

# Divide and Conquer Algorithms

---

CS 336: Design and Analysis of Algorithms. Chapter 5 in [KT].  
©Konstantin Makarychev

---

## Divide and Conquer

- Solve problem in three steps:
  - Partition your problem in two or more subproblems.
  - Solve each of them separately.
  - Combine solutions.

# **Sorting**

---

## **Merge Sort**

---

# The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page

*Computer Science Department,  
Stanford University, Stanford, CA 94305, USA  
sergey@cs.stanford.edu and page@cs.stanford.edu*

## 4.2.3 Document Index

The document index keeps information about each document. It is a fixed width ISAM (Index sequential access mode) index, ordered by docID. The information stored in each entry includes the current document status, a pointer into the repository, a document checksum, and various statistics. If the document has been crawled, it also contains a pointer into a variable width file called docinfo which contains its URL and title. Otherwise the pointer points into the URLlist which contains just the URL. This design decision was driven by the desire to have a reasonably compact data structure, and the ability to fetch a record in one disk seek during a search

Additionally, there is a file which is used to convert URLs into docIDs. It is a list of URL checksums with their corresponding docIDs and is sorted by checksum. In order to find the docID of a particular URL, the URL's checksum is computed and a binary search is performed on the checksums file to find its docID. URLs may be converted into docIDs in batch by doing a merge with this file. This is the technique the URLresolver uses to turn URLs into docIDs. This batch mode of update is crucial because otherwise we must perform one seek for every link which assuming one disk would take more than a month for our 322 million link dataset.

## Merge Sort

**Given:** Sequence  $a_1, \dots, a_n$ .

**Goal:** Return sorted sequence.

# Merge Sort

**Given:** Sequence  $a_1, \dots, a_n$ .

**Goal:** Return sorted sequence.

```
function MergeSort (array A)
```

1. Split array  $A$  into left and right:  $L$  and  $R$ .
2. Sort  $L$  and  $R$ .
3. Combine or merge solutions.

# Merge Sort

**Given:** Sequence  $a_1, \dots, a_n$ .

**Goal:** Return sorted sequence.

```
function MergeSort (array A)
```

1. Split\* array  $A$  into  $L = a_1, \dots, a_k$  and  $R = a_{k+1}, \dots, a_n$  for  $k = \lfloor n/2 \rfloor$ , where  $n = |A|$ .
2. MergeSort( $L$ ) and MergeSort( $R$ ).
3.  $A = \text{Merge}(L, R)$ .

\*If  $n = 1$  i.e.,  $A$  contains one element, return  $A$  right away.

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

append  $L[i]$  to A;

$i = i + 1$

else

append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

append  $L[i]$  to A;

$i = i + 1$

else

append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17
1			

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

    append  $L[i]$  to A;

$i = i + 1$

else

    append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17
1			

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

    append  $L[i]$  to A;

$i = i + 1$

else

    append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17
1	3		

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

    append  $L[i]$  to A;

$i = i + 1$

else

    append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17
1	3		

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

    append  $L[i]$  to A;

$i = i + 1$

else

    append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17
1	3	4	

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

    append  $L[i]$  to A;

$i = i + 1$

else

    append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17
1	3	4	

```
function Merge(array L and R)
```

Create new array A.

$i = 0; j = 0;$

while ( $i < L.size()$  and  $j < R.size()$ )

if  $L[i] \leq R[j]$

    append  $L[i]$  to A;

$i = i + 1$

else

    append  $R[j]$  to A;

$j = j + 1$

return A

1	10	15	20
3	4	11	17
1	3	4	10

## Correctness of Merge

**Claim:** The running time of *Merge* is  $O(|L| + |R|)$ .

- At every step of the while loop  $i$  or  $j$  is incremented by 1.  
Hence, the total number of steps is  $|L| + |R|$ .

**Claim:** If  $L$  and  $R$  are sorted, then  $A$  is sorted.

**Proof:** Observe that each number appended to  $A$  is less than or equal to every remaining number in  $L$  and  $R$ .

- Thus,  $A$  is sorted.

## Running Time

Assume  $n = 2^d$ .

- $T(n) = 2T(n/2) + f(n)$ , where  $f(n) = O(n)$ .

## Running Time

Assume  $n = 2^d$ .

- $T(n) = f(n) + 2T(n/2)$ , where  $f(n) = O(n)$ .
  - $T(n) = f(n) + 2f\left(\frac{n}{2}\right) + 4T\left(\frac{n}{4}\right)$ .
  - $T(n) = f(n) + 2f\left(\frac{n}{2}\right) + 4f\left(\frac{n}{4}\right) + 8T\left(\frac{n}{8}\right)$ .
  - ...
- 

## Running Time

Assume  $n = 2^d$ .

- $T(n) = f(n) + 2T(n/2)$ , where  $f(n) = O(n)$ .
  - $T(n) = f(n) + 2f\left(\frac{n}{2}\right) + 4T\left(\frac{n}{4}\right)$ .
  - $T(n) = f(n) + 2f\left(\frac{n}{2}\right) + 4f\left(\frac{n}{4}\right) + 8T\left(\frac{n}{8}\right)$ .
  - $T(n) = f(n) + 2f\left(\frac{n}{2}\right) + 4f\left(\frac{n}{4}\right) + \dots + 2^d f(1) = \log_2 n \cdot O(n) = O(n \log n)$ .
-

## Running Time

- **Claim:** If  $T(n) = 2T\left(\frac{n}{2}\right) + f(n)$ , where  $f(n) = O(n)$ , then  $T(n) = O(n \log n)$ .
- 

## Running Time

- **Claim:** If  $T(n) = 2T\left(\frac{n}{2}\right) + f(n)$ , where  $f(n) = O(n)$ , then  $T(n) = O(n \log n)$ .
  - **Claim:** If  $T(n) = 2T\left(\frac{n}{2}\right) + f(n)$  and  $T(1) = f(1)$ , where  $f(n) \leq cn$ , then  $T(n) \leq c n (\log_2 n + 1)$  for  $n \geq 2$ .
-

## Running Time

- **Claim:** If  $T(n) = 2T\left(\frac{n}{2}\right) + f(n)$  and  $T(1) = f(1)$ , where  $f(n) \leq cn$ , then  $T(n) \leq c n (\log_2 n + 1)$  for  $n \geq 2$ .
- **Proof by induction on  $n$ .**
- Base case  $n = 1$ :  $T(1) = f(1) = c$ .
- Inductive step:

$$T(n) = 2T\left(\frac{n}{2}\right) + f(n) \leq 2c \cdot \frac{n}{2} \left( \log_2 \frac{n}{2} + 1 \right) + cn.$$

---

## Running Time

- **Claim:** If  $T(n) = 2T\left(\frac{n}{2}\right) + f(n)$  and  $T(1) = f(1)$ , where  $f(n) \leq cn$ , then  $T(n) \leq c n (\log_2 n + 1)$  for  $n \geq 2$ .
- **Proof by induction on  $n$ .**
- Base case  $n = 1$ :  $T(1) = f(1) = c$ .
- Inductive step:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + f(n) \leq cn \left( \log_2 \frac{n}{2} + 1 \right) + cn = \\ &= cn(\log_2 n + 1). \end{aligned}$$

---

# Fast Multiplication

Multiplying long numbers is a bit harder...

$$\begin{array}{r} & & 1 & 0 & 1 & 1 & 0 \\ & & 1 & 1 & 0 & 1 & 1 \\ \hline & & 1 & 0 & 1 & 1 & 0 \\ & & 1 & 0 & 1 & 1 & 0 \\ & & 0 & 0 & 0 & 0 & 0 \\ & & 1 & 0 & 1 & 1 & 0 \\ \hline & & 1 & 0 & 1 & 1 & 0 \\ & & 1 & 0 & 1 & 0 & \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Use grade school multiplication algorithm!

## Divide and Conquer

- The running time of the grade school algorithm is  $O(n^2)$ , where  $n$  is the length of the numbers i.e.  $n = \max(\lceil \log_2 a \rceil, \lceil \log_2 b \rceil)$ .
- Can we do better?
- Write:  $a = 2^k a_1 + a_2$  and  $b = 2^k b_1 + b_2$ .

$$a = \underbrace{101}_{a_1} \underbrace{111}_{a_2} \text{ and } b = \underbrace{110}_{b_1} \underbrace{100}_{b_2}$$

## Divide and Conquer

- Write:  $a = 2^k a_1 + a_2$  and  $b = 2^k b_1 + b_2$ .

$$a = \underbrace{101}_{a_1} \underbrace{111}_{a_2} \text{ and } b = \underbrace{110}_{b_1} \underbrace{100}_{b_2}$$

$$ab = 2^{2k} a_1 b_1 + 2^k a_1 b_2 + 2^k a_2 b_1 + a_2 b_2$$

- **Attempt 1:** Let  $k = n/2$ . Compute  $a_1 b_1, a_1 b_2, a_2 b_1, a_2 b_2$ .
- Then,  $T(n) = 4T(n/2) + O(1)$ . Hence,  $T(n) = O(n^2)$ . ☹

КИБЕРНЕТИКА И ТЕОРИЯ РЕГУЛИРОВАНИЯ

А. КАРАЦУБА И Ю. ОФМАН

УМНОЖЕНИЕ МНОГОЗНАЧНЫХ ЧИСЕЛ НА АВТОМАТАХ

(Представлено академиком А. Н. Колмогоровым 13.11.1962)

Одной из задач настоящей заметки, так же как ранее опубликованной заметки Ю. Офмана<sup>(1)</sup>, обозначения и определения которой используются также здесь, является рассмотрение проблемы нахождения нижних оценок алгоритмической сложности дискретных функций. Пока никаких нетривиальных оценок снизу не существует. Весьма вероятно, что возможна теория, которая одним методом оценивает снизу число операций над числами при умножении многозначных чисел и количеством операций над числами при решении системы линейных уравнений и пр.

Два  $m$ -значных двоичных числа помещаются на вход двоичного автомата (вход из  $k = 2m$  звеньев). На выходе из  $2m + 1$  звеньев должна быть получена оценка производимого произведения. Для определения таким образом функции  $y = d_s$  (здесь  $s$  элементарно получается нижним оценкой сложности реализующих ее двоичных автоматов):  $N_s \geq m$ ,  $T_s \geq \log_2 m$ .

Далее излагается схема доказательства двух теорем.

Теорема 1 (Офман). При любом  $s$ ,  $1 < s < m$ , функция  $d_s$  может быть реализована двоичным автоматом с характеристиками, имеющими при  $m \rightarrow \infty$  (разномерно по  $s$  в указанных пределах) характеристики

$$N_s \asymp \frac{m^s}{3}, T_s \asymp s \log_2 m.$$

При  $s = 1$  получаем автомат с характеристиками

$$N \asymp m^2, T \asymp \log_2 m, \quad (1)$$

при  $s = m$  с характеристиками

$$N \asymp m, T \asymp m \log_2 m. \quad (2)$$

Теорема 2 (Карацуба). Функция  $d_s$  может быть реализована двоичным автоматом с характеристиками

$$N_s \asymp m, T_s \asymp \log_2 m.$$

Далее этих результатов авторам заметки продвинуться не удалось. Очевидно, что  $N_s$  и  $T_s$  удовлетворяют оценкам

$$N_s \asymp m, T_s \asymp \log_2 m,$$

но неизвестно, соединим ли такие порядки роста  $N$  и  $T$  между собой.

Обычный школьный способ умножения, измененный лишь в том, что произведение множимого на каждый разряд множителя получаются параллельно и складываются автомтом 3, приводит к оценкам (1). Если же образовывать произведение множимого на отдельные знаки множителя последовательно и прибавлять их к накапленной сумме ранее образованных про-

изведений, пользуясь многократно автомтом для сложения теоремы 2 (см. (2)), то можно привести к оценке (2). Вспомогательное устройство, которое последовательно выводит в сумматоры новые накапливаемые единицы, конструктуется без больших затруднений в пределах требуемой оценки (2).

Для получения автомата, существование которого утверждается в теореме 1, множитель делится на группы разрядов по  $s$  разрядов в группе. Умножение на знаки множителя из одной разрядной группы производится последовательно, а сложение результатов умножения на каждую разрядную группу производится параллельно.

Для доказательства теоремы 2 заметим, что умножение можно заменить сложением и возведением в квадрат:  $ab = \frac{1}{4}[(a+b)^2 - (a-b)^2]$ . Деление на четверть не представляет больших затруднений в двоичной системе счисления. Таким образом, оказывается достаточным оценить порядки роста  $N$  и  $T$  для функции  $y = d_s(x)$ , соответствующей возведению в квадрат  $2m$ -значного двоичного числа.

$$(x_1 x_2 \dots x_m)^2 = x_1 2^{2m-1} + x_2 2^{2m-2} + \dots + x_m.$$

Формул

$$\begin{aligned} (x_1 x_2 \dots x_m)^2 &= 2^{m-4} [(x_1 x_2 \dots x_m) + (x_{m+1} \dots x_{2m})]^2 + \\ &+ (2^{2m} - 2^{m-4}) (x_1 x_2 \dots x_m)^2 + (1 - 2^{m-4}) (x_{m+1} x_{m+2} \dots x_{2m})^2 \end{aligned}$$

показывает, что возведение в квадрат  $2m$ -значного числа сводится к трем возведениям в квадрат  $m$ -значных чисел<sup>\*</sup> и операции (сложение, умножение на степень двойки), осуществление которых можно произвести весьма экономно, пользуясь приемами, указанными в (1).

Лемма. Если возведение в квадрат  $r$ -значного числа можно произвести автомтом с  $N = N_r$ ,  $T = T_r$ , то для возведения в квадрат  $2^{r+1}$ -значного числа можно построить автомтом с  $N = N_{r+1} = 3N_r + c \cdot 2^r$ ,  $T = T_{r+1} = T_r + c \cdot r$ .

При помощи леммы легко проводится индуктивное доказательство теоремы 2.

Для представлений автоматаами без обратных связей (суперпозиции) зерна теорема 2 и частный случай теоремы 1, соответствующий формуле (1).

Поступило

9.11.1962

ЦИТИРОВАННАЯ ЛИТЕРАТУРА

<sup>1</sup> Ю. Офман, ДАН, 145, № 1 (1962).

\* Сумма  $(x_1 x_2 \dots x_m) + (x_{m+1} \dots x_{2m})$  может иметь  $m+1$  знаков, но сведение в квадрат  $(m+1)$ -значного числа к возведению в квадрат  $m$ -значного числа делается при помощи формулы  $(2a+b)^2 = 4a^2 + 4ab + b^2$ , где  $b = 0,1$ .



Anatoly Karatsuba

## Karatsuba Algorithm

- Write:  $a = 2^k a_1 + a_2$  and  $b = 2^k b_1 + b_2$ . Let  $k = n/2$ .

$$ab = 2^{2k} a_1 b_1 + 2^k (a_1 b_2 + a_2 b_1) + a_2 b_2$$

- How to compute  $a_1 b_1$ ,  $a_2 b_2$ , and  $(a_1 b_2 + a_2 b_1)$ ?

## Karatsuba Algorithm

- Write:  $a = 2^k a_1 + a_2$  and  $b = 2^k b_1 + b_2$ . Let  $k = n/2$ .

$$ab = 2^{2k} a_1 b_1 + 2^k (a_1 b_2 + a_2 b_1) + a_2 b_2$$

- Compute  $a_1 b_1$ ,  $a_2 b_2$ , and  $(a_1 + a_2)(b_1 + b_2)$ , then

$$a_1 b_2 + a_2 b_1 = (a_1 + a_2)(b_1 + b_2) - (a_1 b_1 + a_2 b_2)$$

- Now, we use 3 multiplications of numbers of length  $n/2$  and  $O(1)$  additions. Hence,

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

## How to solve the recurrence?

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

- You can use “Master Theorem”.
- Or guess the answer: Let  $n = 2^d$  and  $f_d = T(2^d)$ . Then,

$$f_d \leq 3f_{d-1} + c2^d$$

- Let’s drop the last term  $c2^d$  for now. Then,

$$\hat{f}_d \leq 3\hat{f}_{d-1} \Rightarrow \hat{f}_d \leq 3^d \hat{f}_0$$

# How to solve the recurrence?

- Let  $n = 2^d$  and  $f_d = T(2^d)$ . Then,

$$f_d \leq 3f_{d-1} + c2^d$$

- We want to find  $g_d$  s.t.  $f_d \leq g_d 3^d$  for all  $d$ . Suppose,  $f_{d-1} \leq g_{d-1} 3^{d-1}$ .

$$\begin{aligned} f_d &\leq 3f_{d-1} + c2^d \leq 3g_{d-1} 3^{d-1} + c2^d \\ &= g_{d-1} 3^d + c2^d = (g_{d-1} + c (2/3)^d) 3^d \leq? g_d 3^d \end{aligned}$$

We need  $g_d \leq g_{d-1} + c (2/3)^d$ .

- Let's set  $g_d = g_0 + c \sum_{i=1}^d (2/3)^i \leq g_0 + 2c$ .

---

\*c.f. with the method of *variation of parameters* in differential equations.

## Running Time

The running time is

$$T(n) = f_{\log_2 n} \leq g 3^{\log_2 n} = g 2^{\log_2 3 \cdot \log_2 n} = O(n^{\log_2 3})$$

Note that

$$\log_2 3 \leq 1.585$$

Hence,  $T(n) = O(n^{1.585})$ .

---

# **Closest Points in the Plane**

---

# **Closest Points in the Plane**

---

CS 336: Design and Analysis of Algorithms. Chapter 5 in [KT].  
©Konstantin Makarychev

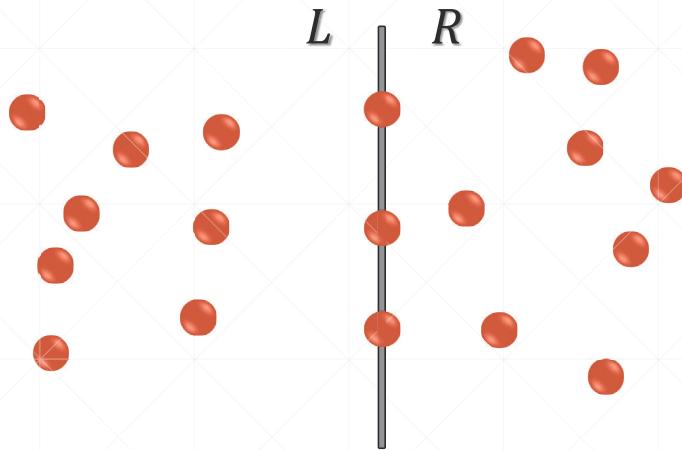
# Closest Points in the Plane

**Given:** Given a set of points  $(x_i, y_i)$ .

**Goal:** Find the pair of closest points.

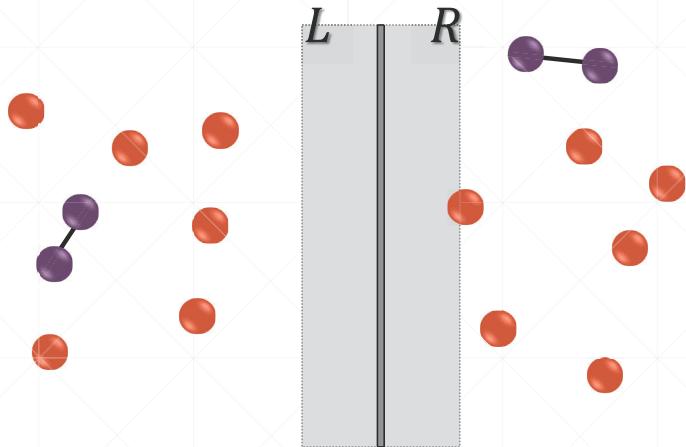
1. Divide points into two groups  $L$  and  $R$ .
  2. Find closest pairs in  $L$  and  $R$ :
    - a.  $[p_L, q_L, \delta_L] = \text{Closest\_Pair}(L);$
    - b.  $[p_R, q_R, \delta_R] = \text{Closest\_Pair}(R);$
  3. Combine solutions.
- 

## Divide into two equal sets $L$ and $R$



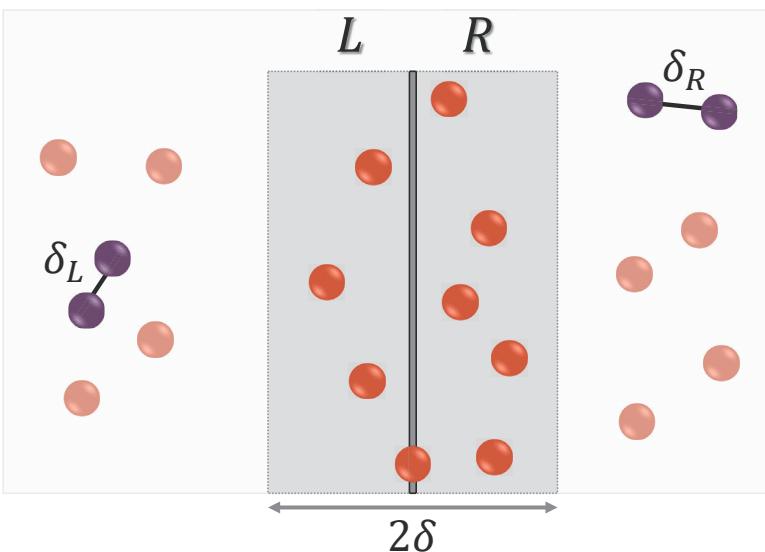
- Sort all points  $(x_i, y_i)$ .
- Pick the median  $\mu$  s.t.
- $L = \{(x_i, y_i) : x_i < \mu\}$
- $R = \{(x_i, y_i) : x_i > \mu\}$  have size at most  $n/2$ .

## Find closest points in $L$ and $R$



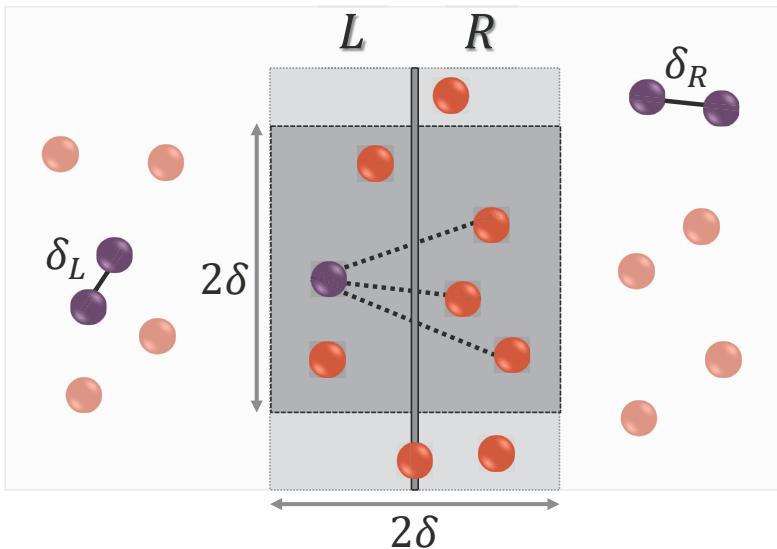
- Find closest pairs of points  $(p_L, q_L)$  and  $(p_R, q_R)$  in  $L$  and in  $R$ .
- Can we return the closest pair among them?
- Yes, if all points are far from the middle line.

## Find closest points in $2\delta$ -strip



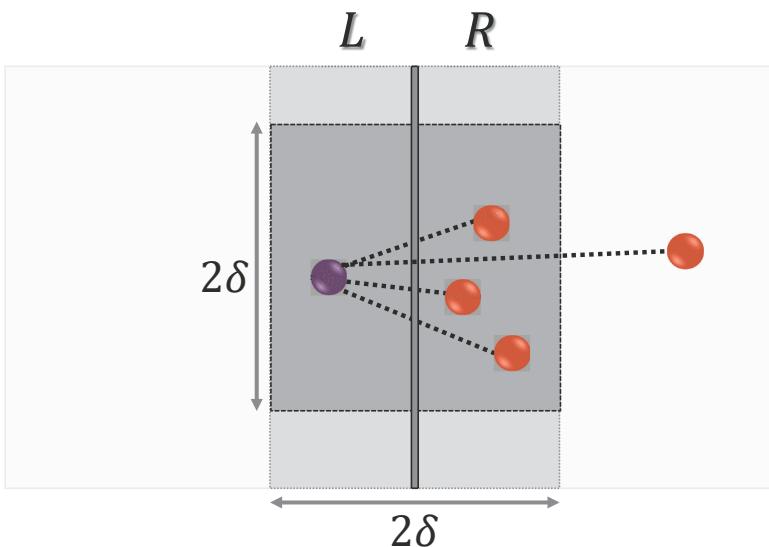
- Find the closest points in  $2\delta$ -strip on the opposite sides of the cut.
- Can we compare all pairs in  $2\delta$ -strip?
- No: Running time  $\sim n^2$ .

## Find closest points in $2\delta$ -strip



- Find the closest points in  $2\delta$ -strip on the opposite sides of the cut.
- Compare each vertex in the  $\delta$ -strip in  $L$  with every vertex in the the  $\delta$ -strip  $R$ .

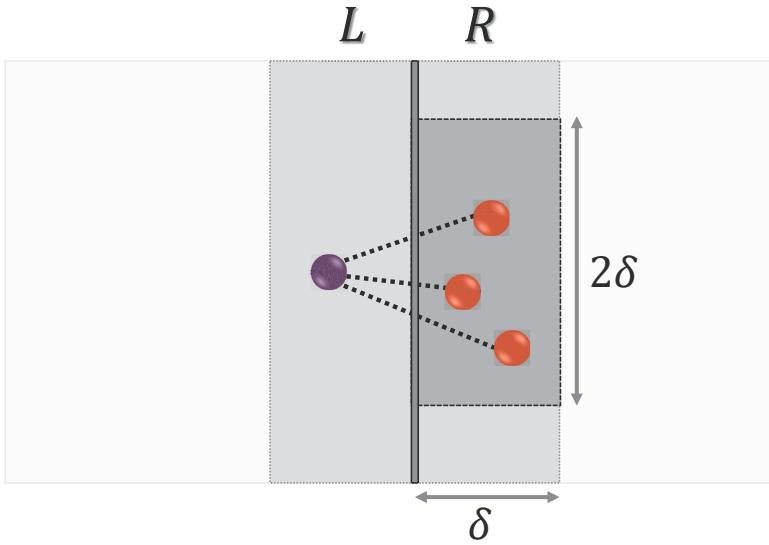
## Find closest points in $\delta$ -strip



### Correctness:

- Any point outside of the  $2\delta$ -strip is at distance at least  $\delta$  from the opposite side.
- Thus, the algorithm finds the closest points.

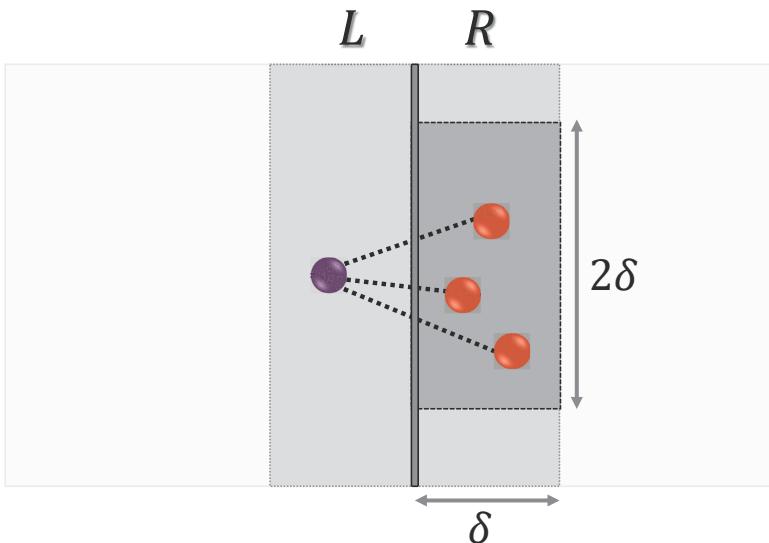
## Find closest points in $\delta$ -strip



**Running time:**

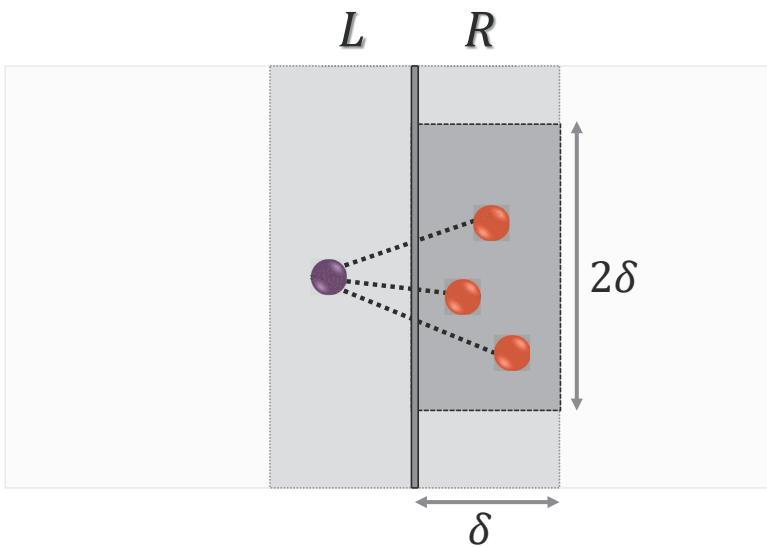
- How many points can lie in the  $\delta \times 2\delta$ -box in  $R$ ?
- How to find these points efficiently?

## How many points can lie in the $\delta \times 2\delta$ -box in $R$ ?



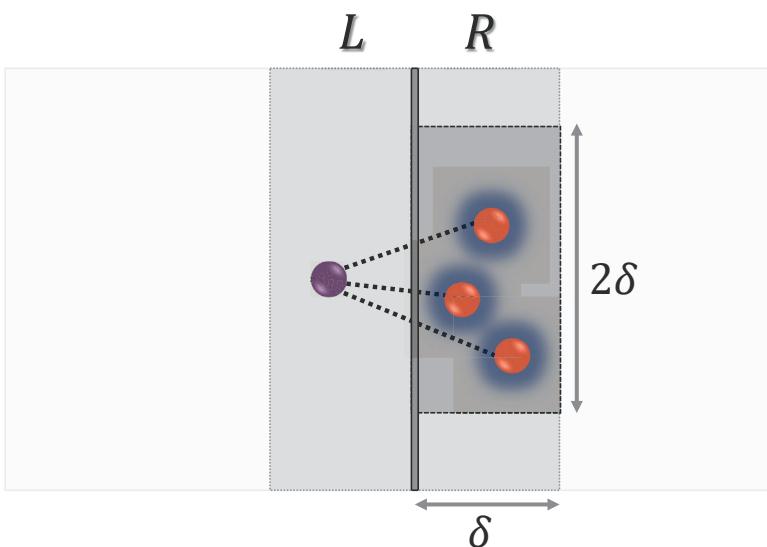
- How many points can lie in the  $\delta \times 2\delta$ -box in  $R$ ?
- There are no two points in the  $\delta$ -strip at distance  $< \delta$ . Why?

## How many points can lie in the $\delta \times 2\delta$ -box in $R$ ?



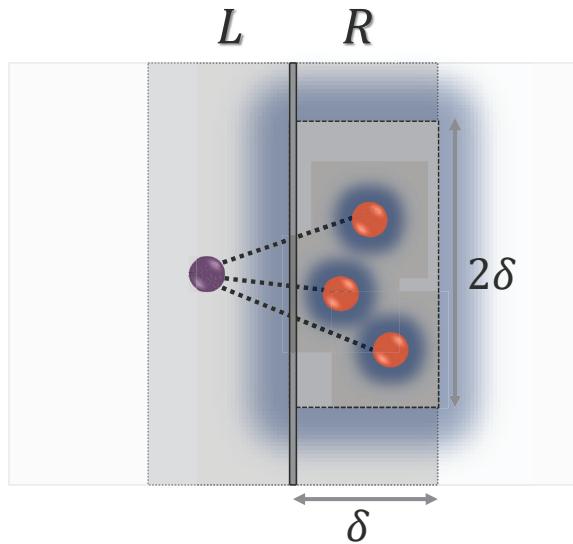
- There are no two points in the  $\delta$ -strip in at distance  $< \delta$ . Why?
  - Because the minimum distance in  $R$  is  $\delta_R \geq \delta$ .

## How many points can lie in the $\delta \times 2\delta$ -box in $R$ ?



- There are no two points in the  $\delta$ -strip in at distance  $< \delta$ . Why?
  - Because the minimum distance in  $R$  is  $\delta_R \geq \delta$ .
- Thus, circles of radius  $\delta/2$  around points in  $R$  do not intersect (but can touch each other).

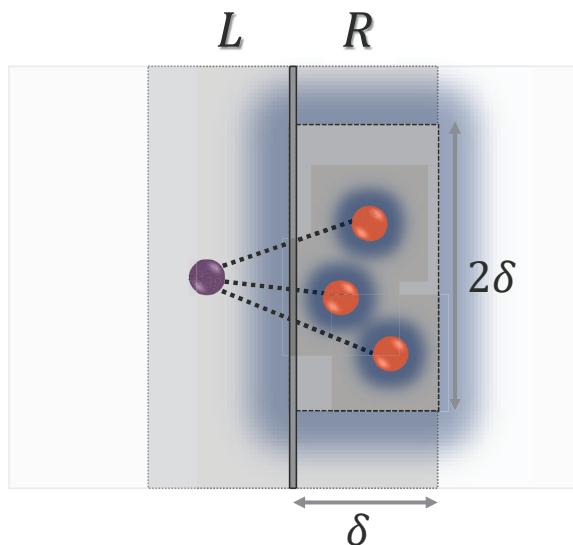
## How many points can lie in the $\delta \times 2\delta$ -box in $R$ ?



- Circles of radius  $\delta/2$  around points in  $R$  do not intersect.
- Some circled may leave the  $\delta \times 2\delta$  box
- ... but all of them lie in the box of size  $2\delta \times 3\delta$ .

$$\text{Area(Circle)} = \pi\delta^2/4$$
$$\text{Area(Box)} = 6\delta^2$$

## How many points can lie in the $\delta \times 2\delta$ -box in $R$ ?



- Circles of radius  $\delta/2$  around points in  $R$  do not intersect.
- ... but all of them lie in the box of size  $2\delta \times 3\delta$ .

$$\text{Area(Circle)} = \pi\delta^2/4$$
$$\text{Area(Box)} = 6\delta^2$$

$$\#\text{circles} = \#\text{points} \leq 7$$

## How to find points in the box quickly?

- The function `ClosestPair` receives two lists  $P_x$  and  $P_y$  of all points in the subproblem: one list is sorted by  $x$ -coordinate another by  $y$ -coordinate.
- To find the closest points in the  $\delta$ -strip we copy points from  $P_y$  that lie in the strip to a new list  $S_y$ . Now we slide a window in  $S_y$  of width  $2\delta$  i.e., we consider intervals  $[i, j]$  s.t.  $|S_y[i] - S_y[j]| \leq 2\delta$ .
- Total #operations is  $O(|P|)$ .

**function** `ClosestPair` ( $P, P_x, P_y$ )

Find the median  $\mu$  s.t.  $|L|, |R| \leq n/2$

Divide all points in  $P$  into two groups:

$$L = \{(x, y) \in P : x < \mu\} \text{ and } R = \{(x, y) \in P : x > \mu\}$$

Find sorted sets  $L_x, L_y$  and  $R_x, R_y$ .

Solve subproblems:

$$\delta_L = \text{ClosestPair}(L, L_x, L_y) \text{ and } \delta_R = \text{ClosestPair}(R, R_x, R_y)$$

Let  $\delta = \min(\delta_L, \delta_R)$ .

Find the closest points in the  $2\delta$ -strip. Let  $\delta_S$  be distance.

Return  $\min(\delta_L, \delta_R, \delta_S)$ .

## Running Time

- Sort the list of all points by  $x$  and by  $y$ :  $O(n \log n)$ .
- After that:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

The same recurrence as in Merge Sort.

Total running time is  $O(n \log n)$ .

---

## Questions?

---