Divide and Conquer

Recall

- Complexity Analysis
  - Comparison of algorithm
  - Big O
    - Simplification
    - From source code
      - Recursive

Today Topic

- Divide and Conquer

DIVIDE AND CONQUER

Algorithm Design Technique

Divide and Conquer

- Mathematical Induction Analogy

- Solve the problem by
  - Divide it into smaller parts
  - Solve the smaller parts **recursively**
  - Merge the result of the smaller parts

Induction

- Prove the smallest case **(the basis step)**
- Relate how the result from the smaller case constitutes the proof of the larger case **(the induction step)**
  - What is the relation ?
    - (n-1) → n? (basic induction)
    - (n-m)→n? (complete induction)

Induction Example

- Show that
  - 1+2+3+4+…+n = n(n+1)/2
- The basis step
  - 1 = 1(1+1)/2 **true**

The Inductive Step

- The inductive step
  - Assume that it is true for (n-1)
  - Check the case (n)
    - 1 + 2 + 3 +… + n = (1 + 2 + 3 + … + (n-1)) + n
    - = (n-1)(n) / 2 + n
    - = n((n-1)/2 + 1)
    - = n(n/2-1/2 + 1)
    - = n(n/2 + 1/2)
    - = n(n+1)/2

The Inductive Step

- The inductive step
  - Assume that it is true for (n-1)
  - Check the case (n)
    - 1 + 2 + 3 +… + n = (1 + 2 + 3 + … + (n-1)) + n
    - = (n-1)(n) / 2 + n
    - = n((n-1)/2 + 1)
    - = n(n/2-1/2 + 1)
    - = n(n/2 + 1/2)
    - = n(n+1)/2

By using (n-1)

The Induction

- By using the result of the smaller case
- We can proof the larger case

Key of Divide and Conquer

- Divide into smaller parts (subproblems)
- Solve the smaller parts **recursively**
- Merge the result of the smaller parts

Steps in D&C

- Questions
  - **What if we has the solution of the subproblem (smaller problem)?**
    - **What is the subproblem? (how to divide?)**
    - **What can we do with the result? (how to conquer?)**
- If we know the answer, the rest comes automatically from the recursion

Code Example

```
ResultType DandC(Problem p) {
    if (p is trivial) {
       solve p directly
       return the result
    } else {
       divide p into p1,p2,...,pn

       for (i = 1 to n)
         ri = DandC(pi)

       combine r1,r2,...,rn into r
       return r
    }
}
```

Code Example

```
ResultType DandC(Problem p) {
    if (p is trivial) {
       solve p directly                          ts
       return the result
    } else {
       divide p into p1,p2,...,pn                td

       for (i = 1 to n)                          tr
         ri = DandC(pi)

       combine r1,r2,...,rn into r               tc
       return r
    }
}
```

Trivial Case · **Divide** · **Recursive** · **Combine**

Examples

- Let's see some examples

MERGE SORT

The Sorting Problem

- Given a sequence of numbers
  - $A = [a_1,a_2,a_3,...,a_n]$
- Output
  - The sequence A that is sorted from min to max

The Question

- What if we know the solution of the smaller problem?
  - What is the smaller problem?
    - Try the same problem → sorting
- What if we know the result of the sorting of some elements?

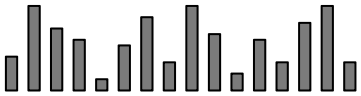
The Question

- How to divide?
  - Let's try sorting of the smaller array
    - Divide exactly at the half of the array
- How to conquer?
  - Merge the result directly

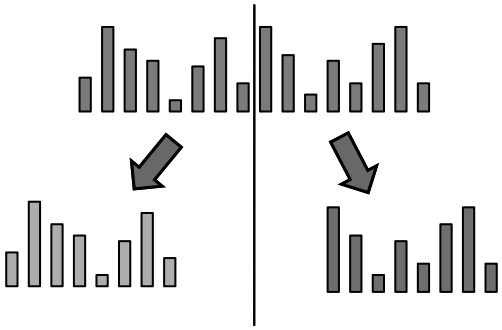
Idea

- Simplest dividing
  - Divide array into two array of half size
- Laborious conquer
  - Merge the result

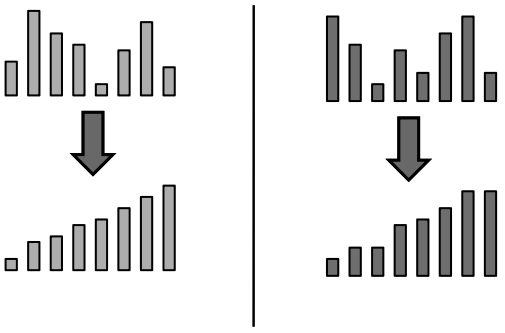
Divide



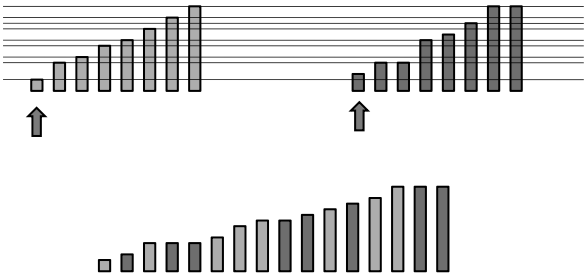
Divide **Θ(1)**



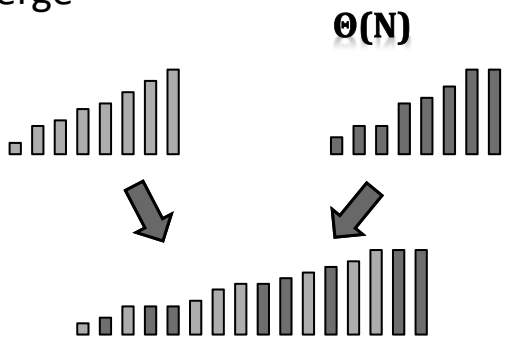
Solve by Recursion **T(N/2)**



Merge



Merge



Analysis

- $T(n) = 2T(n/2) + O(n)$
- Master method
  - $T(n) = O(n \lg n)$

QUICK SORT

The Sorting Problem (again)

- Given a sequence of numbers
  - $A = [a_1, a_2, a_3, \ldots, a_n]$
- Output
  - The sequence A that is sorted from min to max

Problem of Merge Sort

- Need the use of external memory for merging
- Are there any other way such that conquering is not that complex?

The Question

- How to divide?
  - Try doing the same thing as the merge sort
  - Add that every element in the first half is less than the second half
    - Can we do that?
- How to conquer?
  - The sorted result should be easier to merge?

Idea

- Laborious dividing
  - Divide array into two arrays
    - Add the requirement of value of the array
- Simplest conquer
  - Simply connect the result

Is dividing scheme possible?

- Can we manage to have two subproblems of equal size
  - That satisfy our need?
- Any trade off?

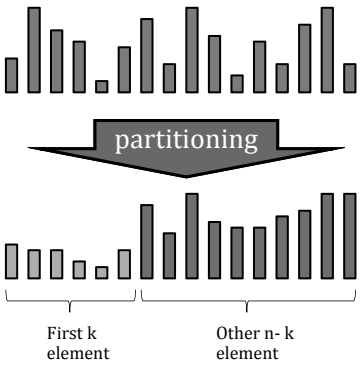
The Median

- We need to know the median
  - There are **n/2 elements** which are not more than the median
  - There are another **n/2 elements** which are not less than the median
- Can we have the median?
  - Hardly possible at this step

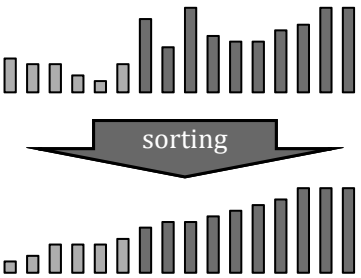
Simplified Division

- Can we simplify?
  - Not using the median?
- Using k^{th} member
  - There are **k elements** which are not more than the median
  - There are another **(n-k) elements** which are not less than the median
- Simply pick any member and use it as a "pivot"

Divide **Θ(N)**



Solve by Recursion



T(K) + T(N – K)

Conquer **O(1)**

Do nothing!

Analysis

- $T(n) = T(k) + T(n - k) + \Theta(n)$
- There can be several cases
  - Up to which K that we chosen

Analysis : Worst Case

- K always is 1st

What should be our T(N) ?

What is the time complexity

Analysis : Worst Case

- K always is 1st

$$T(n) = T(1) + T(n - 1) + \Theta(n)$$
$$= T(n - 1) + \Theta(n)$$
$$= \Sigma\, \Theta(i)$$
$$= \Theta(n^2)$$

Not good

Analysis : Best Case

- K always is the median

What should be our T(N) ?

What is the time complexity

Analysis : Best Case

- K always is the median

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

The same as the merge sort (without the need of external memory)

Fixing the worst case

- When will the worst case happen?
  - When we always selects the smallest element as a pivot
  - Depends on pivot selection strategy
- If we select the first element as a pivot
  - What if the data is sorted?

Fixing the worst case

- Select wrong pivot leads to worst case.
- There will exist some input such that for any strategy of "deterministic" pivot selection leads to worst case.

Fixing the worst case

- Use "non-deterministic" pivot selection
  - i.e., randomized selection
- Pick a random element as a pivot
- It is unlikely that every selection leads to worst case
- We can hope that, on average,
  - it is O(n log n)

MODULO EXPONENTIAL

The Problem

- Given x, n and k
- Output: the value of x^n mod k

Naïve method

```
res = x mod k;
for (i = 2 to n) do {
    res = res * x;
    res = res mod k;
}
```

$\Theta(n)$

Using basic facts:
(a * b) mod k = ((a mod k) * (b mod k)) mod k

The Question

- What if we knows the solution to the smaller problem?
  - Smaller problem → smaller N
  - x^n ?? from $x^{(n-e)}$?
- Let's try $x^{(n/2)}$

The Question

- How to divide?
  - $x^n = x^{n/2} * x^{n/2}$ (n is even)
  - $x^n = x^{n/2} * x^{n/2} * x$ (n is odd)
- How to conquer?
  - Compute $x^{n/2}$ mod k
    - Square and times with x, if needed,
    - mod k afterward

Analysis

- $T(n) = T(n/2) + \Theta(1)$
- $= O(\log n)$

Example 2^{92} mod 10

- $2^{92} = 2^{46} \times 2^{46}$ $= 4 \times 4 \bmod 10 = 6$
- $2^{46} = 2^{23} \times 2^{23}$ $= 8 \times 8 \bmod 10 = 4$
- $2^{23} = 2^{11} \times 2^{11} \times 2$ $= 8 \times 8 \times 2 \bmod 10 = 8$
- $2^{11} = 2^5 \times 2^5 \times 2$ $= 2 \times 2 \times 2 \bmod 10 = 8$
- $2^5 = 2^2 \times 2^2 \times 2$ $= 4 \times 4 \times 2 \bmod 10 = 2$
- $2^2 = 2^1 \times 2^1$ $= 2 \times 2 \bmod 10 = 4$

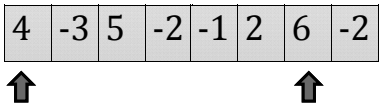
MAXIMUM CONTIGUOUS SUM OF SUBSEQUENCE

(MCS)

The MCS Problem

- Given a sequence of numbers
  - $A = [a_1, a_2, a_3, \ldots, a_n]$
    - There are n members
- Find a subsequence $s = [a_i, a_{i+1}, a_{i+2}, \ldots, a_k]$
  - Such that the sum of the element in s is maximum

Example



Sum = 11

Naïve approach

- Try all possible sequences
  - How many sequence?
  - How much time does it use to compute the sum?

Naïve approach

- Try all possible sequences
  - There are **Θ(n2)** sequence
  - Each sequence takes **O(n)** to compute the sum
- Hence, it's **O(n3)**

- Can be improved by
  - Remembering the summation
  - Using DP (will be discussed in their respective lecture)

The DC Approach

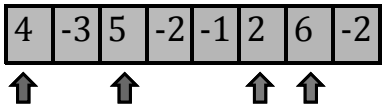
- What if we know the solution of the smaller problem
  - What if we know MSS of the first half and the second half?

The Question

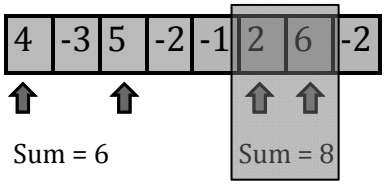
- How to divide?
  - By half of the member
- How to conquer?
  - Does the result of the subproblems can be used to compute the solution?
  - Let's see

Combining the result from sub MSS

Combining the result from sub MSS

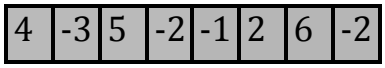


Combining the result from sub MSS



Shall this be our answer?

Combining the result from sub MSS

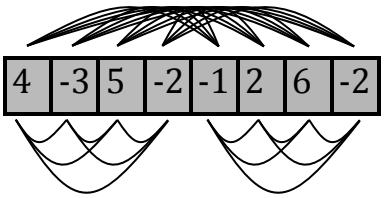


Do we consider all possibilities?

Combining the result from sub MSS

But not the pair including member from both half

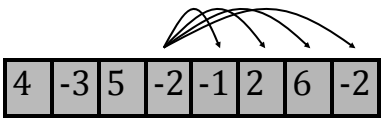
There are $(n/2)^2$ additional pairs



Considered all pairs in each half

Compute max of cross over

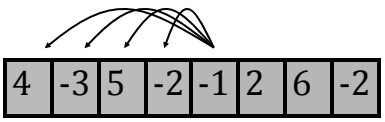
- Consider this



- The max sequence from the green part is always the same, regardless of the pink part
- The sequence always start at the left border

Compute max of cross over

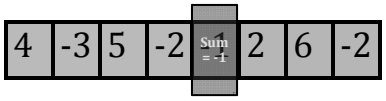
- Similarly



- The max sequence from the pink part is always the same, regardless of the green part
- The sequence always start at the right border

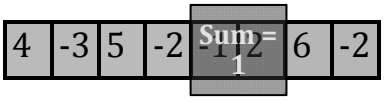
Compute max of cross over

- Just find the marginal max from each part



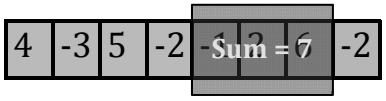
Compute max of cross over

- Just find the marginal max from each part



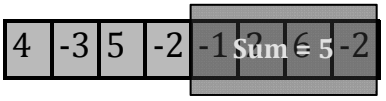
Compute max of cross over

- Just find the marginal max from each part



Compute max of cross over

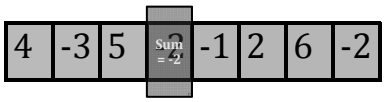
- Just find the marginal max from each part



Less than previous

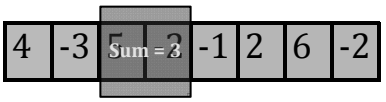
Compute max of cross over

- Just find the marginal max from each part



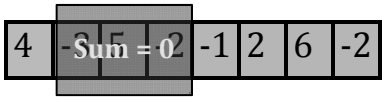
Compute max of cross over

- Just find the marginal max from each part



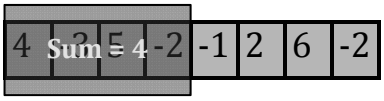
Compute max of cross over

- Just find the marginal max from each part



Compute max of cross over

- Just find the marginal max from each part

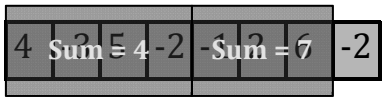


This is max

Compute max of cross over

- Just find the marginal max from each part

takes Θ(n/2) takes Θ(n/2)



Total = 11

Combine

- Max from the three parts
  - The left half
  - The right half
  - The cross over part
- Use the one that is maximum

Analysis

- $T(n) = 2\,T(n/2) + \Theta(n)$ — Left and right parts
- $= \Theta(n \log n)$ — Cross over part

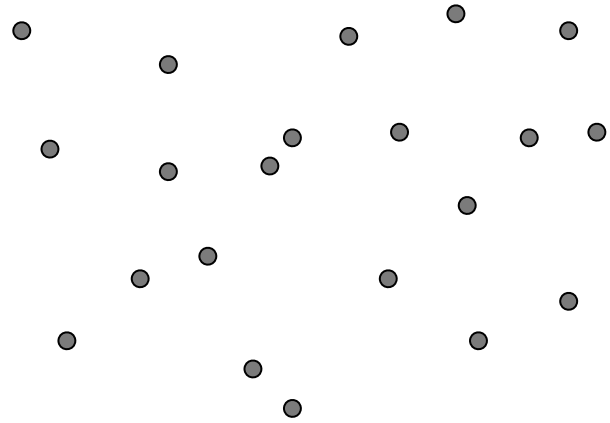
- $a = 2, b = 2, c = \log_2 2 = 1, f(n) = O(n)$
- $n^c = n^2 = \Theta(n)$
  - $f(n) = \Theta(n^c)$ this is case 1 of the master's method
- Hence, $T(n) = \Theta(n \log n)$

CLOSEST PAIR

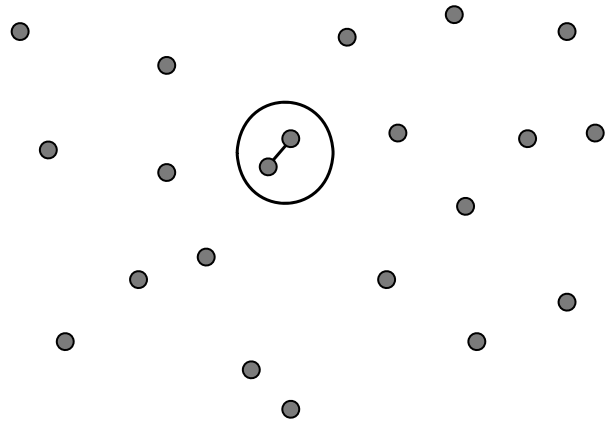
The Problem

- Given
  - N points in 2D
    - $(x_1,y_1), (x_2,y_2), \dots ,(x_n,y_n)$
- Output
  - A pair of points from the given set
    - $(x_a,y_a), (x_b,y_b)$ $1 <= a,b <= n$
    - Such that the distance between the points is minimal

Input Example



Output Example



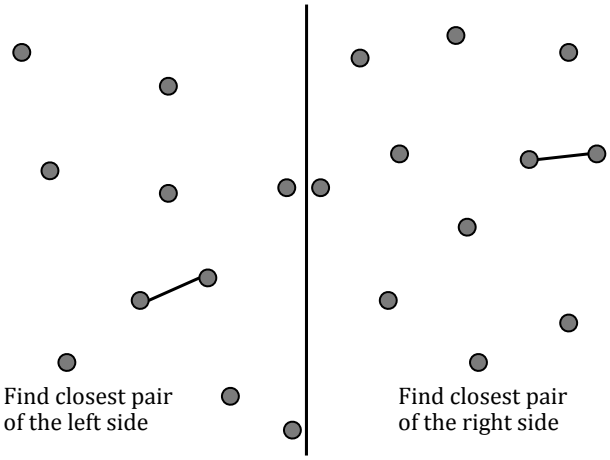
The Naïve Approach

- Try all possible pairs of points
  - There are $\mathbf{n(n+1)/2}$ pairs
  - Compute the distance of each pair
    - Takes $\mathbf{\Theta(1)}$ for each pair
- In total, it is $\mathbf{\Theta(n2)}$

DC approach

- What if we know the solution of the smaller problem
  - What if we know the Closest Pair of half of the points?
    - Which half?

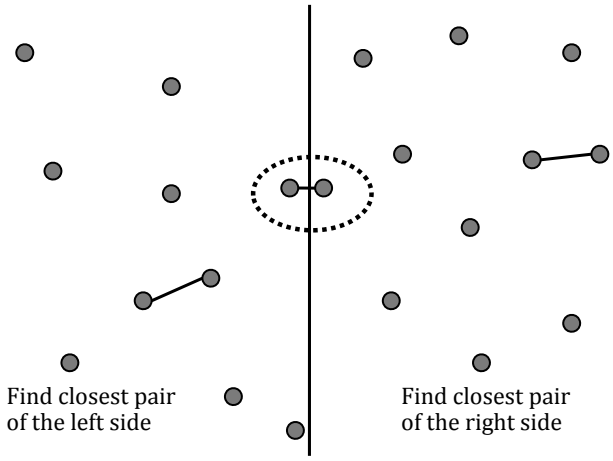
Divide by X axis



Conquer

- Like the MSS problem
  - Solutions of the subproblems do not cover every possible pair of points
  - Missing the pairs that "span" over the boundary

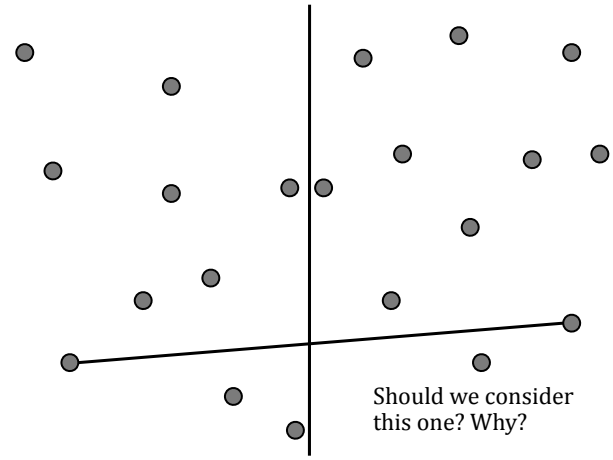
Divide by X axis



Conquer

- Like the MSS problem
  - Solutions of the subproblems do not cover every possible pair of points
  - Missing the pairs that "span" over the boundary
  - There are $\mathbf{(n/2)^2}$ such pairs
  - Again, if we simply consider everything, it would be
    - $\mathbf{O(n^2)}$, still quadratic running time
  - Can we do better?

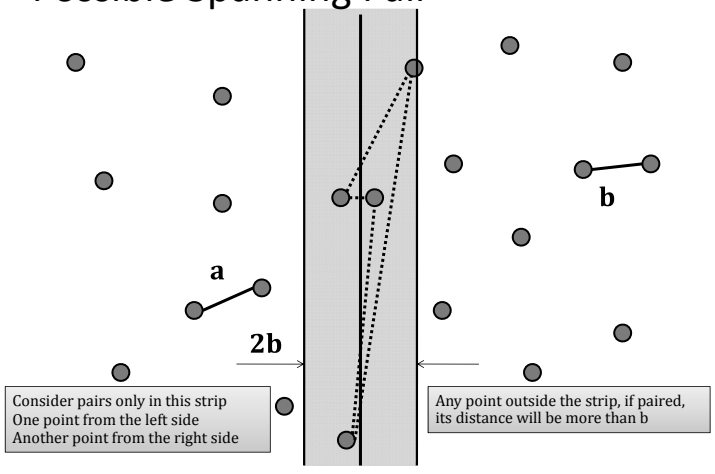
Find Closest Spanning Pair



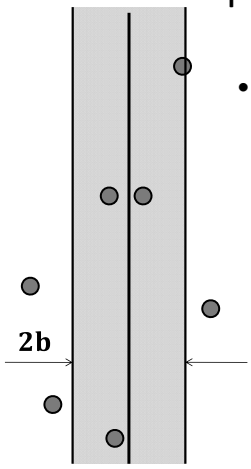
Find Closest Spanning Pair

- Should not consider the pair on the far left with that on the far right
- Should consider only the "nearer" pairs
  - How we know that they are **surely** too far away
  - Anything that we can use to guarantee that some particular pairs should not be considered

Possible Spanning Pair

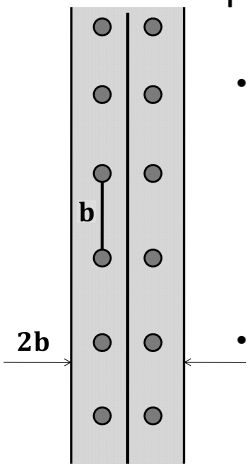


Point in Strips



- How many points in the strip?
  - Should be less than **n**?
  - It is, in the previous example
  - Well? Is it true?

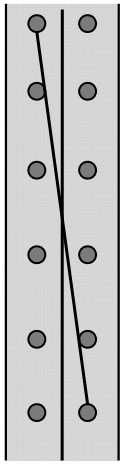
Point in Strips is O(N)



- Bad news
  - N points are possible
  - Consider a set of vertically aligned point
    - Each are **b** unit apart
  - So, every point will be in the strip of **2b** width
- The problem
  - If we check every pair of points, we still stuck with **O(n^2)** time

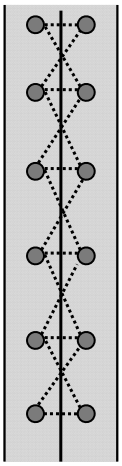
The Solution

- Think Vertically
  - Do we have to check for every pair in the strip?
  - Do we need to consider this pair?
    - No, just like the case of X-axis
    - Don't consider pairs that is surely further than **b**



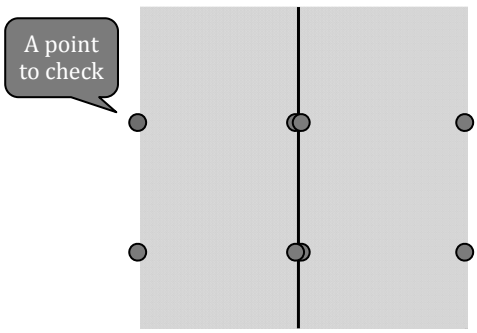
Spanning Pair to be considered

- X-value must be in the strip
  - Check only point in the left side to point in the right side
- Y-value
  - For points in the strip
    - Check only point whose y-value is not more than **b** unit apart



Question is still remains

- How many pair to be checked?



There are, **at worse**, 7 more points for each starting point

3 of them are on the same side, and we need not to check

Implementation Detail

- In practice, to check just only those 4 points lying on the opposite side
- If we loops over every button first to test whether the y-value falls within range
  - That is still **O(n)**!!!!
- The points must be sorted!!!
  - ??? Additional work ????

Naïve approach

- Sort every time we do recursive call
- Each step requires additional **O(n lg n)**
  - That would result in **O(n lg^2 n)**

Better Approach

- Sorting a point
- Point must be sorted in x-value so that dividing can be done in **O(1)**
- Point must also be sorted in y-value
  - When checking point in the strip, when y-value is to far, we can stop immediately
- Both sorting can be done in **O(n lg n)** at the preprocess step
- Data is passed to the function in two separated list, one is x-sorted another one is y-sorted
  - When divide, both list are separated
  - Can be done in **O(n)**

Analysis

- $T(n) = 2\,T(n/2) + 4O(n) + O(n)$
- $= \Theta(n \log n)$

Left and right parts

Point in strip

Divide the list

- $a = 2, b = 2, c = \log_2 2 = 1, f(n) = O(n)$
- $n^c = n^2 = \Theta(n)$
  - $f(n) = \Theta(n^c)$ this is case 1 of the master's method
- Hence, $T(n) = \Theta(n \log n)$

MATRIX MULTIPLICATION

The Problem

- Given two square matrix
  - $A_{n \times n}$ and $B_{n \times n}$
  - $A \in R^{n \times n}$ and $B \in R^{n \times n}$
- Produce
  - C = AB

Multiplying Matrix

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,k} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,1} & b_{k,2} & \cdots & b_{k,n} \end{bmatrix}$$

$$c_{i,j} = \Sigma(a_{i,k} * b_{k,j})$$

Naïve Method

```
for (i = 1; i <= n;i++) {
  for (j = 1; j <= n;j++) {
     sum = 0;
     for (k = 1;k <= n;k++) {
       sum += a[i][k] * b[k][j];
     }
     c[i][j] = sum;
  }
}
```

$O(N^3)$

Simple Divide and Conquer

- Divide Matrix into Block

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

- Multiply the block

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

Simple Divide and Conquer

- What is the complexity?

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

- $T(n) = 8T(n/2) + O(n^2)$
- Master method gives $O(n^3)$
  - Still the same

Strassen's Algorithm

- We define

$$\begin{aligned} \mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\ \mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\ \mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

- Note that each M can be computed by one single multiplication of n/2 * n/2 matrices
- The number of addition is $10(n/2)^2$

Strassen's Algorithm

- Compute the result

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6 \end{aligned}$$

- 8 more additions of n/2 * n/2 matrices
- Hence, total addition is 18 $(n/2)^2$

Analysis

- $T(n) = 7T(n/2) + O(n^2)$
  - Using Master's method
  - a = 7, b = 2, $f(n) = O(n^2)$
    - $c = \log_2 7 \approx 2.807$
  - Hence, $f(n) = O(n^{2.807})$ ← this is case 1 of the master method
- So, $T(n) = O(n^{2.807})$