Algorithm design and analysis

— Divide and conquer —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Algorithm design and analysis

— Mergesort —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Divide and Conquer

- Break up a problem into several parts.

Divide and Conquer

- Break up a problem into several parts.
- Solve each part recursively.

Divide and Conquer

- Break up a problem into several parts.
- Solve each part recursively.
- Solve base cases by brute force.

Divide and Conquer

- Break up a problem into several parts.
- Solve each part recursively.
- Solve base cases by brute force.
- Efficiently combine solutions for sub-problems into final solution.

Divide and Conquer

- Break up a problem into several parts.
- Solve each part recursively.
- Solve base cases by brute force.
- Efficiently combine solutions for sub-problems into final solution.
- Common use:
  - Partition problem into two equal sub-problems of size $n/2$.
  - Solve each part recursively.
  - Combine the two solutions in $O(n)$ time.
  - Resulting running time is $O(n \log n)$.

Mergesort

SORT

INSTANCE Nonempty list $L = x_1, x_2, \ldots, x_n$ of integers.

SOLUTION A permutation $y_1, y_2, \ldots, y_n$ of $x_1, x_2, \ldots, x_n$ such that $y_i \leq y_{i+1}$, for all $1 \leq i < n$.

Mergesort

Sort

INSTANCE Nonempty list $L = x_1, x_2, \ldots, x_n$ of integers.

SOLUTION A permutation $y_1, y_2, \ldots, y_n$ of $x_1, x_2, \ldots, x_n$ such that $y_i \leq y_{i+1}$, for all $1 \leq i < n$.

- Mergesort is a divide-and-conquer algorithm for sorting.
  1. Partition L into two lists A and B of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ respectively.
  2. Recursively sort A.
  3. Recursively sort B.
  4. Merge the sorted lists A and B into a single sorted list.

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while *both lists are nonempty* **do**
    Let a_i and b_j be the elements pointed to by the *current* pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller element belonged to;
end
Append the rest of the non-empty list to the output.

1	4	8	9

3	5	6	7

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while *both lists are nonempty* **do**
    Let a_i and b_j be the elements pointed to by the *current* pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller element belonged to;
end
Append the rest of the non-empty list to the output.

1	4	8	9

3	5	6	7

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

```
Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while both lists are nonempty do
    Let a_i and b_j be the elements pointed to by the
        current pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller
        element belonged to;
end
Append the rest of the non-empty list to the output.
```

1	4	8	9

3	5	6	7

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

Maintain a current pointer for each list;
Initialise each pointer to the front of the list;

while *both lists are nonempty* **do**
  Let a_i and b_j be the elements pointed to by the *current* pointers;
  Append the smaller of the two to the output list;
  Advance the current pointer in the list that the smaller element belonged to;
end
Append the rest of the non-empty list to the output.

1	4	8	9

3	5	6	7

1

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

```
Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while both lists are nonempty do
    Let a_i and b_j be the elements pointed to by the
        current pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller
        element belonged to;
end
Append the rest of the non-empty list to the output.
```

1	4	8	9

3	5	6	7

1 3

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

```
Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while both lists are nonempty do
    Let a_i and b_j be the elements pointed to by the
        current pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller
        element belonged to;
end
Append the rest of the non-empty list to the output.
```

1	4	8	9

3	5	6	7

1 3 4

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

Maintain a current pointer for each list;
Initialise each pointer to the front of the list;

while *both lists are nonempty* **do**
    Let a_i and b_j be the elements pointed to by the *current* pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller element belonged to;
end
Append the rest of the non-empty list to the output.

1	4	8	9

3	5	6	7

1 3 4 5

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

```
Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while both lists are nonempty do
    Let a_i and b_j be the elements pointed to by the
        current pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller
        element belonged to;
end
Append the rest of the non-empty list to the output.
```

1	4	8	9		3	5	6	7

1 3 4 5 6

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while *both lists are nonempty* **do**
  Let a_i and b_j be the elements pointed to by the *current* pointers;
  Append the smaller of the two to the output list;
  Advance the current pointer in the list that the smaller element belonged to;
end
Append the rest of the non-empty list to the output.

1	4	8	9

3	5	6	7

1 3 4 5 6 7

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while *both lists are nonempty* **do**
    Let a_i and b_j be the elements pointed to by the *current* pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller element belonged to;
end
Append the rest of the non-empty list to the output.

1	4	8	9

3	5	6	7

1 3 4 5 6 7 8

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

Maintain a current pointer for each list;
Initialise each pointer to the front of the list;

while *both lists are nonempty* **do**
    Let a_i and b_j be the elements pointed to by the *current* pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller element belonged to;
end
Append the rest of the non-empty list to the output.

1	4	8	9

3	5	6	7

1 3 4 5 6 7 8 9

Merging Two Sorted Lists

Algorithm: Intercalation

input : $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots b_l$.
output: The distances of the vertices from s

```
Maintain a current pointer for each list;
Initialise each pointer to the front of the list;
while both lists are nonempty do
    Let a_i and b_j be the elements pointed to by the
        current pointers;
    Append the smaller of the two to the output list;
    Advance the current pointer in the list that the smaller
        element belonged to;
end
Append the rest of the non-empty list to the output.
```

Running time of this algorithm is $\boxed{O(k+l)}$.

Analysing Mergesort

- Worst-case running time for n elements ($T(n)$) is at most the sum of the worst-case running time for $\lfloor n/2 \rfloor$ elements, for $\lceil n/2 \rceil$ elements, for splitting the input into two lists, and for merging two sorted lists.
- Assume n is a power of 2.

Analysing Mergesort

- Worst-case running time for n elements ($T(n)$) is at most the sum of the worst-case running time for $\lfloor n/2 \rfloor$ elements, for $\lceil n/2 \rceil$ elements, for splitting the input into two lists, and for merging two sorted lists.
- Assume n is a power of 2.

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn, n > 2 \\ T(2) &\leq c \end{aligned}$$

Analysing Mergesort

- Worst-case running time for n elements ($T(n)$) is at most the sum of the worst-case running time for $\lfloor n/2 \rfloor$ elements, for $\lceil n/2 \rceil$ elements, for splitting the input into two lists, and for merging two sorted lists.
- Assume n is a power of 2.

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn, n > 2 \\ T(2) &\leq c \end{aligned}$$

- Three basic ways of solving this recurrence relation:
  1. "Unroll" the recurrence (somewhat informal method).
  2. Guess a solution and substitute into recurrence to check.
  3. Guess solution in $O()$ form and substitute into recurrence to determine the constants.

Unrolling the recurrence

- Recursion tree has log n levels.
- Total work done at each level is cn.
- Running time of the algorithm is $cn \log n$.

Substituting a Solution into the Recurrence

- Guess that the solution is $cn \log n$ (logarithm to the base 2).
- Use induction to check if the solution satisfies the recurrence relation.
- Base case: $n = 2$. Is $T(2) = c \leq 2c \log 2$? Yes.
- Inductive step: assume $T(m) \leq cm \log_2 m$ for all $m < n$. Therefore, $T(n/2) \leq (cn/2) \log n - cn/2$.

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&\leq 2\left((cn/2) \log n - cn/2\right) + cn \\
&= cn \log n
\end{aligned}
$$

Partial Substitution

- Guess that the solution is $kn \log n$ (logarithm to the base 2).
- Substitute guess into the recurrence relation to check what value of k will satisfy the recurrence relation.
- $k \geq c$ will work.

Partial Substitution

- Guess that the solution is $kn \log n$ (logarithm to the base 2).
- Substitute guess into the recurrence relation to check what value of k will satisfy the recurrence relation.
- $k \geq c$ will work.

- Divide into q sub-problems of size $n/2$ and merge in $O(n)$ time. Two distinct cases: $q = 1$ and $q > 2$.
- Divide into two sub-problems of size $n/2$ and merge in $O(n^2)$ time.

Algorithm design and analysis

— Counting inversions —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Divide and Conquer Algorithms

- Study three divide and conquer algorithms:
  - Counting inversions.
  - Finding the closest pair of points.
  - Integer multiplication.
- First two problems use clever conquer strategies.

Divide and Conquer Algorithms

- Study three divide and conquer algorithms:
  - Counting inversions.
  - Finding the closest pair of points.
  - Integer multiplication.
- First two problems use clever conquer strategies.
- Third problem uses a clever divide strategy.

Motivation

- Collaborative filtering match one user's preferences to those of other users.

Motivation

- Collaborative filtering match one user's preferences to those of other users.
- Meta-search engines merge results of multiple search engines to into a better search result.

Motivation

- Collaborative filtering match one user's preferences to those of other users.
- Meta-search engines merge results of multiple search engines to into a better search result.
- Fundamental question: how do we compare a pair of rankings?

Motivation

- Collaborative filtering match one user's preferences to those of other users.
- Meta-search engines merge results of multiple search engines to into a better search result.
- Fundamental question: how do we compare a pair of rankings?
- Suggestion: two rankings are very similar if they have few inversions.

Motivation

- Collaborative filtering match one user's preferences to those of other users.
- Meta-search engines merge results of multiple search engines to into a better search result.
- Fundamental question: how do we compare a pair of rankings?
- Suggestion: two rankings are very similar if they have few inversions.
  - Assume one ranking is the ordered list of integers from 1 to n.
  - The other ranking is a permutation $a_1, a_2, \ldots, a_n$ of the integers from 1 to n.
  - The second ranking has an inversion if there exist i, j such that $i < j$ but $a_i > a_j$.
  - The number of inversions s is a measure of the difference between the rankings.

Counting Inversions

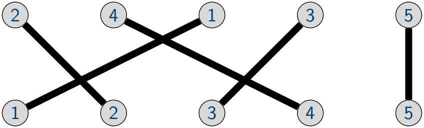
INSTANCE A list $L = x_1, x_2, \ldots, x_n$ of distinct integers between 1 and n.

SOLUTION The number of pairs $(i, j), 1 \leq i < j \leq n$ such $a_i > a_j$.

COUNTING INVERSIONS

INSTANCE A list $L = x_1, x_2, \ldots, x_n$ of distinct integers between 1 and n.

SOLUTION The number of pairs $(i, j), 1 \leq i < j \leq n$ such $a_i > a_j$.



Counting Inversions

Counting Inversions

INSTANCE A list $L = x_1, x_2, \ldots, x_n$ of distinct integers between 1 and n.

SOLUTION The number of pairs $(i, j), 1 \leq i < j \leq n$ such $a_i > a_j$.

1 5 4 8 10 2 6 9 12 11 3 7

1 5 4 8 10 2 6 9 12 11 3 7

$5-4, 5-2, 4-2, 8-2, 10-2$

$6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7$

Counting Inversions

Counting Inversions

INSTANCE A list $L = x_1, x_2, \ldots, x_n$ of distinct integers between 1 and n.

SOLUTION The number of pairs $(i, j), 1 \leq i < j \leq n$ such $a_i > a_j$.

1 5 4 8 10 2 6 9 12 11 3 7

1 5 4 8 10 2 6 9 12 11 3 7

$5-4, 5-2, 4-2, 8-2, 10-2$

$6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7$

$5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7$

Counting Inversions: Algorithm

- How many inversions can be there in a list of n numbers?

Counting Inversions: Algorithm

- How many inversions can be there in a list of n numbers? $\Omega(n^2)$. We cannot afford to compute each inversion explicitly.

Counting Inversions: Algorithm

- How many inversions can be there in a list of n numbers? $\Omega(n^2)$. We cannot afford to compute each inversion explicitly.
- Sorting removes all inversions in $O(n \log n)$ time.Can we modify the **Mergesort** algorithm to count inversions?

Counting Inversions: Algorithm

- How many inversions can be there in a list of n numbers? $\Omega(n^2)$. We cannot afford to compute each inversion explicitly.
- Sorting removes all inversions in $O(n \log n)$ time.Can we modify the **Mergesort** algorithm to count inversions ?
- Candidate algorithm:
  1. Partition L into two lists A and B of size $n/2$ each.
  2. Recursively count the number of inversions in A.
  3. Recursively count the number of inversions in B. and one element in B.

Counting Inversions: Algorithm

- How many inversions can be there in a list of n numbers? $\Omega(n^2)$. We cannot afford to compute each inversion explicitly.
- Sorting removes all inversions in $O(n \log n)$ time.Can we modify the **Mergesort** algorithm to count inversions?
- Candidate algorithm:
  1. Partition L into two lists A and B of size $n/2$ each.
  2. Recursively count the number of inversions in A.
  3. Recursively count the number of inversions in B. and one element in B.

Key idea: problem is much easier if A and B are sorted!

Counting Inversions: Conquer Step

Counting Inversions: Final Algorithm

Algorithm: Sort and count

input : The list L of elements
output: The number of inversion and the sorted list L

```
if |L| = 1 then
    there is no inversions;
else
    Divide the list into two halves: A and B;
    (r_A, A) = sort-and-count(A);
    (r_B, B) = sort-and-count(B);
    (r, L) = merge-and-count(A, B);
end
r = r_A + r_B + r
```

Counting Inversions: Final Algorithm

Algorithm: Sort and count

input : The list L of elements
output: The number of inversion and the sorted list L

if $|L| = 1$ **then**
  there is no inversions;
else
  Divide the list into two halves: A and B;
  $(r_A, A) = \text{sort-and-count}(A)$;
  $(r_B, B) = \text{sort-and-count}(B)$;
  $(r, L) = \text{merge-and-count}(A, B)$;
end
$r = r_A + r_B + r$

Running time $T(n)$ of the algorithm is $O(n \log n)$ because

$$T(n) \leq 2T(n/2) + O(n).$$

Algorithm design and analysis

— Some exercises —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Finding element

Let A be an array with n numbers. Design a divide-and-conquer algorithm for finding the position of the largest element in the array A.

11	12	7	4	8	5	9	3

Finding element

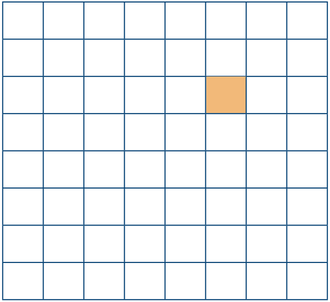
Let A be an array with n numbers. Design a divide-and-conquer algorithm for finding both the smallest and largest elements in the array A.

11	12	7	4	8	5	9	3

Tromino puzzle

Tromino puzzle

A tromino is an L-shaped tile formed by adjacent 1-by-1 squares. The problem is to cover any 2^n-by-2^n chessboard with one missing square (anywhere on the board) with trominoes. Trominoes should cover all the squares of the board except the missing one with no overlaps.



Algorithm design and analysis

— Integer Multiplication —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Integer Multiplication

MULTIPLY INTEGERS

INSTANCE Two n-digit binary integers x and y

SOLUTION The product xy

Multiply Integers

INSTANCE Two n-digit binary integers x and y

SOLUTION The product xy

- Multiply two n-digit integers.

Integer Multiplication

Multiply Integers

INSTANCE Two n-digit binary integers x and y

SOLUTION The product xy

- Multiply two n-digit integers.
- Result has at most $\boxed{2n}$ digits.

Integer Multiplication

MULTIPLY INTEGERS

INSTANCE Two n-digit binary integers x and y

SOLUTION The product xy

- Multiply two n-digit integers.
- Result has at most $2n$ digits.
- Algorithm we learnt in school takes

$$
\begin{array}{r}
12 \\
\times\ 13 \\
\hline
36 \\
12 \\
\hline
156
\end{array}
\qquad\qquad
\begin{array}{r}
1100 \\
\times\ 1101 \\
\hline
1100 \\
0000 \\
1100 \\
1100 \\
\hline
10011100
\end{array}
$$

Integer Multiplication

MULTIPLY INTEGERS

INSTANCE Two n-digit binary integers x and y

SOLUTION The product xy

- Multiply two n-digit integers.
- Result has at most $2n$ digits.
- Algorithm we learnt in school takes $O(n^2)$ operations. Size of the input is not 2 but $2n$

$$
\begin{array}{r}
12 \\
\times\, 13 \\
\hline
36 \\
12 \\
\hline
156
\end{array}
\qquad\qquad
\begin{array}{r}
1100 \\
\times\, 1101 \\
\hline
1100 \\
0000 \\
1100 \\
1100 \\
\hline
10011100
\end{array}
$$

Divide-and-Conquer Algorithm

- Assume integers are binary.
- Let us use divide and conquer

Divide-and-Conquer Algorithm

- Assume integers are binary.
- Let us use divide and conquer by splitting each number into first $n/2$ bits and last $n/2$ bits.
- Let x be split into x_0 (lower-order bits) and x_1 (higher-order bits) and y into y_0 (lower-order bits) and y_1 (higher-order bits).

$$xy =$$

Divide-and-Conquer Algorithm

- Assume integers are binary.
- Let us use divide and conquer by splitting each number into first $n/2$ bits and last $n/2$ bits.
- Let x be split into x_0 (lower-order bits) and x_1 (higher-order bits) and y into y_0 (lower-order bits) and y_1 (higher-order bits).

$$\begin{aligned} xy &= (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0) \\ &= x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0. \end{aligned}$$

Divide-and-Conquer Algorithm

- Assume integers are binary.
- Let us use divide and conquer by splitting each number into first $n/2$ bits and last $n/2$ bits.
- Let x be split into x_0 (lower-order bits) and x_1 (higher-order bits) and y into y_0 (lower-order bits) and y_1 (higher-order bits).

$$\begin{aligned} xy &= (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0) \\ &= x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0. \end{aligned}$$

- Each of x_1, x_0, y_1, y_0 has $n/2$ bits, so we can compute $x_1 y_1$, $x_1 y_0$, $x_0 y_1$, and $x_0 y_0$ recursively, and merge the answers in $O(n)$ time.

Divide-and-Conquer Algorithm

- Assume integers are binary.
- Let us use divide and conquer by splitting each number into first $n/2$ bits and last $n/2$ bits.
- Let x be split into x_0 (lower-order bits) and x_1 (higher-order bits) and y into y_0 (lower-order bits) and y_1 (higher-order bits).

$$
\begin{aligned}
xy &= (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0) \\
&= x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0.
\end{aligned}
$$

- Each of x_1, x_0, y_1, y_0 has $n/2$ bits, so we can compute $x_1 y_1$, $x_1 y_0$, $x_0 y_1$, and $x_0 y_0$ recursively, and merge the answers in $O(n)$ time.
- What is the running time $T(n)$?

Divide-and-Conquer Algorithm

- Assume integers are binary.
- Let us use divide and conquer by splitting each number into first $n/2$ bits and last $n/2$ bits.
- Let x be split into x_0 (lower-order bits) and x_1 (higher-order bits) and y into y_0 (lower-order bits) and y_1 (higher-order bits).

$$\begin{aligned} xy &= (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0) \\ &= x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0. \end{aligned}$$

- Each of x_1, x_0, y_1, y_0 has $n/2$ bits, so we can compute $x_1 y_1$, $x_1 y_0$, $x_0 y_1$, and $x_0 y_0$ recursively, and merge the answers in $O(n)$ time.
- What is the running time $T(n)$?

$$T(n) \leq 4T(n/2) + cn$$

Divide-and-Conquer Algorithm

- Assume integers are binary.
- Let us use divide and conquer by splitting each number into first $n/2$ bits and last $n/2$ bits.
- Let x be split into x_0 (lower-order bits) and x_1 (higher-order bits) and y into y_0 (lower-order bits) and y_1 (higher-order bits).

$$\begin{aligned} xy &= (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0) \\ &= x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0. \end{aligned}$$

- Each of x_1, x_0, y_1, y_0 has $n/2$ bits, so we can compute $x_1 y_1$, $x_1 y_0$, $x_0 y_1$, and $x_0 y_0$ recursively, and merge the answers in $O(n)$ time.
- What is the running time $T(n)$?

$$\begin{aligned} T(n) &\leq 4T(n/2) + cn \\ &\leq O(n^2) \end{aligned}$$

Improving the Algorithm

- Four sub-problems lead to an $O(n^2)$ algorithm.

Improving the Algorithm

- Four sub-problems lead to an $O(n^2)$ algorithm.

Improving the Algorithm

- Four sub-problems lead to an $O(n^2)$ algorithm.

- Four sub-problems lead to an $O(n^2)$ algorithm.
- What is the running time $T(n)$?

Final Algorithm

Algorithm design and analysis

— Closest Pair of Points —

Silvio Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2023

Computational Geometry

- Algorithms for geometric objects: points, lines, segments, triangles, spheres, polyhedra, ldots.
- Started in 1975 by Shamos and Hoey.
- Problems studied have applications in a vast number of fields: ecology, molecular biology, statistics, computational finance, computer graphics, computer vision, . . .

Computational Geometry

- Algorithms for geometric objects: points, lines, segments, triangles, spheres, polyhedra, ldots.
- Started in 1975 by Shamos and Hoey.
- Problems studied have applications in a vast number of fields: ecology, molecular biology, statistics, computational finance, computer graphics, computer vision, ...

Closest Pair of Points

INSTANCE A set P of n points in the plane

SOLUTION The pair of points in P that are the closest to each other.

Computational Geometry

- Algorithms for geometric objects: points, lines, segments, triangles, spheres, polyhedra, ldots.
- Started in 1975 by Shamos and Hoey.
- Problems studied have applications in a vast number of fields: ecology, molecular biology, statistics, computational finance, computer graphics, computer vision, ...

Closest Pair of Points

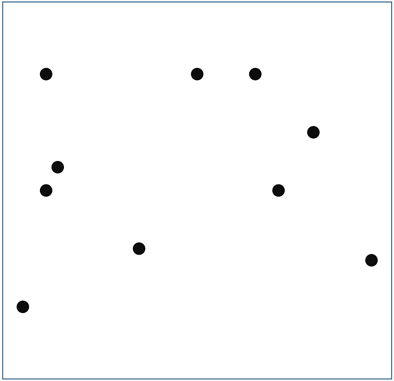
INSTANCE A set P of n points in the plane

SOLUTION The pair of points in P that are the closest to each other.

- At first glance, it seems any algorithm must take $\Omega(n^2)$ time.
- Shamos and Hoey figured out an ingenious $O(n \log n)$ divide and conquer algorithm.

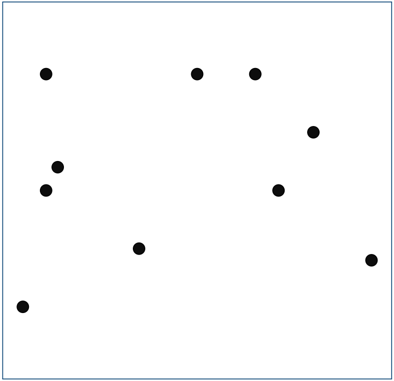
Closest Pair: Set-up

- Let $P = \{p_1, p_2, \ldots, p_n\}$ with $p_i = (x_i, y_i)$.



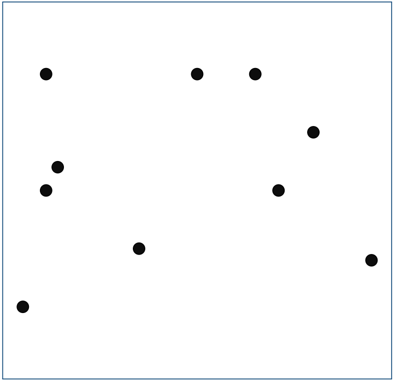
Closest Pair: Set-up

- Let $P = \{p_1, p_2, \dots, p_n\}$ with $p_i = (x_i, y_i)$.
- Use $d(p_i, p_j)$ to denote the Euclidean distance between p_i and p_j.



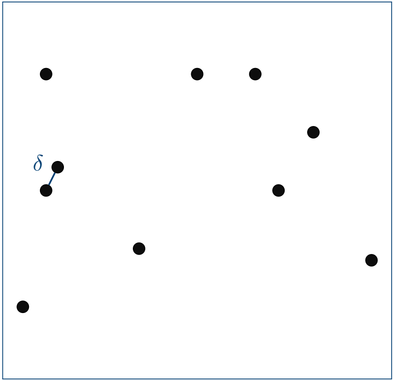
Closest Pair: Set-up

- Let $P = \{p_1, p_2, \ldots, p_n\}$ with $p_i = (x_i, y_i)$.
- Use $d(p_i, p_j)$ to denote the Euclidean distance between p_i and p_j.
- Goal: find the pair of points p_i and p_j that minimize $d(p_i, p_j)$.



Closest Pair: Set-up

- Let $P = \{p_1, p_2, \ldots, p_n\}$ with $p_i = (x_i, y_i)$.
- Use $d(p_i, p_j)$ to denote the Euclidean distance between p_i and p_j.
- Goal: find the pair of points p_i and p_j that minimize $d(p_i, p_j)$.



Closest Pair: Algorithm Skeleton

1. Divide P into two sets Q and R of $n/2$ points such that each point in Q has x-coordinate less than any point in R.

Closest Pair: Algorithm Skeleton

1. Divide P into two sets Q and R of $n/2$ points such that each point in Q has x-coordinate less than any point in R.
2. Recursively compute closest pair in Q and in R, respectively.

Closest Pair: Algorithm Skeleton

1. Divide P into two sets Q and R of $n/2$ points such that each point in Q has x-coordinate less than any point in R.
2. Recursively compute closest pair in Q and in R, respectively.
3. Let δ_1 be the distance computed for Q, δ_2 be the distance computed for R, and $\delta = \min(\delta_1, \delta 2)$.

Closest Pair: Algorithm Skeleton

1. Divide P into two sets Q and R of $n/2$ points such that each point in Q has x-coordinate less than any point in R.
2. Recursively compute closest pair in Q and in R, respectively.
3. Let δ_1 be the distance computed for Q, δ_2 be the distance computed for R, and $\delta = \min(\delta_1, \delta 2)$.
4. Compute pair (q, r) of points such that $q \in Q$, $r \in R$, $d(q, r) < \delta$ and $d(q, r)$ is the smallest possible.
   - How do we implement this step in $O(n)$ time?

Closest Pair: Algorithm Skeleton

1. Divide P into two sets Q and R of $n/2$ points such that each point in Q has x-coordinate less than any point in R.
2. Recursively compute closest pair in Q and in R, respectively.
3. Let δ_1 be the distance computed for Q, δ_2 be the distance computed for R, and $\delta = \min(\delta_1, \delta 2)$.
4. Compute pair (q, r) of points such that $q \in Q$, $r \in R$, $d(q, r) < \delta$ and $d(q, r)$ is the smallest possible.
   - How do we implement this step in $O(n)$ time?

Assignment

Implement the problem to find the closest pair in a plane.