알고리즘

01. Divide & Conquer

2014/09/12

미디어소프트웨어학과
민경하

Contents

0. Prologue

1. Divide & conquer

2. Graph

3. Greedy algorithm

4. Dynamic programming

1. Divide & Conquer

1.0 Introduction

1.1 Multiplication

1.2 Recurrence relation

1.3 Sorting

1.4 Medians

1.5 Matrix multiplication

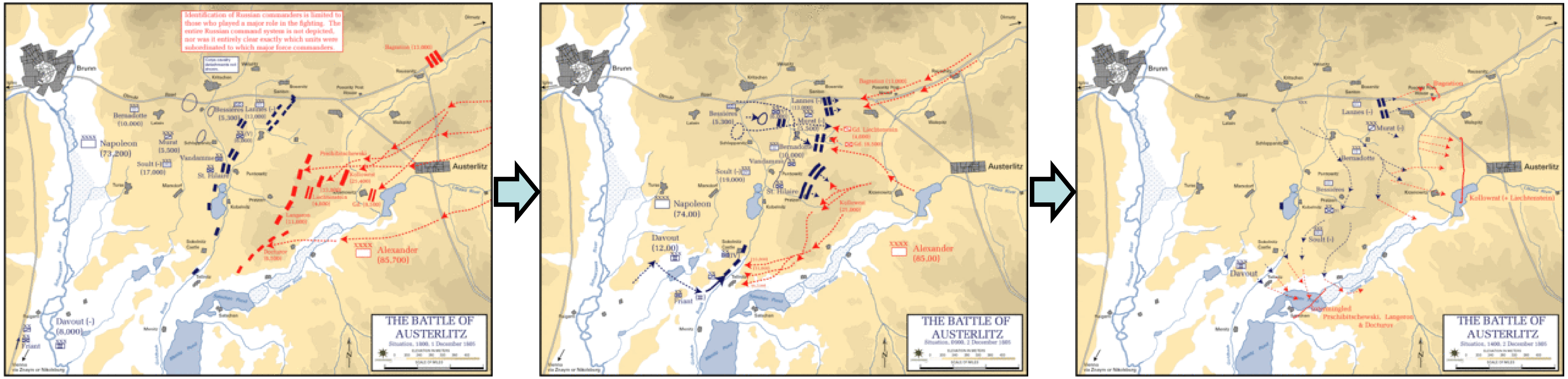
1.6 Finding closest pair

1.0 Introduction

- Battle of Austerlitz

1.0 Introduction

- Battle of Austerlitz (1805)
  - Battle of three emperors
  - France (65,000) VS Austria + Russia (80,000)
  - The end of the 3rd anti-France aliance



1.0 Introduction

- Key idea of divide & conquer
    - Solve a problem of *n* inputs by splitting the input into *k* subsets
- Three steps of divide & conquer
    - – Divide
        - Breaking a problem into subproblems
    - – Conquer
        - Recursively solving these subproblems
    - – Combine (optional)
        - Appropriately combining their answers

1.0 Introduction

- Abstract algorithm for DnC (recursive)

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

1.0 Introduction

- Most divide & conquer algorithms are implemented using recursive call
- Three check points
  - Same format
  - Reduced problem size
  - Degenerate case

1.0 Introduction

(1) Same format

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

1.0 Introduction

(2) Reduced problem size

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

1.0 Introduction

(3) Degenerate case

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

1.0 Introduction

- Performance analysis for DnC
  - T (n): time complexity of DnC ( ) for n inputs
  - g (n): for small input
  - f_1 (n): for DIVIDE ( )
  - f_2 (n): for COMBINE ( )

$$T(n) = \begin{cases} g(n), & \text{for small } n \\ 2T(n/2) + f_1(n) + f_2(n), & \text{otherwise} \end{cases}$$

$$T(n) = \begin{cases} g(n), & \text{for small } n \\ aT(n/b) + O(n^d), & \text{otherwise} \end{cases}$$

1.0 Introduction

- The simplest divide & conquer algorithm is
- Binary search:
  - Let $A = \{a_1, \ldots a_n\}$ be a list of elements which are sorted in nondecreasing order.
  - Determine whether x is in A or not. If x is in A, then find i such that $a_i = x$.

1.0 Introduction

- Straightforward search

```
int bsearch_straightforward ( int s, int e, int A[], int x )
{
    for ( i = s; i <= e; i++ ) {
        if ( A[i] == x )
            return i;
    }

    return NONE;
}
```

  - Performance analysis?

1.0 Introduction

- Binary search (Divide & Conquer)

```
int bsearch ( int s, int e, int A[], int x )
{
    if ( s == e )
        if ( A[s] == x )
            return s;
        else
            return NONE;
    m ← (s+e)/2;
    if ( A[m] == x )
        return m;
    else if ( A[m] > x )
        return bsearch ( s, m, A, x );
    else
        return bsearch ( m + 1, e, A, x );
}
```

1.0 Introduction

- Binary search → check three key points

```
int bsearch ( int s, int e, int A[], int x )
{
    if ( s == e )
        if ( A[s] == x )
            return s;
        else
            return NONE;
    m ← (s+e)/2;
    if ( A[m] == x )
        return m;
    else if ( A[m] > x )
        return bsearch ( s, m, A, x );
    else
        return bsearch ( m + 1, e, A, x );
}
```

1.0 Introduction

- Binary search (Divide & Conquer)
  - Performance analysis

$$T(n) = \begin{cases} 1, & \text{for } n = 1 \\ T(n/2) + 1, & \text{otherwise} \end{cases}$$

$$T(n) = \log n$$

1.1 Multiplication

- Multiplying two integers of n-digit.
  - u & v: n-digit integers
  - Time for adding u & v: O(n)

Carry:	1			1	1	1		
		1	1	0	1	0	1	(53)
		1	0	0	0	1	1	(35)
	1	0	1	1	0	0	0	(88)

1.1 Multiplication

- Multiplying two integers of n-digit.
  - Time for multiplying u & v: $O(n^2)$

$$
\begin{array}{rrrrrrrrl}
 & & & & 1 & 1 & 0 & 1 & \\
 & & & \times & 1 & 0 & 1 & 1 & \\
\hline
 & & & & 1 & 1 & 0 & 1 & \text{(1101 times 1)} \\
 & & & 1 & 1 & 0 & 1 & & \text{(1101 times 1, shifted once)} \\
 & & 0 & 0 & 0 & 0 & & & \text{(1101 times 0, shifted twice)} \\
+ & 1 & 1 & 0 & 1 & & & & \text{(1101 times 1, shifted thrice)} \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \text{(binary 143)}
\end{array}
$$

  - Can we improve it by divide & conquer?

1.1 Multiplication

- Gauss's original suggestion
  - $(a + b\,i)\,(c + d\,i) = ac - bd + (ad + bc)\,i$
  - How many multiplications?
  - Actually, 3 instead of 4
    - $ad + bc = (a + b)(c + d) - ac - bd.$

1.1 Multiplication

- Two binary numbers x & y with length n

$$x = [x_L][x_R] = 2^s x_L + x_R$$

$$y = [y_L][y_R] = 2^s y_L + y_R, \text{ where } s = \left\lceil \frac{n}{2} \right\rceil$$

$$xy = (2^s x_L + x_R)(2^s y_L + y_R)$$

$$= 2^n x_L y_L + 2^s (x_L y_R + x_R y_L) + x_R y_R$$

– T(n) ➔ 4 T(n/2) + O(n)

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

– T(n) ➔ 3 T(n/2) + O(n)

1.1 Multiplication

- Divide u & v such that

$$u = w \times 10^s + x$$

$$v = y \times 10^s + z, \text{ where } s = \left\lceil n/2 \right\rceil$$

$$uv = (w \times 10^s + x)(y \times 10^s + z)$$

$$= wy \times 10^{2s} + (wz + xy) \times 10^s + xz$$

  - T(n) ➔ 4 T(n/2) + n

$$(wz + xy) = (w + x)(y + z) - wy - xz$$

  - T(n) ➔ 3 T(n/2) + n

1.1 Multiplication

- Algorithm

```
void multiply ( int u, int v )
{
    n ← min ( digit of u, digit of v );
    if ( n is small enough )
        return u * v;
    s ← n div 2;
    w ← u div 10^s;
    x ← u mod 10^s;
    y ← v div 10^s;
    z ← v mod 10^s;

    r ← multiply ( w + x, y + z );
    p ← multiply ( w, y );
    q ← multiply ( x, z );

    return p * 10^2s + (r – p – q) * 10^s + q;
}
```

1.1 Multiplication

- Algorithm → Check three points

```
void multiply ( int u, int v )
{
    n ← min ( digit of u, digit of v );
    if ( n is small enough )
        return u * v;
    s ← n div 2;
    w ← u div 10^s;
    x ← u mod 10^s;
    y ← v div 10^s;
    z ← v mod 10^s;

    r ← multiply ( w + x, y + z );
    p ← multiply ( w, y );
    q ← multiply ( x, z );

    return p * 10^2s + (r – p – q) * 10^s + q;
}
```

1.1 Multiplication

- Performance analysis
  - Recurrence relation

$$T(n) = 3T(n/2) + n$$

- a = 3, b = 2, k = 1.
- a = 3 > b^k = 2^1,

$$T(n) = \theta(n^{\log_2 3})$$

1.2 Recurrence relation

- Recurrence relation
  - An equation in which each term of the sequence is defined as a function of the preceding terms
  - Examples
    - $a_n = a_{n-1} + 2$
    - $f(n) = n\, f(n-1)$
    - $f(n) = f(n-1) + f(n-2)$
    - $f(n) = f(n/2) + n$
  - Solutions
    - Repeated substitution or telescoping
    - Guess & verification
    - Master theorem

1.2 Recurrence relation

Solution 1: Repeated substitution

- Continually substitute the recurrence relation on the right hand side
- Substitute a value into the original equation and then derive a previous version of the equation

- Examples

  (1) $T(n) = T(n-1) + n$

  $T(1) = 1$

  (2) $T(n) = 2T(n/2) + n$

  $T(1) = 1$

1.2 Recurrence relation

Solution 1: Repeated substitution

(1) $T(n) = T(n-1) + n$

$T(1) = 1$

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= (T(n-2) + (n-1)) + n \\
&= (T(n-3) + (n-2)) + (n-1) + n \\
&\ldots \\
&= T(1) + 2 + 3 + \ldots + n \\
&= 1 + 2 + \ldots + n \\
&= n(n+1)/2 \\
&= O(n^2)
\end{aligned}
$$

1.2 Recurrence relation

Solution 1: telescoping

(2) $T(n) = 2T(n/2) + n$

$T(1) = 1$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ T(n/2) &= 2T(n/4) + n/2 \\ T(n/4) &= 2T(n/8) + n/4 \\ &\ldots \\ T(2) &= 2T(1) + 2 \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ 2T(n/2) &= 4T(n/4) + n \\ 4T(n/4) &= 8T(n/8) + n \\ &\ldots \\ 2^{k-1} T(2) &= 2^k T(1) + n \end{aligned}$$

$(n = 2^k)$

$$\begin{aligned} T(n) &= n + n + \ldots + n \\ &= (k+1)\, n = n \log n + n \\ &= O(n \log n) \end{aligned}$$

1.2 Recurrence relation

Solution 2: Guess & verification

- Guess the solution of the recurrence relation
- Verify that the solution is correct
- Examples

$$T(n) = 2T(n/2) + n$$

1.2 Recurrence relation

Solution 2: Guess & verification

– Examples

$$T(n) = 2T(n/2) + n$$

<Guess>

$$T(n) = O(n \log n), \text{ i.e. } T(n) \le c\, n \log n$$

<Verification>

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\le 2c(n/2)\log(n/2) + n \\ &= cn\log n - cn\log 2 + n \\ &= cn\log n + (-c\log 2 + 1)n \\ &\le cn\log n \end{aligned}$$

1.2 Recurrence relation

Solution 3: Master theorem

– if T(n) is as follows,

$$T(n) = aT(n/b) + O(n^d)$$

then T(n) is

$$T(n) = \begin{cases} O(n^d), \text{ if } d > \log_b a \\ O(n^d \log n), \text{ if } d = \log_b a \\ O(n^{\log_b a}), \text{ if } d < \log_b a. \end{cases}$$

1.2 Recurrence relation

- Performance comparison

Function	n: 10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

1.3 Sorting

- Problem:
  - Arrange the elements of a set in an ascending (or descending) order
  - [2, 1, 4, 3, 7, 5, 6] ➔ [1, 2, 3, 4, 5, 6, 7]
- Sorting algorithms
  - O (n^2) algorithms for pairwise comparison
    - Bubble sort
    - Insertion sort
    - Selection sort
    - ……
  - Can it be faster? → Use divide & conquer
    - Merge sort
    - Quick sort

1.3 Sorting

- Merge sort
  - Given a sequent of n elements { a_1, a_2, …, a_n }, split them into two set.
  - Sort each set individually and merge them to produce a single sorted sequence of n elements.

  - Three steps
    - Divide
      - Split the list into two halves
    - Conquer
      - Recursively sort each half
    - Combine
      - Merge the two sorted sublists

1.3 Sorting

- Merge sort (Divide & Conquer)

```
void msort( int s, int e, int A[] )
{
    if ( s == e )
        return;
    m ← (s+e)/2;
    msort ( s, m, A );
    msort ( m+1, e, A );
    merge ( s, m, e, A );
}
```

1.3 Sorting

- Merge sort (Check three points)

```
void msort( int s, int e, int A[] )
{
    if ( s == e )
        return;
    m ← (s+e)/2;
    msort ( s, m, A );
    msort ( m+1, e, A );
    merge ( s, m, e, A );
}
```

1.3 Sorting

- Merge sort (Example)

16	12	5	38	19	4	20	27

1.3 Sorting

- Merge sort (Performance analysis)
  - Recurrence relation

$$T(n) = \begin{cases} 2T(n/2) + cn, & n > 1 \\ a, & n = 1 \end{cases}$$

$$T(n) = an + cn \log n$$

1.3 Sorting

- Iterative merge sort
  - A merge sort without recursive call

```
function iterative-mergesort(a[1...n])
Input: elements a_1, a_2, ..., a_n to be sorted

Q = [ ] (empty queue)
for i = 1 to n:
   inject(Q, [a_i])
while |Q| > 1:
   inject(Q, merge(eject(Q), eject(Q)))
return eject(Q)
```

1.3 Sorting

- Quick sort
  - Given a sequence of n elements { a_1, a_2, …, a_n }, split them into two set.
  - Rearrange the elements so that merging later is not necessary.
  - Rearrange such that $a_i <= a_j$ for all i between 1 and m and j between m+1 and n ➔ partitioning
  - Two steps
    - Divide
      - Split the list into two halves (with partitioning)
    - Conquer
      - Recursively sort each half

1.3 Sorting

- partition

```
int partition ( int s, int e, int numbers[] )
{   int pivot, left, right;
    left = s;       right = e;          pivot = A[s];
    while (left < right) {
        while ((numbers[right] >= pivot) && (left < right))
            right--;
        if (left != right)
             numbers[left++] = numbers[right];

        while ((numbers[left] <= pivot) && (left < right))
            left++;
        if (left != right)
            numbers[right--] = numbers[left];
    }
    numbers[left] = pivot;
    return left;
}
```

1.3 Sorting

- Quick sort (Divide & Conquer)

```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

1.3 Sorting

- Quick sort (Example)

16	12	5	38	19	4	20	27

1.3 Sorting

- Quick sort (Performance analysis)
  - Recurrence relation

$$T(n) = (n+1) + \frac{1}{n} \sum_{1 \le k \le n} (T(k-1) + T(n-k))$$

1.4 Median (Select k-th)

- Median
  - The 50^{th} percentile element of a list
  - Ex: Median of [45, 1, 10, 30, 25] → 25
- Generalized problem:
  - Select the k-th smallest element among a set of n unsorted elements $\{a_1, a_2, \ldots, a_n\}$.
- Key idea:
  - Use partition ( ) in quick sort algorithm
    - The partition ( ) returns the position of the pivot → m
    - If k < m, then select k-th in the left subset
    - Else, find select (k-m)-th in the right subset

1.4 Median (Select k-th)

- Select (Divide & Conquer)

```
void select_kth ( int k, int s, int e, int A[ ] )
{
    if ( s == e )
        return A[s];

    int m = partition ( s, e, A );

    if ( k < m )
      select_kth ( k, s, m – 1, A );
    else if ( m > k )
      select_kth ( k-m, m + 1, e, A );
    else
      return A[k];
}
```

1.4 Median (Select k-th)

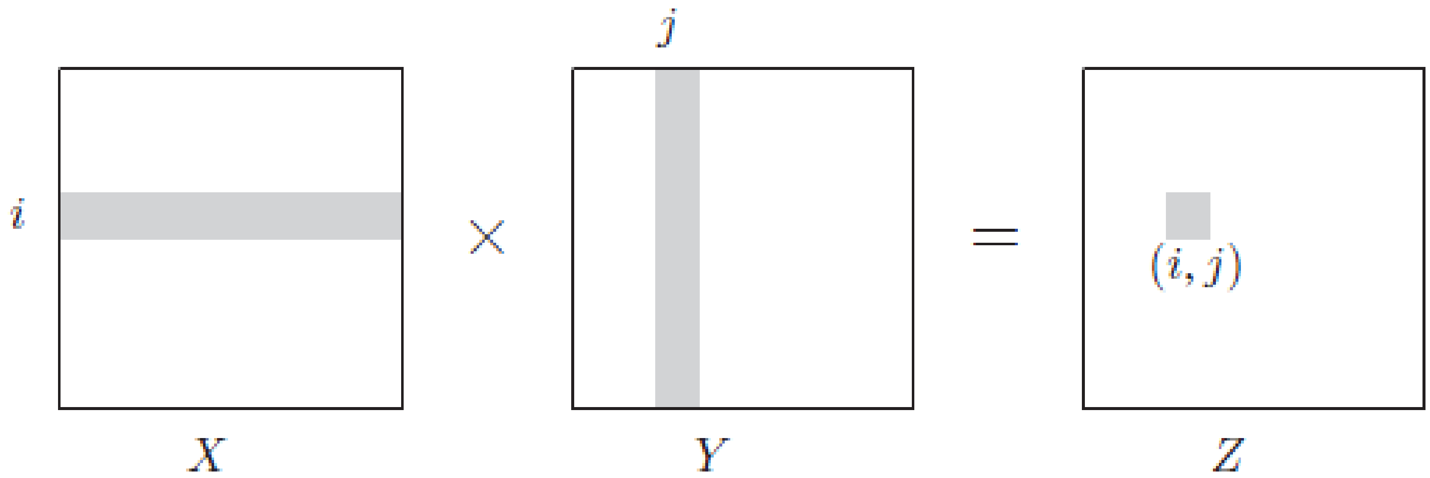
- Selection (Performance analysis)
  - Recurrence relation

$$T(n) = n + T(n/2)$$

1.5 Matrix Multiplication

- Multiplying two matrices: Z = XY
  - X & Y: n × n matrices
  - Time complexity: $O(n^3)$

$$Z_{i,j} = \sum_{k=1}^{n} X_{ik} Y_{kj}$$



1.5 Matrix Multiplication

- Multiplying two matrices: Z = XY
  - X & Y: n × n matrices

```
void mat_mult( int **Z, int **X, int **Y, int n )
{
    int i, j, k;

    for ( i = 0; i < n; i++ )
        for ( j = 0; j < n; j++ )
            for ( k = 0, Z[i][j] = 0; k < n; k++ )
                Z[i][j] += X[i][k]*Y[k][j];
}
```

1.5 Matrix Multiplication

- Improve the performance using DnC
  - Divide X & Y into 2 × 2 groups

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \ Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$Z = XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

  - 8 (n/2) × (n/2) multiplications
  - Matrix addition → $O(n^2)$

1.5 Matrix Multiplication

- Performance analysis
  - 8 (n/2) × (n/2) multiplications → 8 T(n/2)
  - Matrix addition → O(n^2)

$$T(n) = 8T(n/2) + O(n^2)$$

$$= O(n^3)$$

  - No improvement !!

1.5 Matrix Multiplication

- Decomposing and assembling multiplications

$$P_1 = A(F - H)$$
$$P_2 = (A + B)H$$
$$P_3 = (C + D)E$$
$$P_4 = D(G - E)$$
$$P_5 = (A + D)(E + H)$$
$$P_6 = (B - D)(G + H)$$
$$P_7 = (A - C)(E + F)$$

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$
$$= \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

1.5 Matrix Multiplication

- Improvement of performance
  - 7 (n/2) × (n/2) multiplications → 7 T(n/2)
  - Matrix addition → O(n^2)

$$T(n) = 7T(n/2) + O(n^2)$$

$$= O(n^{\log_2 7})$$

$$\approx O(n^{2.81})$$

1.6 Finding the closest pair of points

- Problem (2.32 at P78)

CLOSEST PAIR
Input: A set of points in the plane, $\{p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)\}$
Output: The closest pair of points: that is, the pair $p_i \neq p_j$ for which the distance between p_i and p_j, that is,

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

is minimized.

For simplicity, assume that n is a power of two, and that all the x-coordinates x_i are distinct, as are the y-coordinates.

1.6 Finding the closest pair of points

Here's a high-level overview of the algorithm:

- Find a value x for which exactly half the points have $x_i < x$, and half have $x_i > x$. On this basis, split the points into two groups, L and R.
- Recursively find the closest pair in L and in R. Say these pairs are $p_L, q_L \in L$ and $p_R, q_R \in R$, with distances d_L and d_R respectively. Let d be the smaller of these two distances.
- It remains to be seen whether there is a point in L and a point in R that are less than distance d apart from each other. To this end, discard all points with $x_i < x - d$ or $x_i > x + d$ and sort the remaining points by y-coordinate.
- Now, go through this sorted list, and for each point, compute its distance to the *seven* subsequent points in the list. Let p_M, q_M be the closest pair found in this way.
- The answer is one of the three pairs $\{p_L, q_L\}$, $\{p_R, q_R\}$, $\{p_M, q_M\}$, whichever is closest.

1.6 Finding the closest pair of points

(a) In order to prove the correctness of this algorithm, start by showing the following property: any square of size $d \times d$ in the plane contains at most four points of L.

(b) Now show that the algorithm is correct. The only case which needs careful consideration is when the closest pair is split between L and R.

(c) Write down the pseudocode for the algorithm, and show that its running time is given by the recurrence:

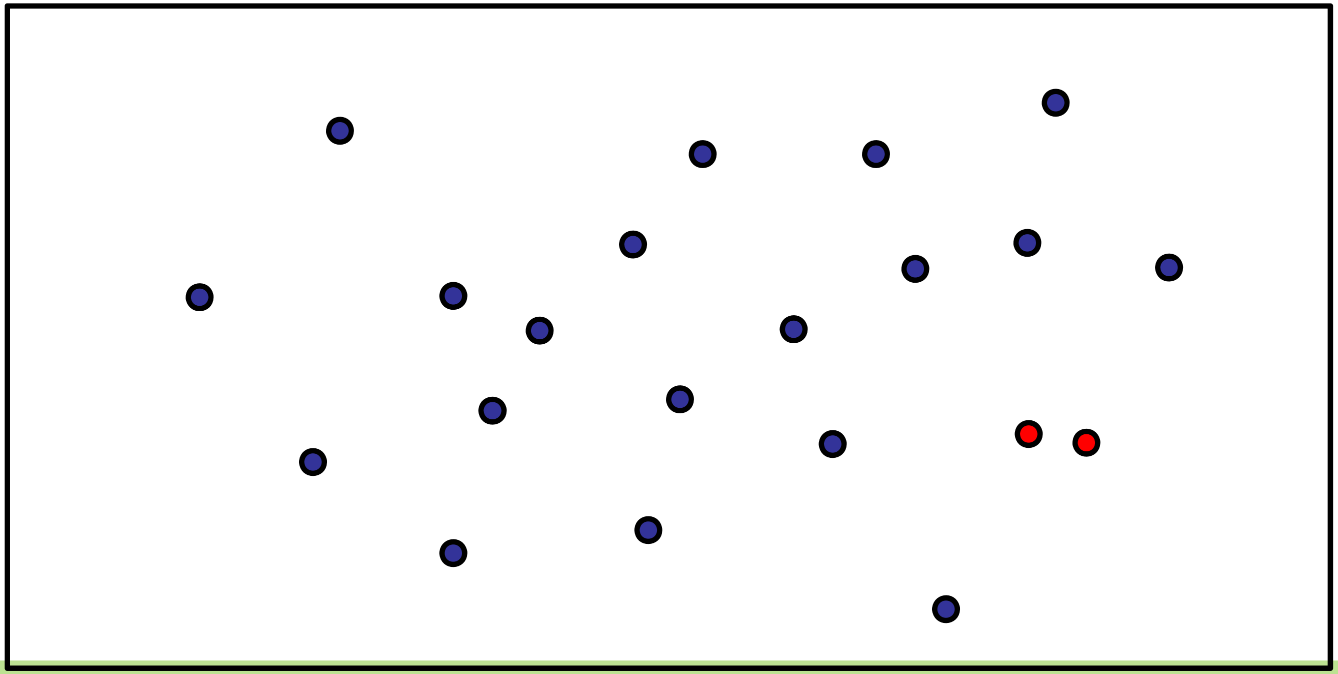
$$T(n) = 2T(n/2) + O(n \log n).$$

Show that the solution to this recurrence is $O(n \log^2 n)$.

(d) Can you bring the running time down to $O(n \log n)$?

1.6 Finding the closest pair of points

- Problem
  - Closest pair
    - Given n points in 2D: $\{p_1, \ldots, p_n\}$, where $p_i = (x_i, y_i)$.
    - Distance between two points: $d(p_i, p_j)$
    - Find i & j such that $d(p_i, p_j)$ is minimum among all i & j



1.6 Finding the closest pair of points

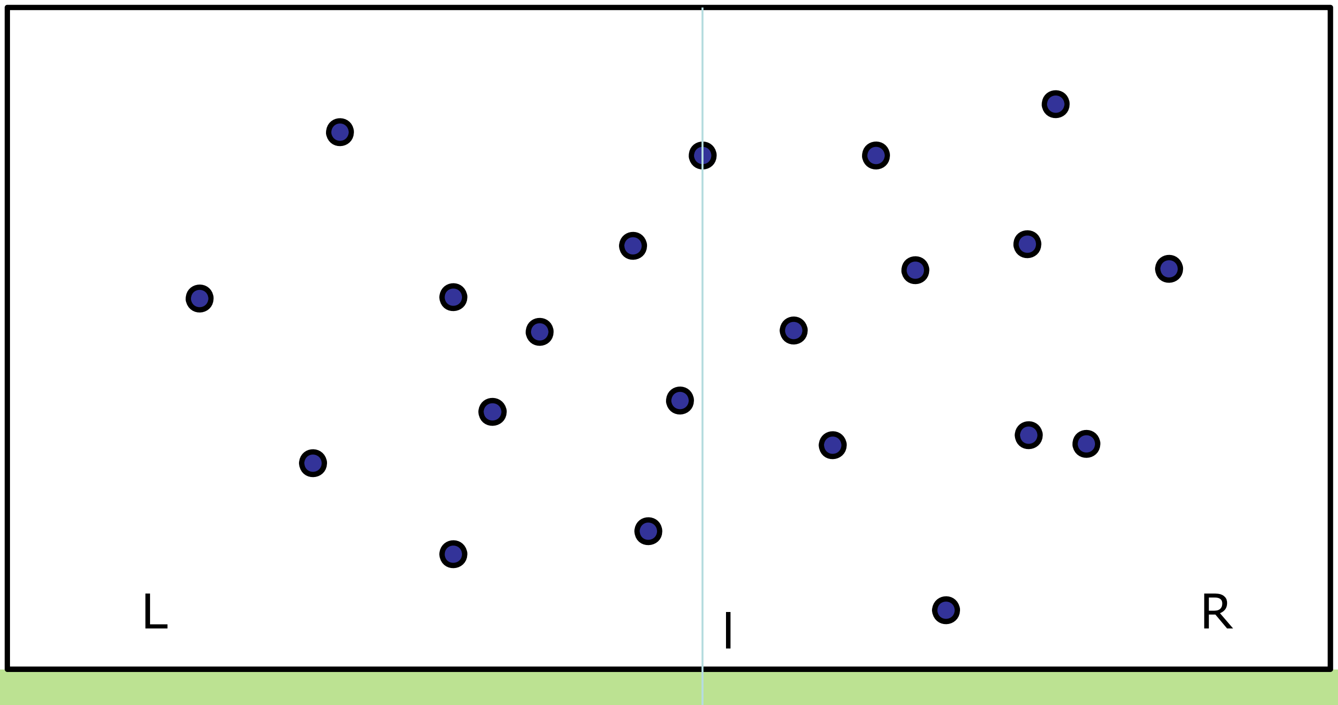
- Solution
  - Bruteforce algorithm
    - Compare pairwisely

```
void closest_pair( int n, Point p )
{
    float min_dist = MAX;
    int min_pair[2];
    for ( i = 0; i < n; i++ ) {
        for ( j = i+1; j < n; j++ ) {
            if ( d (p[i], p[j]) < min_dist )
                min_dist = d (p[i], p[j]);
                min_pair = (i, j);
        }
    }
}
```

    - Time complexity?

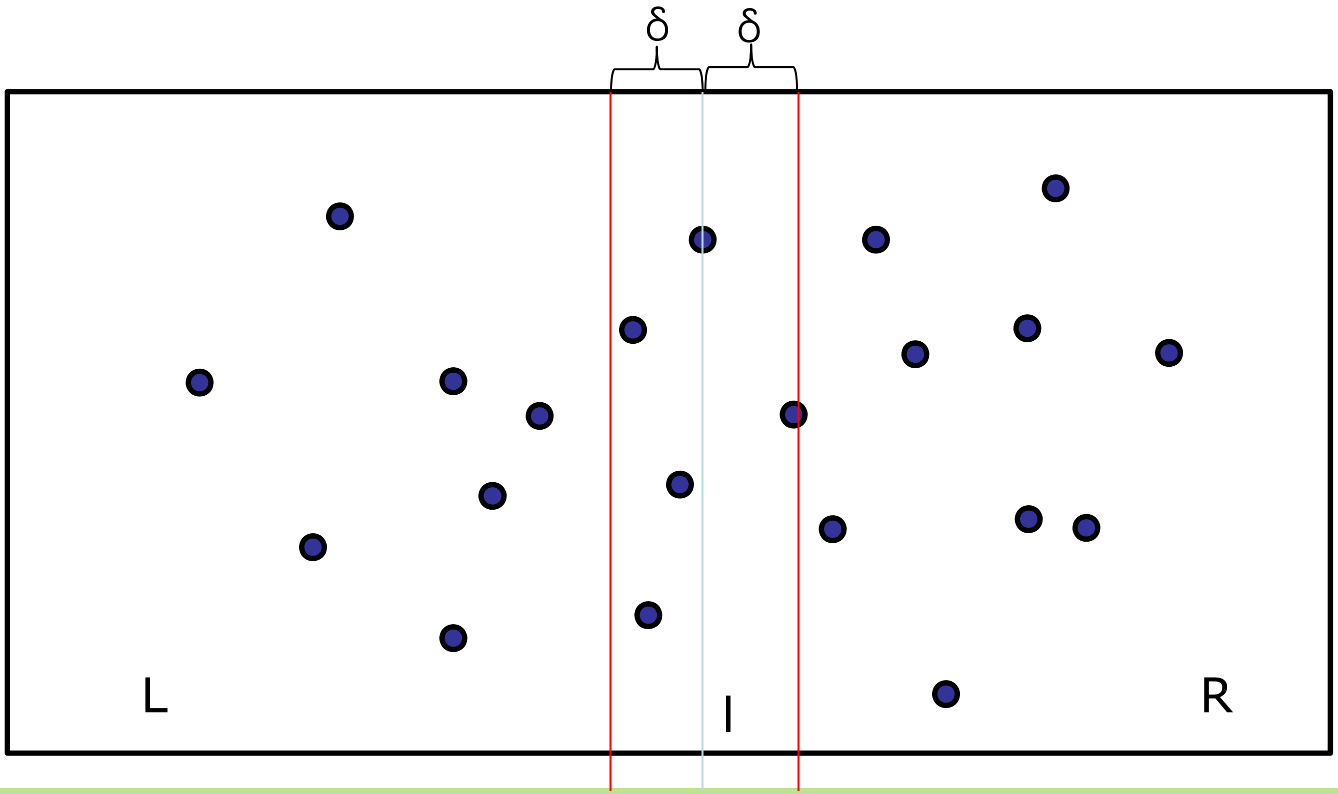
1.6 Finding the closest pair of points

- Solution
  - Divide and conquer
    - Sort the points in x-coordinate
    - Set a line $l = x_{median}$
    - Divide points as $L = \{p_i \mid x_i < l\}$ and $R = \{p_i \mid x_i > l\}$



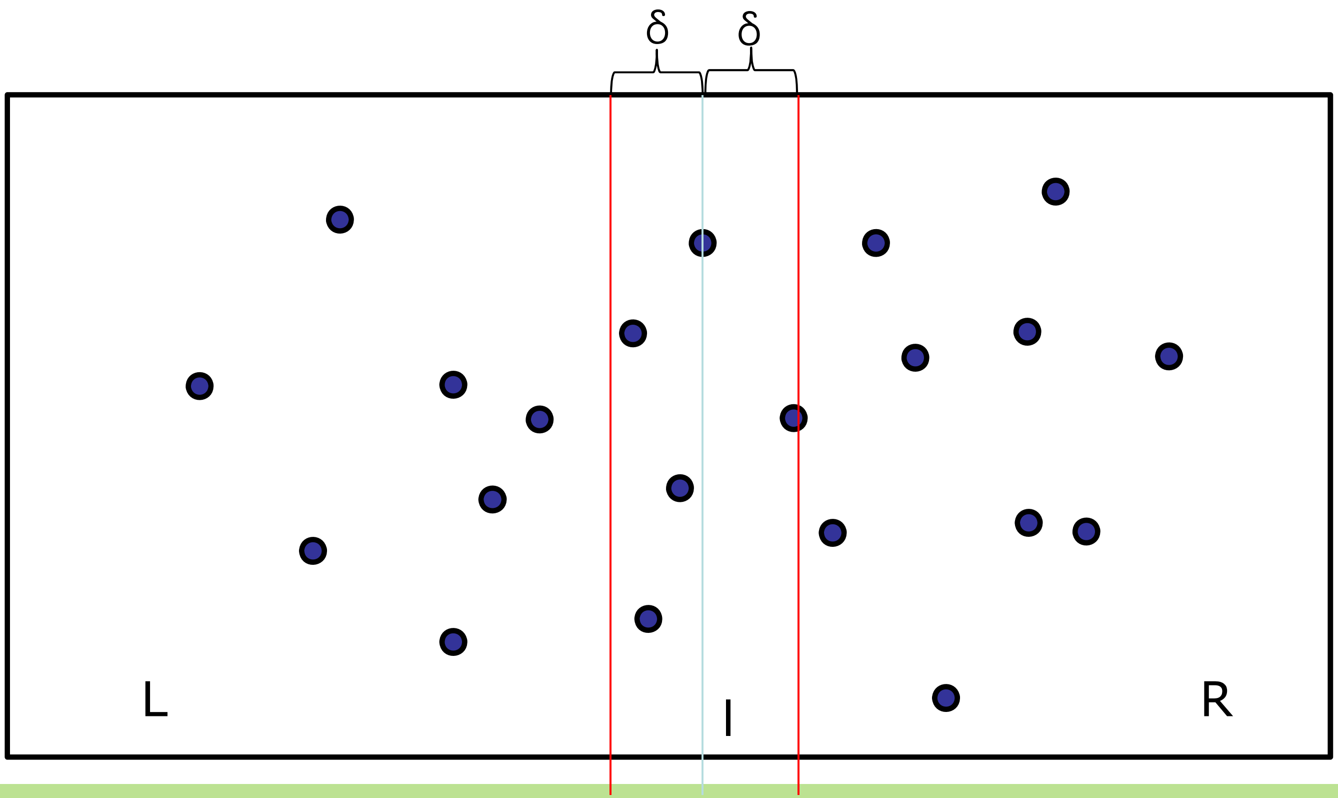
1.6 Finding the closest pair of points

- Solution
  - Divide and conquer
    - Min_pair (P) = Min_pair (L) + Min_Pair (R) + Merge
      - δ = min (Min_pair (L), Min_pair(R))



1.6 Finding the closest pair of points

- Solution
  - Divide and conquer
    - Merge finds closest pair only for points in (l – δ, l + δ)
      ➔ It takes O(n)



1.6 Finding the closest pair of points

- Solution
  - Divide and conquer algorithm

```
float closest_pair( int n, Point p )
{
    if ( n == 2 )
        return d (P[0], P[1]);
    l = median_X ( P );
    float lmin = closest_pair ( L );
    float rmin = closest_pair ( R );
    float delta = min (lmin, rmin);
    float mmin = merge ( delta );
    return min ( mmin, delta );
}
```

1.6 Finding the closest pair of points

- Analysis
  - Time complexity
    - Sorting in x-coordinate: O(n log n)
    - Closest-pair

$$T(n) = 2T(n/2) + O(n)$$
$$= O(n \log n)$$

    - In total, O(n log n)

1. Divide & Conquer

1.0 Introduction

1.1 Multiplication

1.2 Recurrence relation

1.3 Sorting

1.4 Medians

1.5 Matrix multiplication

1.6 Finding closest pair

Contents

0. Prologue

1. Divide & conquer

2. Graph

3. Greedy algorithm

4. Dynamic programming