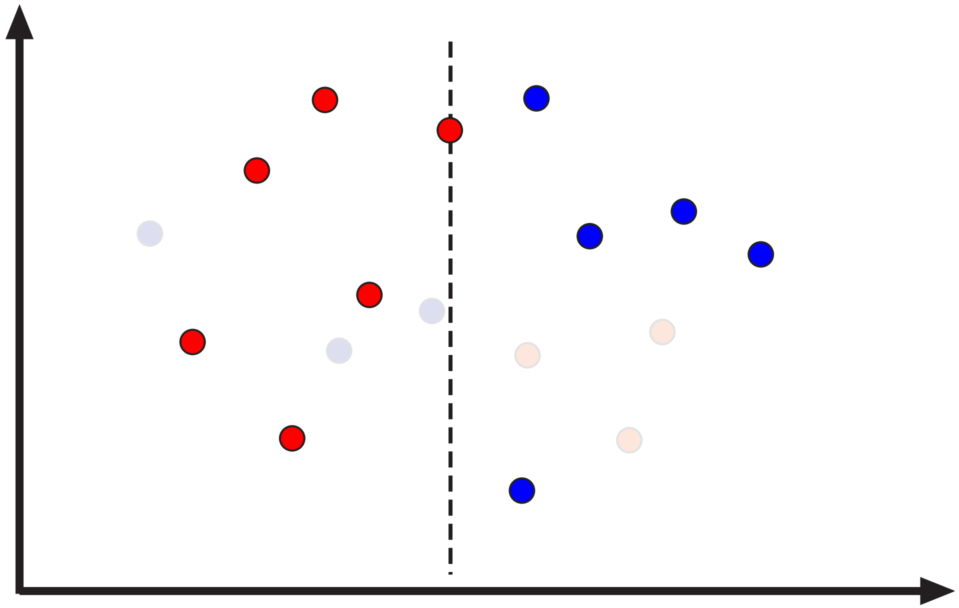


- Given set R of red points, set B of blue points, find

$$(r,b) \in R \times B \quad \text{s.t.} \quad d(r,b) = \min\{d(\rho,\beta) \mid \rho \in R, \beta \in B\}$$

Approach:

- Subdivide plane by line though x-median of $R \cup B$, recurse.
- Merge by processing $(R_{\text{left}}, B_{\text{right}})$

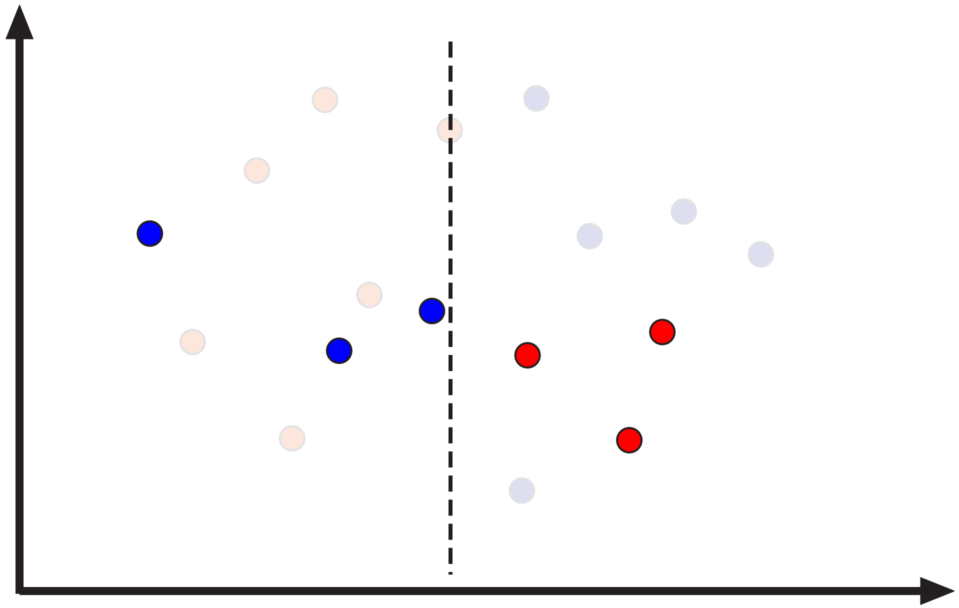


- Given set R of red points, set B of blue points, find

$$(r,b) \in R \times B \quad \text{s.t.} \quad d(r,b) = \min\{d(\rho,\beta) \mid \rho \in R, \beta \in B\}$$

Approach:

- Subdivide plane by line though x-median of $R \cup B$, recurse.
- Merge by processing $(R_{\text{left}}, B_{\text{right}})$ and $(B_{\text{left}}, R_{\text{right}})$.

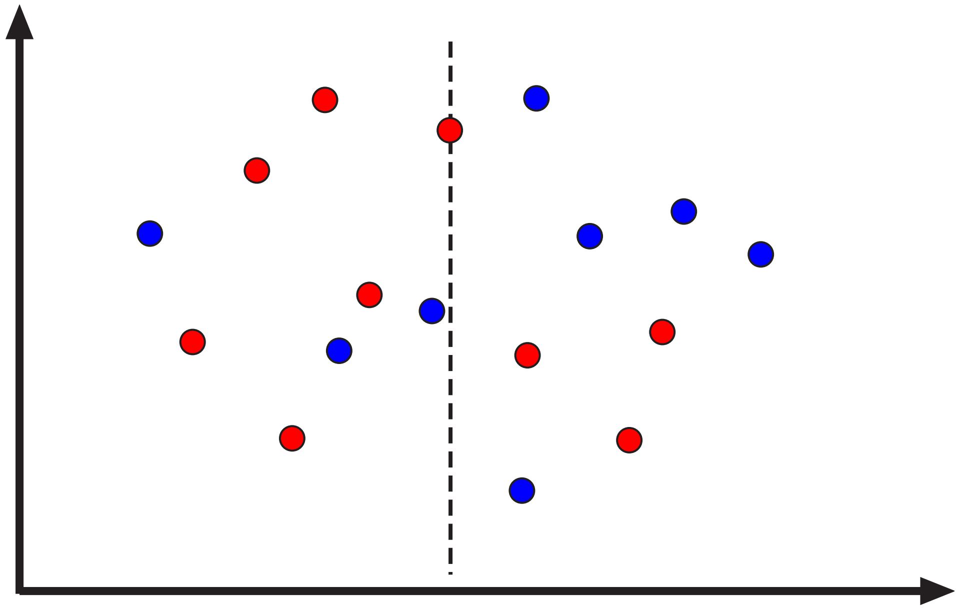


- Given set R of red points, set B of blue points, find

$$(r,b) \in R \times B \quad \text{s.t.} \quad d(r,b) = \min\{d(\rho,\beta) \mid \rho \in R, \beta \in B\}$$

Approach:

- Subdivide plane by line though x-median of $R \cup B$, recurse.
- Merge by processing $(R_{\text{left}}, B_{\text{right}})$ and $(B_{\text{left}}, R_{\text{right}})$.



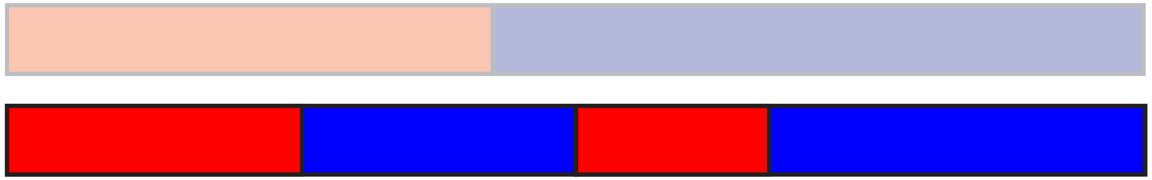
Realizing the Divide-Step In-Place

Problem:

- Subdividing into “left” and “right” partitions red and blue points.
- Points do not have a color tag (else: $\Theta(N)$ extra bits!).
- Need to revert partitioning before merge step.

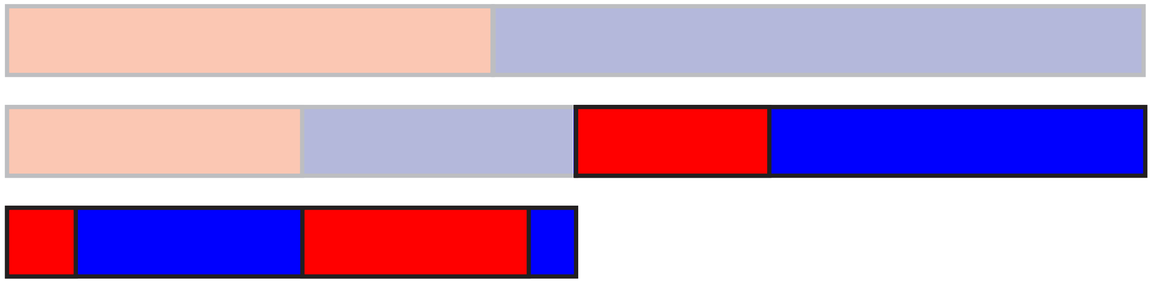
Problem:

- Subdividing into “left” and “right” partitions red and blue points.
- Points do not have a color tag (else: $\Theta(N)$ extra bits!).
- Need to revert partitioning before merge step.



Problem:

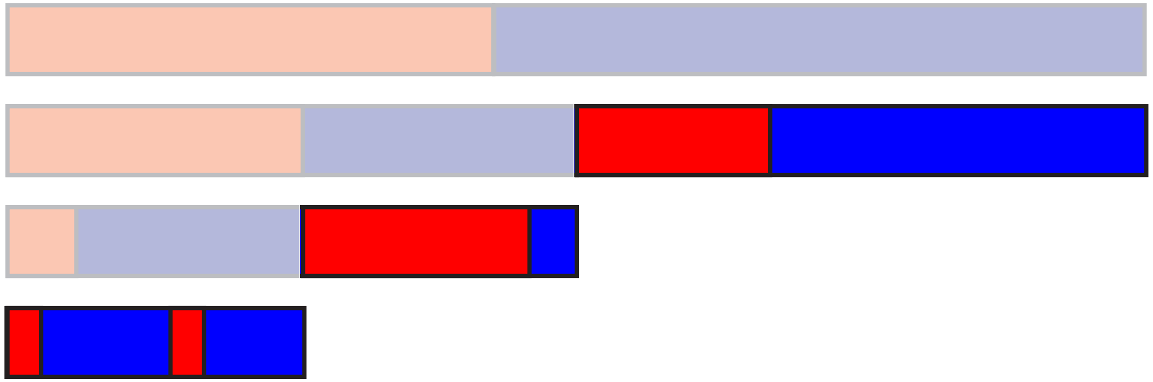
- Subdividing into “left” and “right” partitions red and blue points.
- Points do not have a color tag (else: $\Theta(N)$ extra bits!).
- Need to revert partitioning before merge step.



Realizing the Divide-Step In-Place

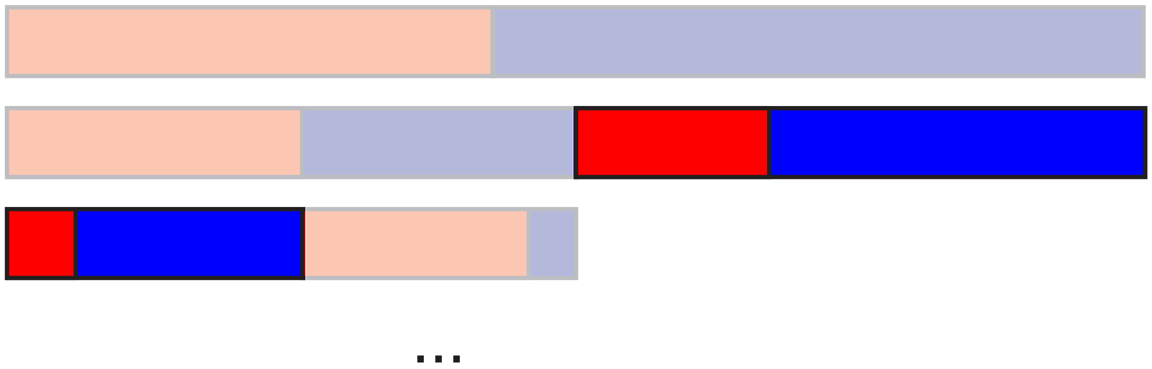
Problem:

- Subdividing into "left" and "right" partitions red and blue points.
- Points do not have a color tag (else: $\Theta(N)$ extra bits!).
- Need to revert partitioning before merge step.



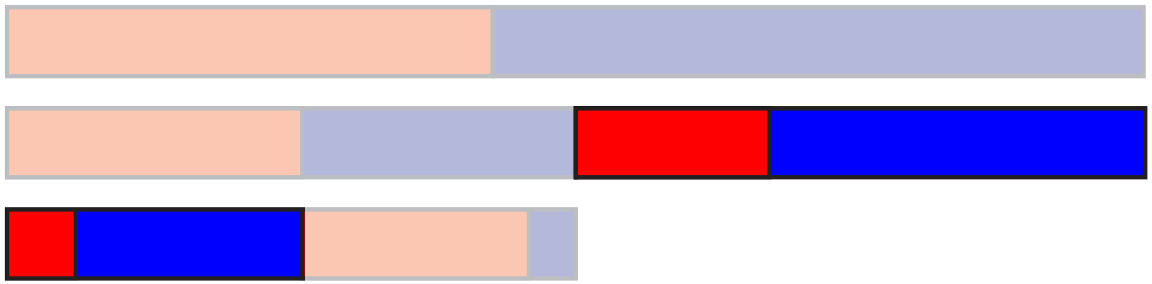
Problem:

- Subdividing into “left” and “right” partitions red and blue points.
- Points do not have a color tag (else: $\Theta(N)$ extra bits!).
- Need to revert partitioning before merge step.



Problem:

- Subdividing into “left” and “right” partitions red and blue points.
- Points do not have a color tag (else: $\Theta(N)$ extra bits!).
- Need to revert partitioning before merge step.



...

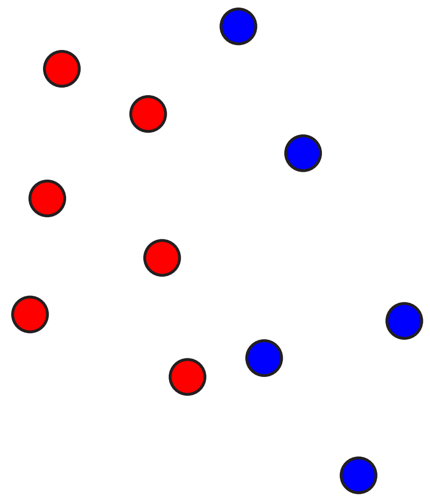
Good News:

- Can use y-direction of points to encode color (. . . some details missing), also may need to stop recursion early.

- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

1: **if** $|R \cup B| \leq c$ **then**
2: Solve brute-force.
3: W.l.o.g. assume $|R| \geq |B|$.
4: Pick random $r \in R$.
5: Find $b \in B$ closest to R.
6: Compute left envelope of disks centered at points in B having radius $d(r, b)$.
7: $S := \{r \in R \mid r \text{ left of envelope}\}$.
8: Repeat with $(B, R \setminus S)$.

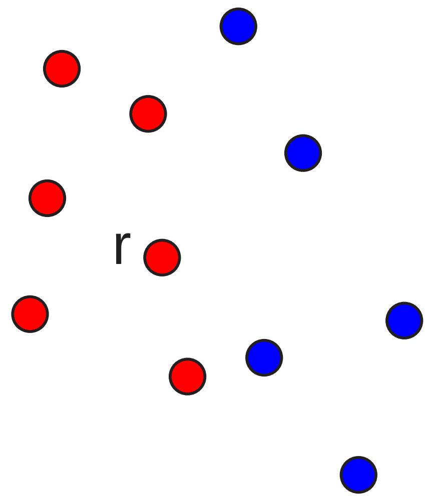


Solving a subproblem

- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

1: **if** $|R \cup B| \leq c$ **then**
2: Solve brute-force.
3: W.l.o.g. assume $|R| \geq |B|$.
4: Pick random $r \in R$.
5: Find $b \in B$ closest to R.
6: Compute left envelope of disks centered at points in B having radius $d(r, b)$.
7: $S := \{r \in R \mid r$ left of envelope$\}$.
8: Repeat with $(B, R \setminus S)$.

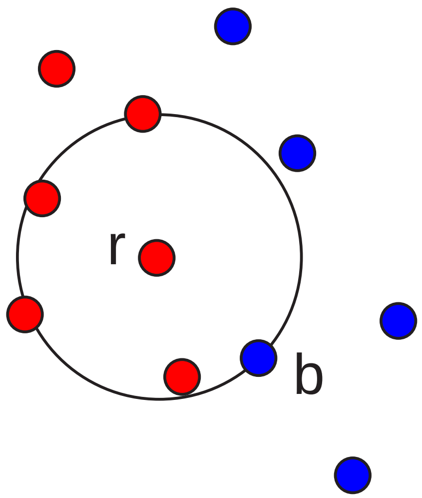


Solving a subproblem

- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

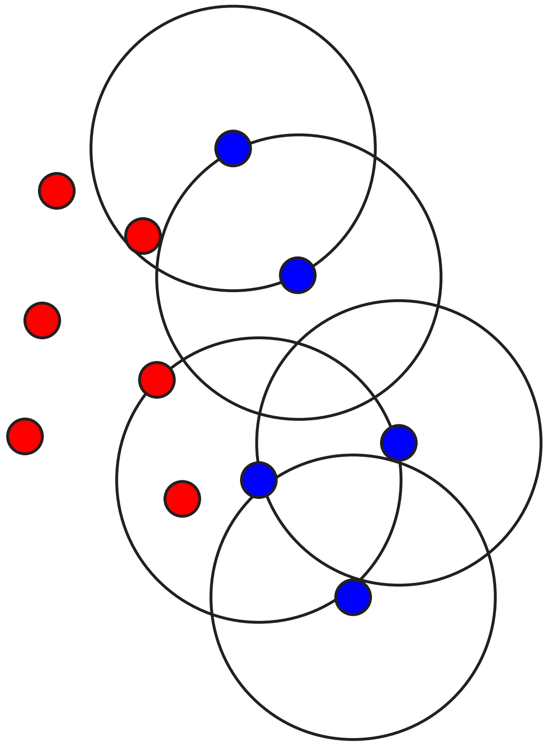
1. **if** $|R \cup B| \leq c$ **then**
2.   Solve brute-force.
3. W.l.o.g. assume $|R| \geq |B|$.
4. Pick random $r \in R$.
5. Find $b \in B$ closest to R.
6. Compute left envelope of disks centered at points in B having radius $d(r, b)$.
7. $S := \{r \in R \mid r$ left of envelope$\}$.
8. Repeat with $(B, R \setminus S)$.



- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

1: **if** $|R \cup B| \leq c$ **then**
2: Solve brute-force.
3: W.l.o.g. assume $|R| \geq |B|$.
4: Pick random $r \in R$.
5: Find $b \in B$ closest to R.
6: Compute left envelope of disks centered at points in B having radius $d(r, b)$.
7: $S := \{r \in R \mid r \text{ left of envelope}\}$.
8: Repeat with $(B, R \setminus S)$.

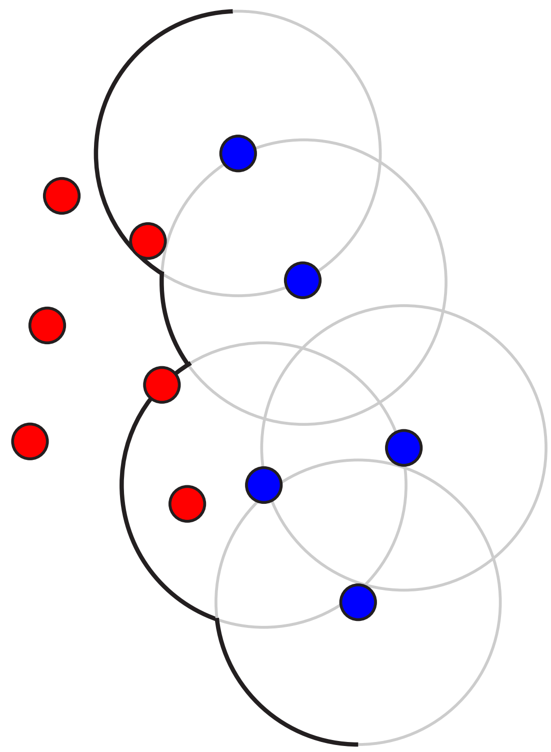


Solving a subproblem

- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

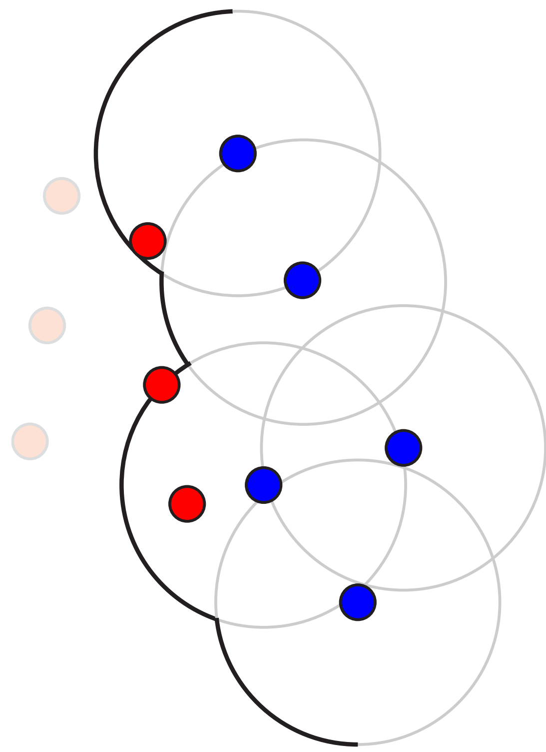
1: **if** $|R \cup B| \leq c$ **then**
2: Solve brute-force.
3: W.l.o.g. assume $|R| \geq |B|$.
4: Pick random $r \in R$.
5: Find $b \in B$ closest to R.
6: Compute left envelope of disks centered at points in B having radius $d(r,b)$.
7: $S := \{r \in R \mid r \text{ left of envelope}\}$.
8: Repeat with $(B, R \setminus S)$.



- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

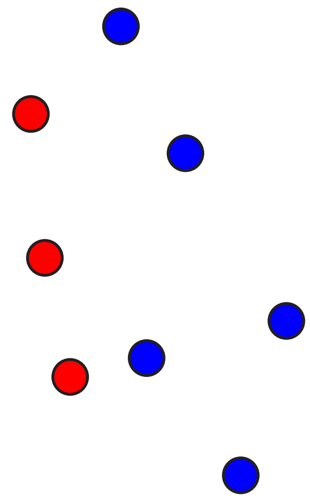
1: **if** $|R \cup B| \leq c$ **then**
2:     Solve brute-force.
3: W.l.o.g. assume $|R| \geq |B|$.
4: Pick random $r \in R$.
5: Find $b \in B$ closest to R.
6: Compute left envelope of disks centered at points in B having radius $d(r, b)$.
7: $S := \{r \in R \mid r$ left of envelope$\}$.
8: Repeat with $(B, R \setminus S)$.



- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

1: **if** $|R \cup B| \leq c$ **then**
2:     Solve brute-force.
3: W.l.o.g. assume $|R| \geq |B|$.
4: Pick random $r \in R$.
5: Find $b \in B$ closest to R.
6: Compute left envelope of disks centered at points in B having radius $d(r, b)$.
7: $S := \{r \in R \mid r \text{ left of envelope}\}$.
8: Repeat with $(B, R \setminus S)$.

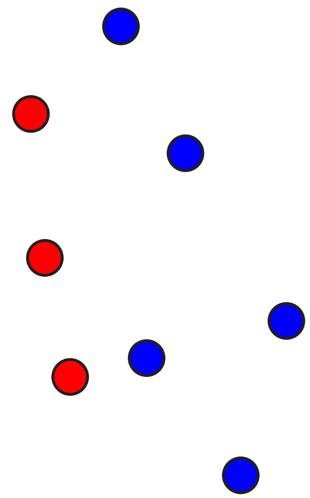


Solving a subproblem

- Find bichromatic closest pair for R, B separated by vertical line.

Require: R, B sorted by $<_y$.

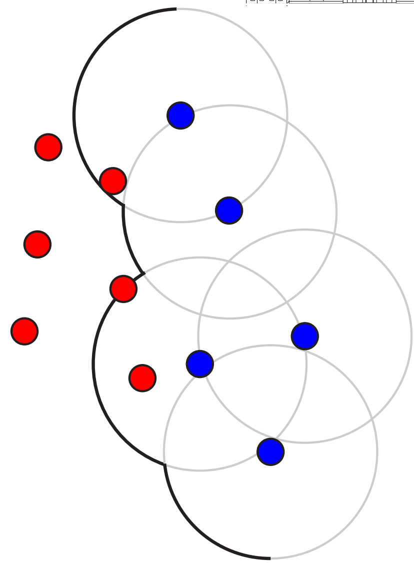
if $|R \cup B| \leq c$ **then**
    Solve brute-force.
W.l.o.g. assume $|R| \geq |B|$.
Pick random $r \in R$.
Find $b \in B$ closest to R.
Compute left envelope of disks centered at points in B having radius $d(r, b)$.
$S := \{r \in R \mid r$ left of envelope$\}$.
Repeat with $(B, R \setminus S)$.



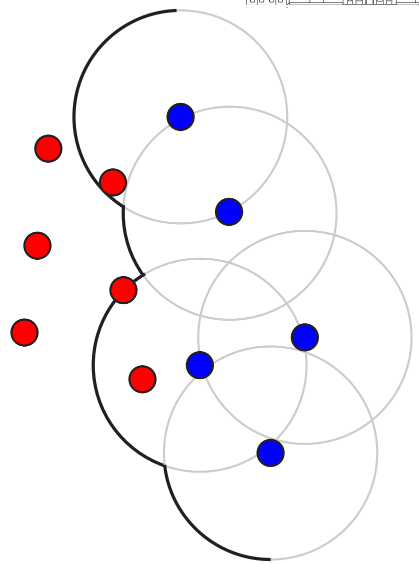
- Algorithm runs in expected linear time ($\rightarrow$ Quicksort-like partition).

Computing the Left Envelope:

- All circles of same radius, so no need to explicitly construct the envelope.
- Use Graham's-scan type algorithm to extract points contributing to left envelope.
- Simultaneously scan these points and R to remove points from R.



Computing the Left Envelope:

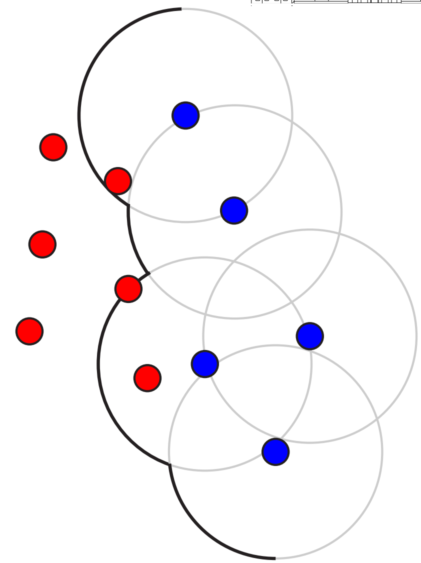


- All circles of same radius, so no need to explicitly construct the envelope.
- Use Graham's-scan type algorithm to extract points contributing to left envelope.
- Simultaneously scan these points and R to remove points from R.

Leaving Out Technical Details:

- Construction can be reverted (restoring $<_y$-order) in-place.
- Removal of points from R can be reverted in-place.

Computing the Left Envelope:



- All circles of same radius, so no need to explicitly construct the envelope.
- Use Graham's-scan type algorithm to extract points contributing to left envelope.
- Simultaneously scan these points and R to remove points from R.

Leaving Out Technical Details:

- Construction can be reverted (restoring $<_y$-order) in-place.
- Removal of points from R can be reverted in-place.

Theorem 3.1
The bichromatic closest pair problem can be solved in-place in expected running time $\mathcal{O}(n \log_2 n)$.

Overview

1. Introduction
2. Closest Pair & All Nearest Neighbors
3. Bichromatic Closest Pair
4. Conclusions

Space-efficient algorithms:

- "What can be done using an array and some pointers?"
- Possible: Geometric *divide-and-conquer* (as long as work is done upon returning from recursion. . .).

Preliminary results:

- (Convex hull and) convex-hull related problems.
- [Bichromatic] closest pair, all-nearest-neighbors.

Work in progress:

- Scheme for general geometric *divide-and-conquer*.
- Immediate consequence: Orthogonal line segment intersection.

Bibliography

[Balaban, 1995] I. J. Balaban. An optimal algorithm for finding segments [*sic!*] intersections. In: *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, pages 211–219, New York, 1995. ACM Press.

[Bentley & Shamos, 1976] J. L. Bentley and M. I. Shamos. Divide-and-Conquer in multidimensional space. In: *Proceedings of the Eighth Annual ACM Symposium on the Theory of Computation*, pages 220–230. Association for Computing Machinery, 1976.

[Brodnik & Munro, 1999] A. Brodnik and J. I. Munro. Membership in constant time and almost minimum place. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.

[Brodnik et al., 1999] A. Brodnik, S. Carlsson, E. D. Demaine, J. I. Munro, and R. Sedgewick. Resizable arrays in optimal time and space. In: F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures, 6th International Workshop, WADS '99*, volume 1663 of *Lecture Notes in Computer Science*, pages 37–48, Berlin, 1999. Springer.

[Brönnimann et al., 2002] H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. T. Toussaint. Optimal in-place planar convex hull algorithms. In: S. Rajsbaum, editor, *Proceedings of Latin American Theoretical Informatics (LATIN 2002)*, volume 2286 of *Lecture Notes in Computer Science*, pages 494–507, Berlin, 2002. Springer.

[Brönnimann et al., 2004] H. Brönnimann, T. M.-Y. Chan, and E. Y. Chen. Towards in-place geometric algorithms. In: *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, page (to appear). ACM Press, 2004.

[Carlsson & Sundström, 1995] S. Carlsson and M. Sundström. Linear-time in-place selection in less than $3n$ comparisons. In: J. Staples, P. Eades, N. Katoh, and A. Moffat, editors, *Algorithms and Computation, 6th International Symposium, Proceedings*, volume 1004 of *Lecture Notes in Computer Science*, pages 244–253, Berlin, 1995. Springer.

[Chan, 1996] T. M.-Y. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Computational Geometry: Theory and Applications*, 16(14):361–368, April 1996.

[Chen & Chan, 2003] E. Y. Chen and T. M.-Y. Chan. A space-efficent algorithm for line segment intersection. In: *Proceedings of the 15th Canadian Conference on Computational Geometry*, pages 68–71, 2003.

[Floyd, 1964] R. W. Floyd. Algorithm 245: Treesort. *Communications of the ACM*, 7(12):701, December 1964.

[Francheschini et al., 2002] G. Francheschini, R. Grossi, J. I. Munro, and L. Pagli. Implicit B-trees: New results for the dictionary problem. In: *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 145–154, 2002.

[Geffert et al., 2000] V. Geffert, J. Katajainen, and T. Pasanen. Asymptotically efficient in-place merging. *Theoretical Computer Science*, 237(1–2):159–181, April 2000.

[Graham, 1972] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[Huang & Knuth, 1986] B.-C. Huang and D. E. Knuth. A one-way, stackless quicksort algorithm. *BIT*, 26:127–130, 1986.

[Katajainen & Pasanen, 1992] J. Katajainen and T. Pasanen. Stable minimum space partitioning in linear time. *BIT*, 32:580–585, 1992.

[Katajainen & Pasanen, 1994] J. Katajainen and T. Pasanen. Sorting multisets stably in minimum space. *Acta Informatica*, 31(4):301–313, 1994.

[Munro, 1986] J. I. Munro. An implicit data structure supporting insertion, deletion, and search in $o(\log^2 n)$ time. *Journal of Computer and System Sciences*, 33(1):66–74, August 1986.

[Raman et al., 2001] R. Raman, V. Raman, and S. S. Rao. Succinct dynamic data structures. In: F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures, 7th International Workshop, WADS 2001*, volume 2125 of *Lecture Notes in Computer Science*, pages 426–437, Berlin, 2001. Springer.

[Wegner, 1987] L. M. Wegner. A generalized, stackless quicksort algorithm. *BIT*, 27:44–48, 1987.