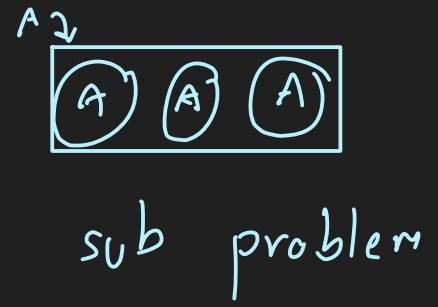
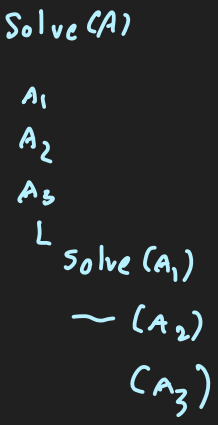
Divide & Conquer

Did you mean recursion?

Nattee Niparnan

What is?

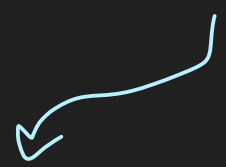




- D&C is an algorithm design framework
  - Some problem can be efficiently solve by this design, some are not
- Key Idea:
  - Solve a problem instance by divide it into smaller instances of the same type
    - A smaller instances is called a subproblem
  - Use recursive to solve the subproblems
    - The subproblem is also solved in the same way: recursive
    - The subproblems are divided repeatedly until the instance can no longer be divide, which should be very small instance that can be solved directly
  - Conquer (Combine) the result of subproblem into a result of the original instance

D&C and Recursive Programming

- Divide and Conquer extensively use recursion
- It is much easier to do recursion according to the definition of D&C
- Analysis of D&C usually be done by Master Method

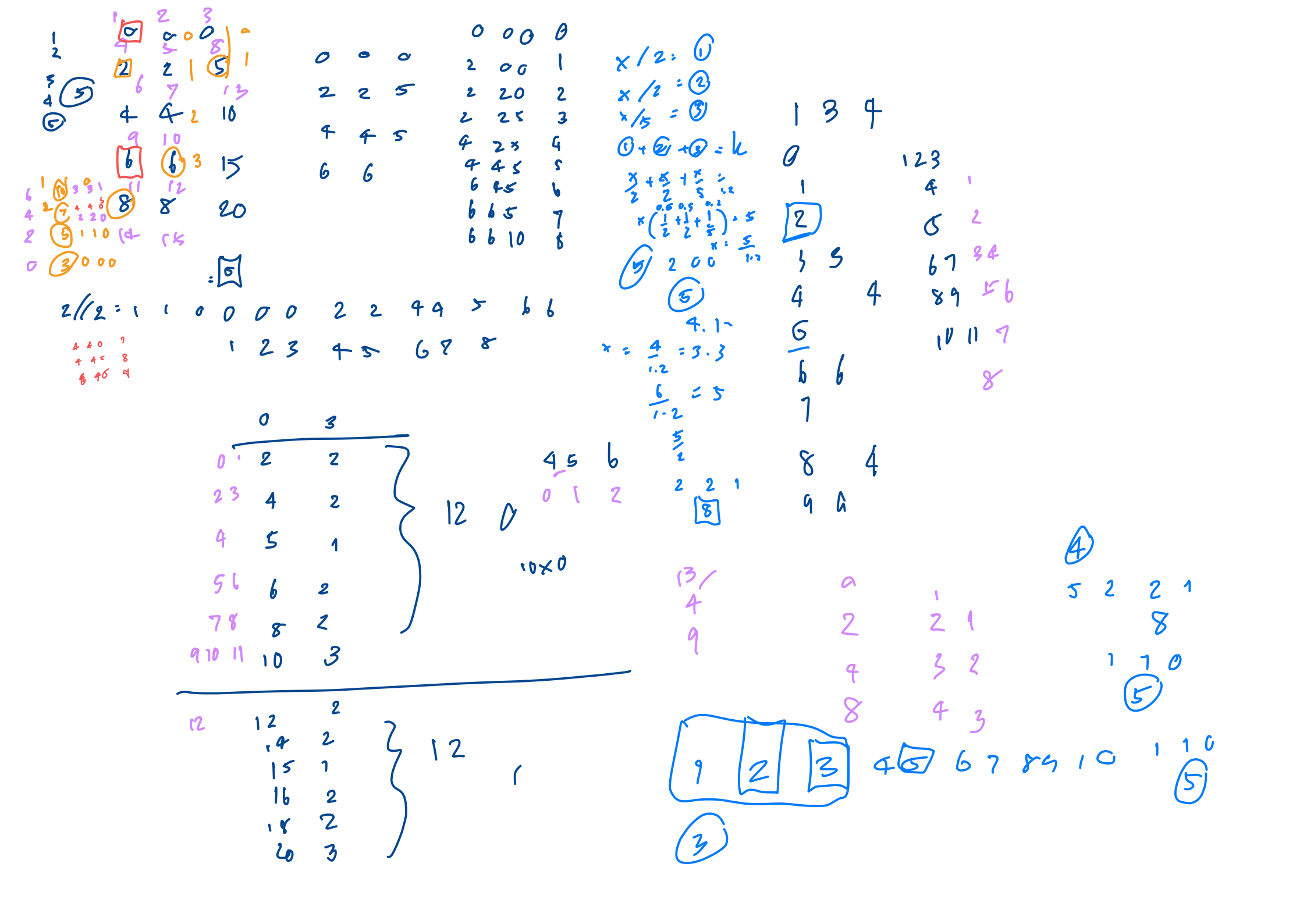


$$T(n) = aT(n/b) + O(n^d)$$

Example

- We will discuss several example
  - Binary Search
  - Merge Sort
  - Quicksort
  - Modulo exponential
  - Maximum Contiguous Sum of Subsequence
  - Strassen’s Matrix Multiplication
  - Closest Pair

Binary Search

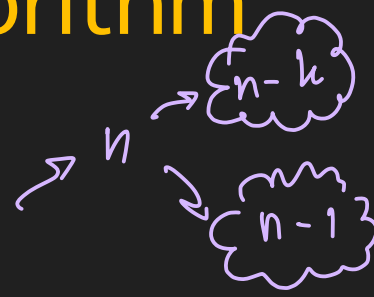


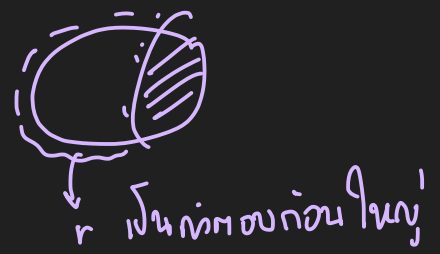
Binary Search Problem

- Input:
  - Array A[1..n], sorted, n >= 1
  - A key k
- Output:
  - If k exists in A, return its first position
  - If k does not, return -1
- Example
  - Input: A = [1, 2, 4, 5, 6, 7, 8, 10], k = 4 output: 3
  - Input: A = [1, 2, 4, 5, 6, 6, 6, 10], k = 6 output: 5 (not 6 nor 7)
  - Input: A = [1, 2, 4, 5, 6, 6, 6, 10], k = 100 output: -1

แนวทาง

A method to design D&C algorithm





- Consider a generic problem instance
- Ask: if we know output of smaller instance, how can we use it to construct the output of the original instance
- Try:
  - Dividing problem into a subproblem with n-1 input, the idea might help solve larger division into n-k inputs
  - Dividing problem into k non-overlapping subproblems, usually 2 problem of the same size

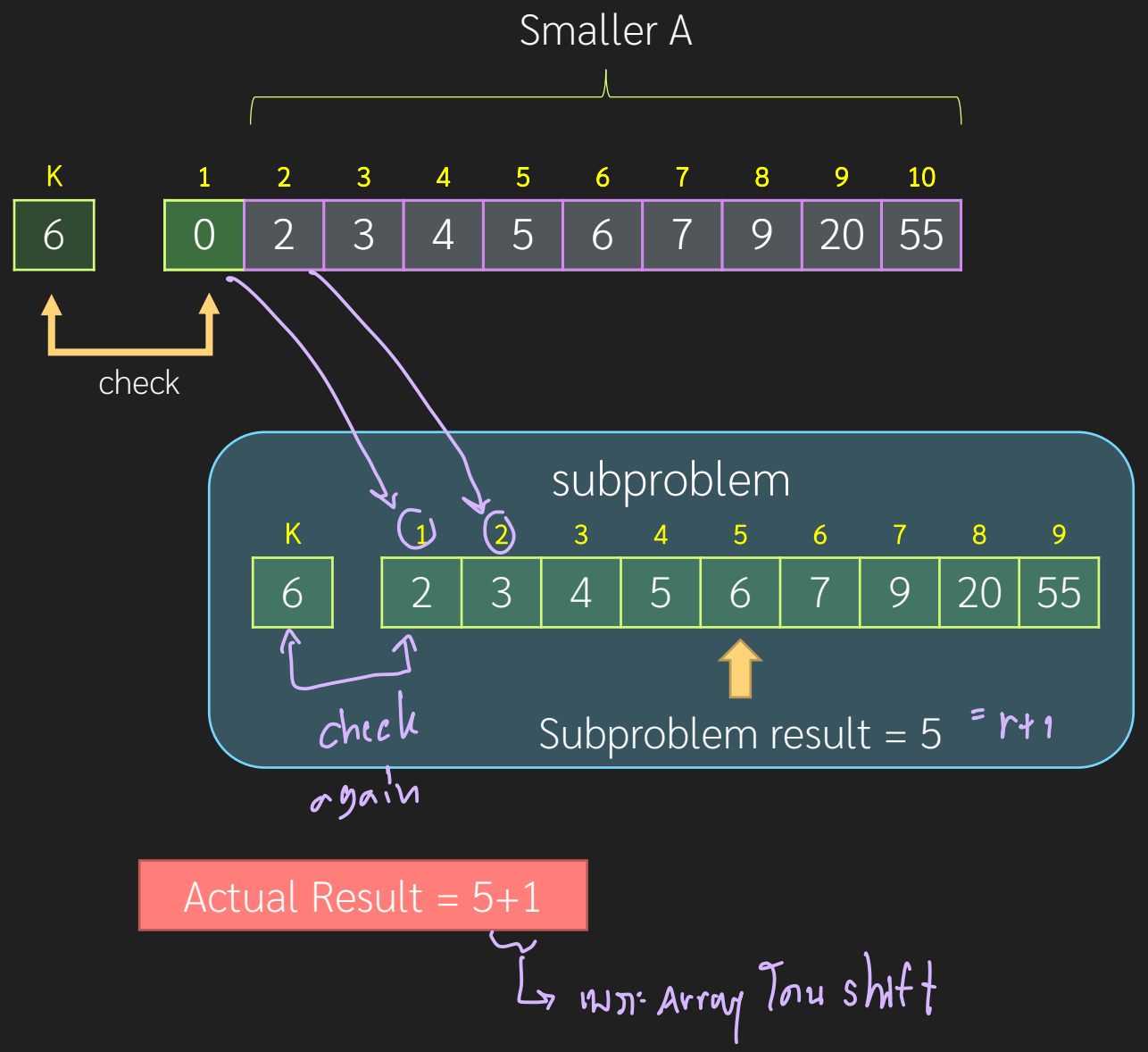
Binary Search, Naïve Approach

```
def bsearch_naive(A[1..n],k)
  for i from 1 to n
    if A[i] == k
      return i
  return -1
end
```

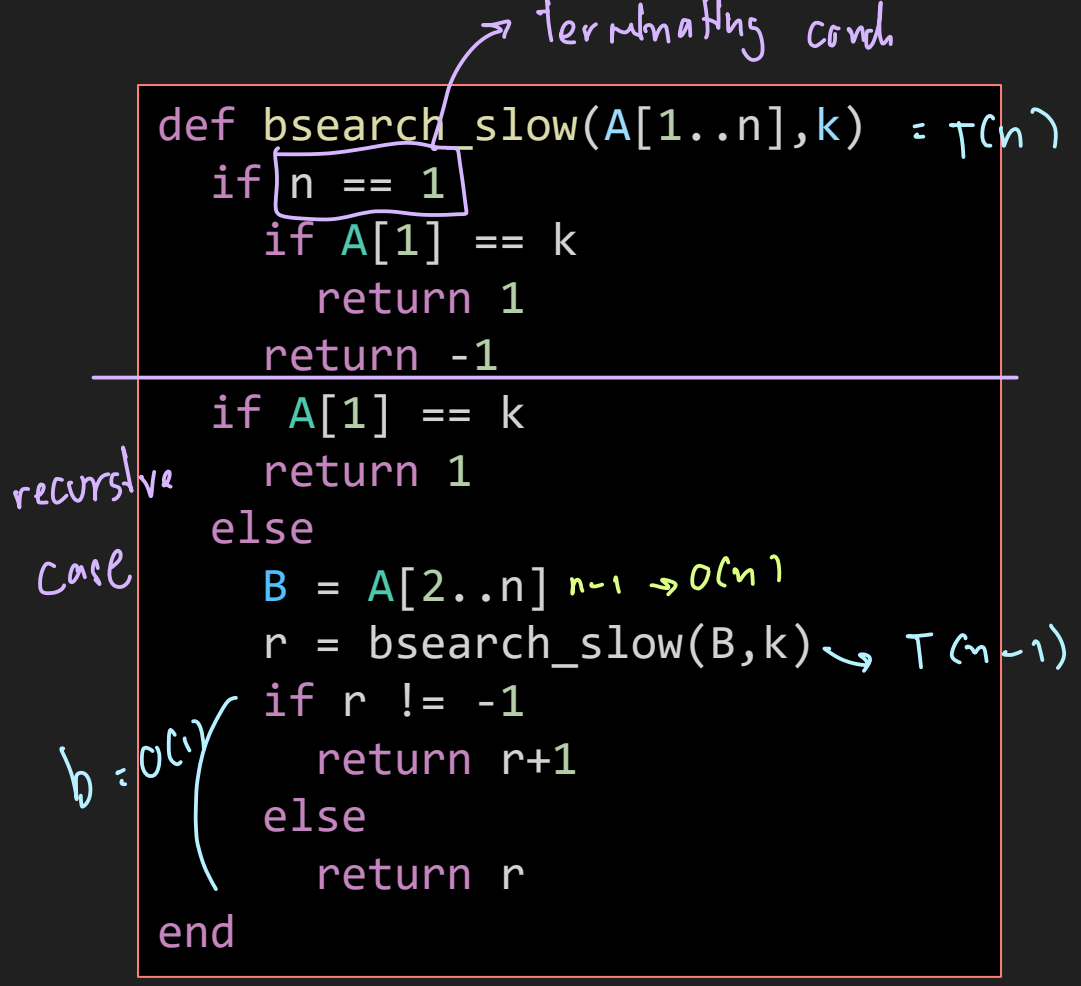
- Normal linear search
- O(N)
- Does not utilize the fact that A[1..n] is sorted

Binary Search, D&C v 0.1

- Key Idea
  - A subproblem is exactly another instance of a Binary Search
  - Divide: A[1..n], k into A[2..n],k
  - Conquer: if A[1] is K, return 1, if not return the result of the sub problem
    - Need to adjust position
  - Trivial case: when n == 1, we check only A[1] without doing any subproblem



Pseudo-code and its analysis



- Trivial case becomes initial condition of the recurrence relation T(1) = O(1)
- T(N) = T(n-1) + a + b
  - a = time to divide the problem
    - O(N) in this case (because we need to create another array)
  - b = time to conquer the problem
    - O(1)
  - T(n) = T(n-1) + O(n)

T(1) = 1

Solving

$$T(n) = \begin{cases} T(n-1) + O(n) & ; n > 1 \\ 1 & ; n = 1 \end{cases}$$

- By substitution

$$T(n) = T(n-1) + n$$
$$T(n-1) = T(n-2) + n - 1$$
$$T(n-2) = T(n-3) + n - 2$$
$$\dots$$
$$T(n-i) = T(n-i-1) + n - i$$
$$\dots$$
$$T(n-(n-2)) = T(n-(n-1)) + 2$$
$$T(1) = 1$$

Sum both sides of the equation,
Recursive terms cancel out

Result is $T(n) = \sum i = \frac{n(n+1)}{2} = O(n^2)$

ของ เท่ $O(n)$

This is slower than naïve!
Because our division is too slow

Improving v0.1

```
template <typename T>
int bsearch_slow(queue<T> &v, T k) {
  if (v.size() == 1) {
    if (v.front() == k)
      return 1;
    return -1
  } else {
    if (v.front() == k)
      return 1;
    v.pop();
    int r = bsearch_slow(v,k);
    return (r == -1) ? r : r+1;
  }
}

int bsearch_slow(vector<T> &v, T k) {
  queue<T> q;
  for (auto &x : v) q.push(v);
  return bsearch_slow(q,k);
}
```

- Better subproblem division using queue
  - T(n) = T(n-1)+1
    - Solve into T(N) = O(N)
  - We also need O(N) (only one time) to convert a given array into a queue
    - Still result in O(N) total time
- We can do better!

Much better v0.1

ไม่มีการสร้าง Q ค่าหายจาก 0...n-1

```
template <typename T>
int bsearch_slow_2(vector<T> &v, T k,int start) {
  if (start == v.size() - 1) {
    if (v[start] == k) return start;
    return -1
  } else {
    if (v[start] == k) return start;
    return bsearch_slow_2(v,k,start+1);
  }
}

template <typename T>
int bsearch_slow_2(vector<T> &v, T k) {
  return bsearch_slow_2(v,k,0);
}
```

[start... v.size()-1]

แทนที่อะไรใหม่ สวยมากๆ

& สำคัญ!

0 - n-1

```
template <typename T>
int bsearch_slow_3(vector<T> &v, T k,int start) {
  if (v[start] == k) return start;
  if (start == v.size()-1) return -1;
  return bsearch_slow_3(v,k,start+1);
}

template <typename T>
int bsearch_slow_3(vector<T> &v, T k) {
  return bsearch_slow_3(v,k,0);
}
```

- We can DRY the code further
- Better code, but still T(N) = O(N), not faster than naïve method

แต่ก็ไม่มีการสร้าง Q

V0.1 Summary

$T(n) = T(n-1) + 0 \rightarrow O$

$T(n-1) = T(n-2) + 0$

- Use one smaller subproblem (less by 1) to help solve the original problem
- Need to divide
- Need to conquer
- Using correct data structure (or better indexing) helps getting better performance
- Pseudo-code can tell the idea more clearly but implementation details depends on expertise in programming

V0.2: Different Division

- V0.1 divides n into n-1, and solve the remaining 1
- Can we try divide A of size n into 2 array of size n/2?

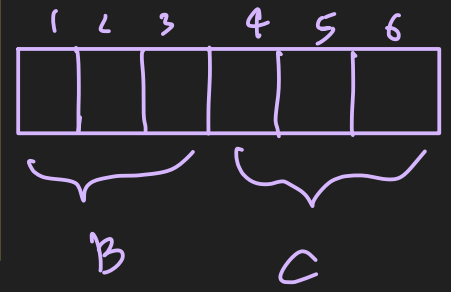
$$T(n) = \begin{cases} 2T(n/2) + O(1) & ;n > 1 \\ 1 & ;n = 1 \end{cases}$$

Assume that we use indexing technique to get B, C efficiently

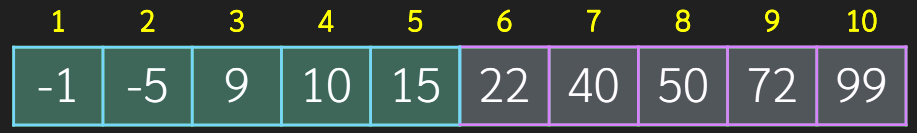
Can we use Master Method to solve this into O(N)?

```
def bsearch_half(A[1..n],k)
  if n == 1
    if A[1] == k
      return 1
    return -1
  m = n/2
  B=A[1..m]
  r = bsearch_half(B,k);
  if r != -1
    return r
  C=A[m+1..n]
  r = bsearch_half(C,k);
  if r != -1
    return r + m
  return -1
end
```

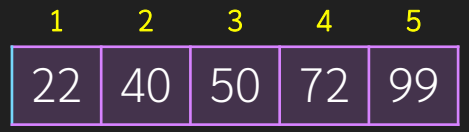
ถ้าโชคร้าย ทำสองรอบ



Divide into two subproblems of similar size



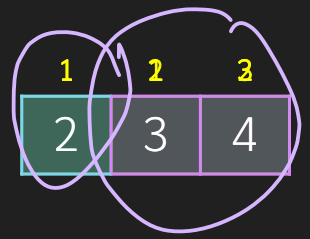
Subproblem 1

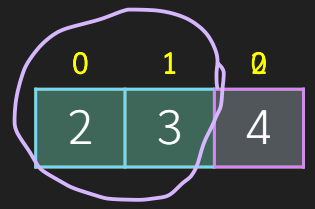


Subproblem 2

n = 3

m = 1





- Divide by m = n/2
  - Integer division
- Subproblem 1 = 1..m
- Subproblem 2 = m+1..n
- For odd number of n, starting index is relevant to the size of each subproblem

Recap

- Naïve (non-)Binary Search is a linear search using O(N)
- V0.1 is just a linear search written as a divide and conquer
  - The cost of divide has to be managed so that the recurrence relation is T(n) = T(n-1) + 1 = O(n)
- V0.2 tries to divide into 2 equal subproblems
  - T(n) = 2T(n/2) + 1 = O(n), different recurrence relation but same performance
- To get better performance, we need to reduce the number of subproblems

Actual Binary Search

- Utilize the fact that the data is sorted
- We will divide the problem into 2 parts: left and right
  - Check the middle point
  - only one part is solved, the other is discarded, depends on the value of the middle point

$$T(n) = \begin{cases} T(n/2) + O(1) & ; n > 1 \\ 1 & ; n = 1 \end{cases}$$

The result is T(n) = O(log n)

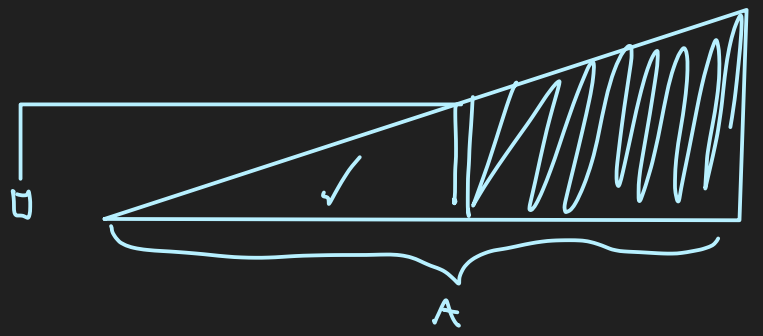
```
def bsearch(A[1..n],k)
  if n == 1
    if A[1] == k
      return 1
    return -1
  else
    m = n/2
    if A[m] => k
      r = bsearch(A[1..m],k)
    else
      r = bsearch(A[m+1..n],k)
      if r != -1
        r = r + m
    return r;
end
```

เหมือนเดิม

T(n/2) T(n/2)

ถ้าใช่ ก็ไม่ไปอีกที่

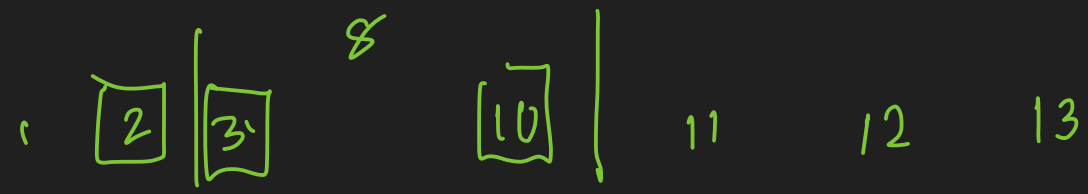
A[m] < k



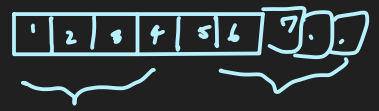
Actual Code

```
template <typename T>
int bsearch(vector<T> &v, T k,int start, int stop) {
  if (start == stop) return v[start] == k ? start : -1;
  int m = (start+stop) >> 1;  //bitwise shift right
  if (v[m] >= k) return bsearch(v,k,start,m);
  else return bsearch(v,k,m+1,stop);
}

template <typename T>
int bsearch(vector<T> &v, T k) {
  return bsearch(v,k,0,v.size()-1);
}
```

- Using moving index
- Non-recursive
- version exists



What's wrong with these code?



(a)

```
template <typename T>
int bsearch(vector<T> &v, T k,int start, int stop) {
  if (start == stop) return v[start] == k ? start : -1;
  int m = (start+stop) >> 1;  //bitwise shift right
  if (v[m] == k) return m;
  if (v[m] <= k) return bsearch(v,k,m+1,stop);
  if (v[m] > k) return bsearch(v,k,start,m-1);
}
```

(b)

```
template <typename T>
int bsearch(vector<T> &v, T k,int start, int stop) {
  if (start > stop) return -1;
  int m = (start+stop) >> 1;  //bitwise shift right
  if (v[m] >= k) return bsearch(v,k,start,m);
  else  return bsearch(v,k,m+1,stop);
}
```

```
template <typename T>
int bsearch(vector<T> &v, T k) {
  return bsearch(v,k,0,v.size()-1);
}
```

- Does it work correctly according to the problem definition?

Non recursive version

ไปลองใช้

- It is possible to write Binary Search as a loop

  $T(n) = T(n/2)$ ถ้ามีเวลาปิดได้

  - Maintain two variable, start and stop
  - Each iteration, start and stop move closer to each other
- When should be stop?
  - What happen when start = stop ?
  - Also when start+1 = stop?
- Try it yourself

Summary for Binary Search

- First example of divide and conquer
- Different dividing strategies may give different performance
  - Not necessarily better than naïve method
- Pseudo-code is better in providing idea and insight
- Actual code is better in analysis
- Care should be taken when converting pseudo-code into a code
- D&C Benefit can be achieved when we divide by half

แบ่งบางแบบอาจไม่ดี

Merge sort

D&C Sorting, easy divide

The Sorting Problem

- Input:
  - Array A[1..n] of n data (we must be able to compare a pair of them)
- Output:
  - The same array but re-arranged such that A[i] <= A[i+1] for every i from 1 to n-1
- Example instance
  - Input: [1,5,3,2,7,1]
  - Output: [1,1,2,3,5,7]

Sorting by Divide & Conquer

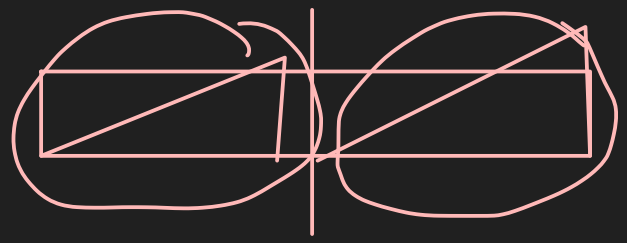
- We have iterative sorting that is $O(n^2)$ such as insertion sort or selection sort
- If, for D&C we get $T(n) = T(n-1) + O(n)$ → $O(n^2)$
- To go better than that, we need
  - $T(n) = T(n-1) + O(\log n)$ → $O(n \log n)$
  - $T(n) = 2T(n/2) + O(n)$ $n \log n$
- Either reduce by one and conquer in $O(\log n)$
  - Can we re-write our heapsort into a divide & conquer with $T(n) = T(n-1) + O(\log n)$ การลบ
- For Merge sort, it is $T(n) = 2T(n/2) + O(n)$

Merge Sort

- Invented by John von Neumann in 1945

Study under David Hilbert

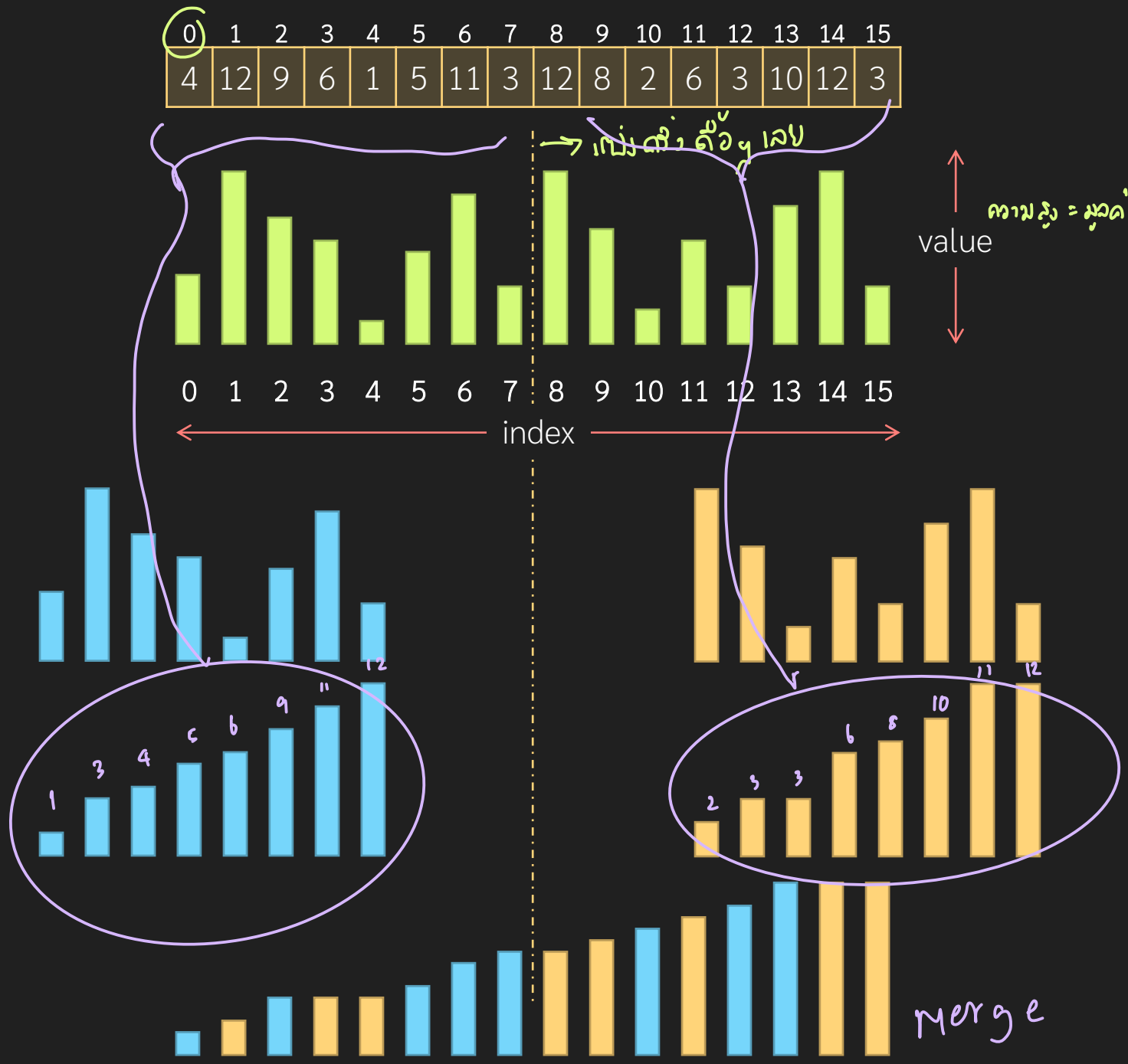
- Divide:



  - At middle point, into 2 non-overlapping subproblems
- Conquer:

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

  - Merge 2 sorted array into one



Example

- Divide at middle point
- Solve each part recursively
- Merge two sorted array

Pseudocode

เปรียบเทียบกับน้อย หน้าสองกอง

```
def merge_sort(A[1..n])
  if n == 1
    return A
  m = n/2
  B = merge_sort(A[1..m])
  C = merge_sort(A[m+1..n])
  A = merge(B,C)
end
```

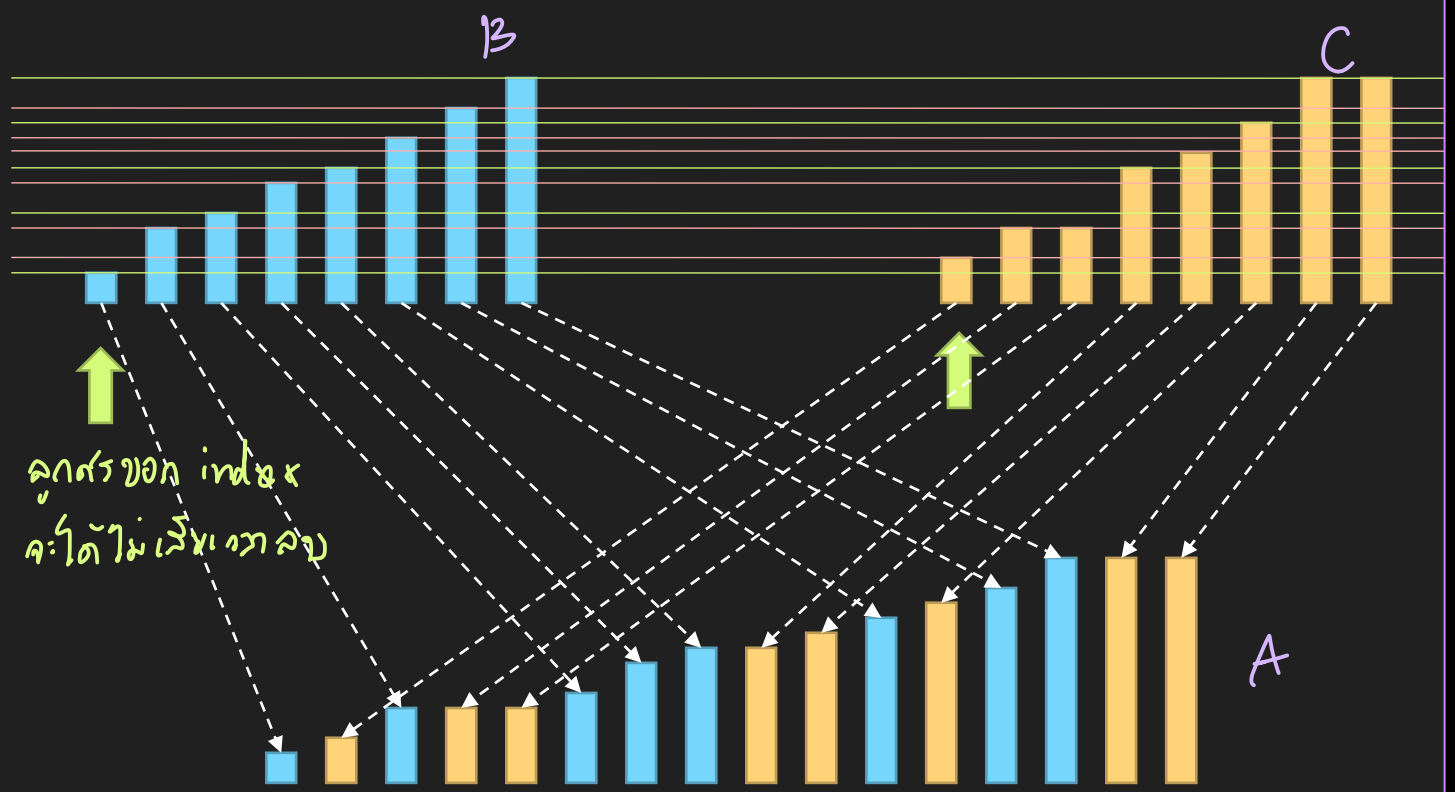
O(n)

- How to merge?
- Think of selection sort, we pick the maximum of the unsorted and move to front of sorted
  - For merge, our unsorted is actually two parts, each is already sorted
  - Picking maximum of the two sorted array is very simple, just compare the max of each array
  - Can also start from minimum

$$T(n) = \begin{cases} 2T(n/2) + T\_merge & ; n > 1 \\ 1 & ; n = 1 \end{cases}$$

The result is T(n) = O(n log n) //
if *T_merge* is O(n)

Merge




```
#B and C is sorted
def merge(B[1..m],C[1..n])
  A = []
  while (B is not empty || C is not empty)
    if (B is not empty && C is not empty)
      if B[1] < C[1]
        move front of B to the end of A
      else
        move front of C to the end of A
    else
      if (C is empty)
        move front of B to the end of A
      else
        move front of C to the end of A
  end
  return A
end
```

Merge is $\theta(n)$

Actual Code

```
template <typename T>
void merge(vector<T> &v,int start, int m, int stop,vector<T> &tmp) {
  bi = start; //index of B
  ci = m+1;   //index of C
  for (int i = start; i<= stop;i++) {
    if (ci > stop) { tmp[i] = v[bi++]; continue; }
    if (bi > m)    { tmp[i] = v[ci++]; continue; }
    tmp[i] = (v[bi] < v[ci]) ? v[bi++] : v[ci++];
  }
  for (int i = start; i<= stop;i++) v[i] = tmp[i];
}

template <typename T>
void merge_sort(vector<T> &v,int start, int stop,vector<T> &tmp) {
  if (start < stop) {
    int m = (start + stop) >> 1;
    merge_sort(v,start,m,tmp);
    merge_sort(v,m+1,stop,tmp);
    merge(v,start,m,stop,tmp);
  }
}
```

ข้ามไปรอบถัดไปเลย

O(n)

copy กลับ

ที่เก็บตัวแปร

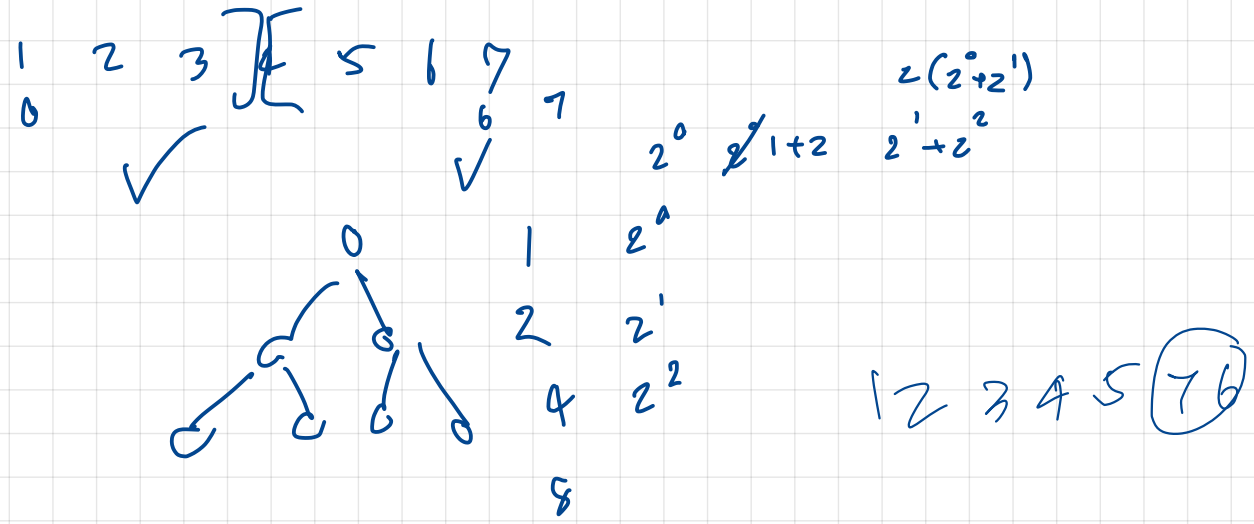
ผลรวม = V

$\frac{n}{5}$ $\frac{n}{6}$ $\frac{n}{5}$ $\frac{n}{5}$ $\frac{n}{5}$

$\left(\frac{n}{5}\right)^5 + 4\left(\frac{n}{5}\right)^3 \left(\frac{n}{5}\right.$

$2\left(\frac{n}{5}\right) + 1\left(n\right.$

$(5+4+3+2+1)\left(\frac{n}{5}\right)$



ขนาด 2^{k-1} ที่นำมาต่อกันเป็นไวรัสขนาด 2^k การต่อกันได้นั้นจะมีกฎสำคัญคือ **จำนวน**
ต่างจากจำนวน 1 ใน b เกิน 1 ตัว

เราต้องการตรวจสอบรหัสพันธุกรรมหลาย ๆ รหัส ที่ได้รับมาว่า ตรงกับไวรัส DOVIC

ข้อมูลนำเข้า

- บรรทัดแรกประกอบด้วยจำนวนเต็ม 2 ตัวคือ n และ k โดยที่ n ระบุจำนวนรหัสพันธุ
  บอกขนาดของรหัสพันธุกรรม โดยที่ 2 <= n <= 10 และ 1 <= k <= 8
- หลังจากนั้นอีก n บรรทัดจะเป็นรหัสพันธุกรรม บรรทัดละ 1 รหัส
  - แต่ละบรรทัดจะประกอบด้วยจำนวนเต็ม 2^k ตัว แต่ละตัวเป็นเลข 0 หรือ 1

ข้อมูลส่งออก

มี n บรรทัด เพื่อระบุว่ารหัสพันธุกรรมแต่ละรหัสเป็นไวรัส DOVIC-20 หรือเปล่า ให้พิ
และ “no” ถ้าไม่เป็น (ตัวพิมพ์เล็ก)

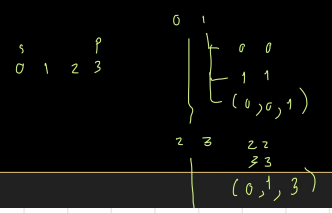
ตัวอย่าง

ข้อมูลนำเข้า	ข้อมูลส่งออก
5 2 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 1 0 1	yes no yes yes yes
4 3 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 0	no no yes no
3 4 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 1 1	no yes yes

ชุดข้อมูลทดสอบ

1) (30%) k <= 4
2) (70%) ไม่มีข้อจำกัดอื่นใด

```cpp
template <typename T>
void merge_sort(vector<T> &v,int start, int stop,vector<T> &tmp) {
  if (start < stop) {
    int m = (start + stop) >> 1;
    merge_sort(v,start,m,tmp);
    merge_sort(v,m+1,stop,tmp);
    merge(v,start,m,stop,tmp);
  }
}
```



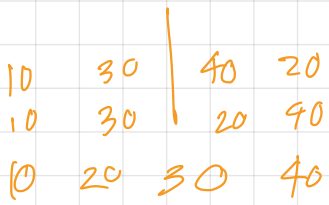
merge_sort (v, 0, 10, tmp)

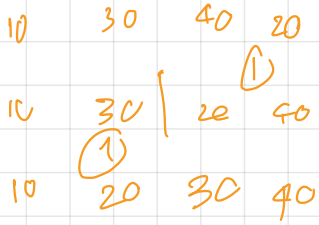
(v, 0, 5, tmp) (v, 6, 10, tmp)

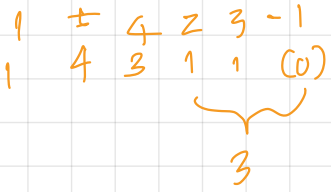
(v, 0, 2, tmp) (v, 3, 5, tmp) (v, 6, 8, tmp) (v, 9, 10)

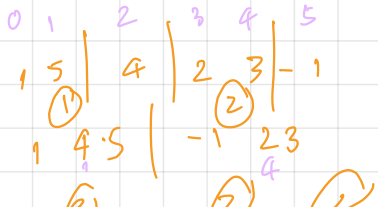
(v, 0, 1, tmp) (v, 2, 2, tmp) (v, 3, 4, tmp) (v, 5, 5, tmp) (v, 6, 7, tmp) (v, 8, 8, tmp) (v, 9, 9, tmp) (v, 10, 10, tmp)

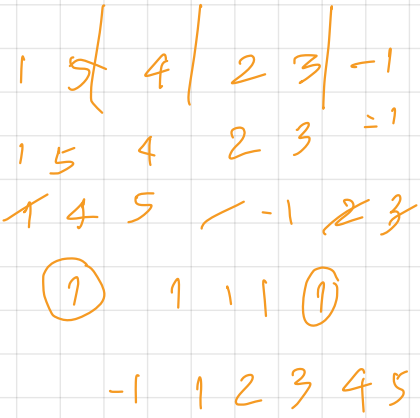
(v, 0, 0, tmp) (v, 1, 1, tmp) (v, 3, 3, tmp) (v, 4, 4, tmp) (v, 6, 6, tmp) (v, 7, 7, tmp)











More on Merge Sort

- Comparing to heap sort, merge sort requires more temporary space
  - Both is O(n log n)
- Merge sort is better for sorting linked list where random access is slow
  - Because it consider adjacent element
  - Used in external sorting and parallel sorting
- Sort in python is Timsort which is a hybrid of Insertion sort and merge sort
  - when sort small amount of data, use insertion, for larger, use merge

Question

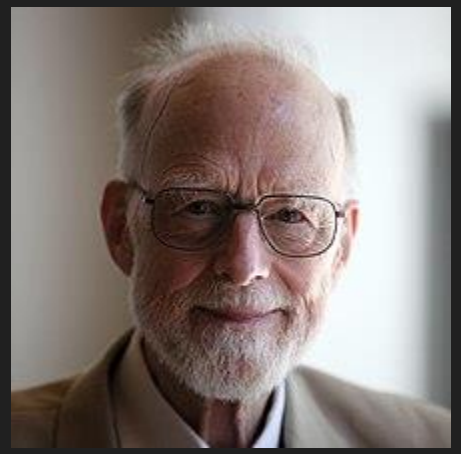
- Instead of divide by half, can we divide into 4 sub problem of same size?
  - T(N) = 4T(n/4) + O(n)
  - How to merge?
  - Better performance? $n \lg n$
- Can we divide into n/2 subproblem?
  - What is the complexity?

merge sort เขียนง่าย
แต่ช้า เสียเวลา
เสีย tmp spaces

Quick Sort

D&C Sorting, easy merge

Quicksort ⇒ ข้อดี tmp space น้อย + เร็วกว่า merge sort



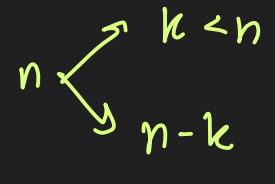
Study under
Andrey Kolmogorov

- Invented in 1959 by Sir Charles Antony Richard Hoare
- Merge sort need large temporary space, can we do better?
  - Instead of merge, can we use simplest possible conquer
- Quicksort achieve this by require additional condition for subproblem
  - We have to divide the problem smartly so that the subproblem satisfy this condition

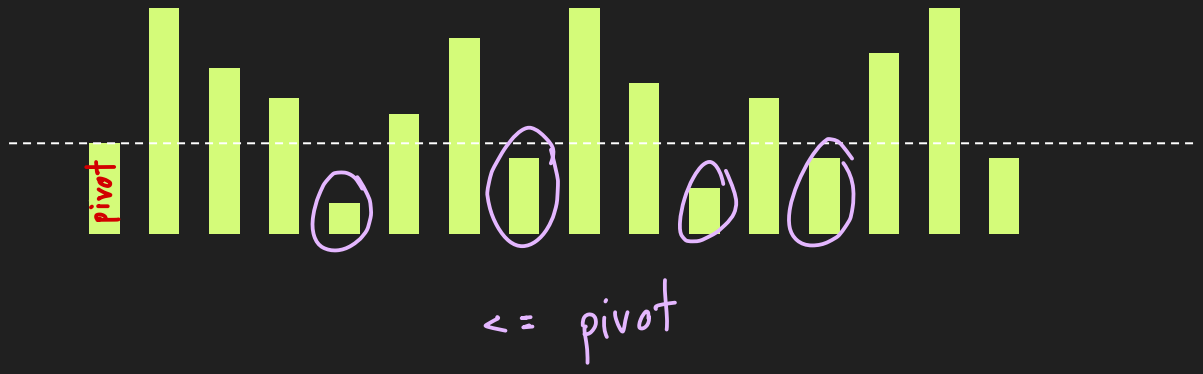
แทนที่ แบ่ง ปนป่วย แบบง่าย
แม่ง ยาก รวมง่าย

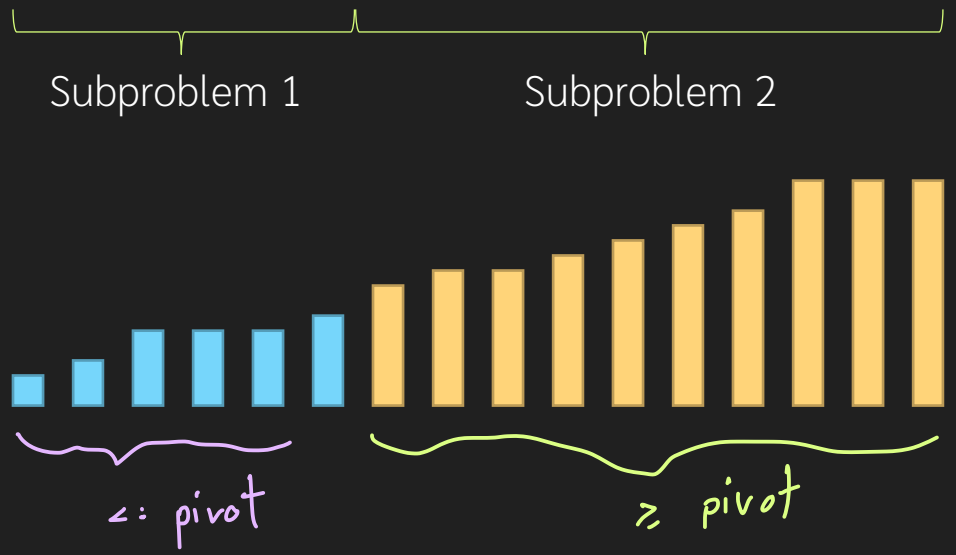
Quick Sort D&C

- Divide:
  - two subproblems, maybe different size
  - Additional condition
    - Subproblem 1 must contains only A[i] such that A[i] <= pivot
    - Subproblem 2 must contains only A[i] such that A[i] > pivot
- Conquer:
  - Because every data in the sub1 is less than elements of sub2, we can conquer by just append sub2 to sub1



Example





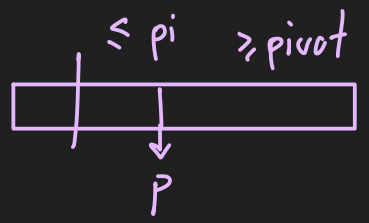
- Pick a partition
- Divide into <= partition part and >= partition part
- Sort recursively
- Just attach result together

Pseudocode

แทบไม่ต่างจาก merge

```
def quick_sort(A[1..n],start,stop)
  if start < stop
    p = partition(A,start,stop)
    quick_sort(A,start,p)
    quick_sort(A,p+1,stop)
end
```

q sort (A)
↳ quick_sort(A, 1, A.size)




```
def partition(A[1..n],start,stop)
  #can use anything except stop
  pivot = A[start]; เลือกตัวแรก

  left = start-1 นอกซ้าย
  right = stop+1
  while (left < right)
    do ทำก่อนค่อย check condition
      left = left + 1 เจอตัว >= ค่าหยุด
    until (A[left] >= pivot)
    do
      right = right - 1
    until (A[right] <= pivot)
    if (left < right) ใช่ก็สลับ
      swap(A[left],A[right])
    else
      return right
  end
  return right
end
```

O(n) ไม่ใช่ n^2

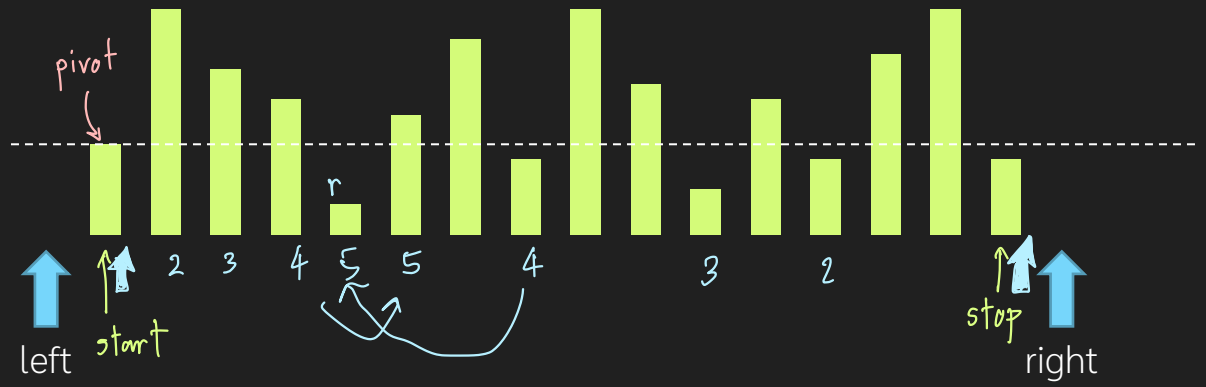
หยุด loop

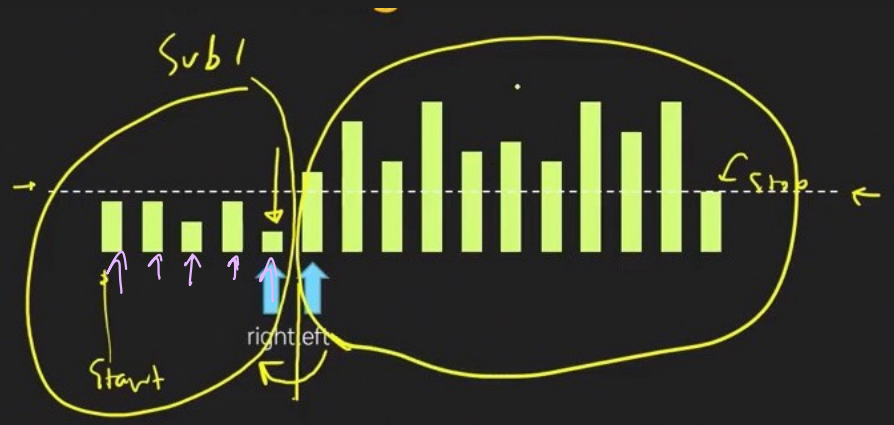
O(n)

ปี 1962

- Partition by Hoare's algorithm
- There is a simpler partitioning algorithm by Nico Lomuto ← หลัง คนมีเข้าใจง่ายกว่า ทางปฏิบัติช้ากว่านิดหน่อย
- Both is O(n)

Partitioning






```
def partition(A[1..n],start,stop)
  #can use anything except stop
  pivot = A[start];

  left = start-1
  right = stop+1
  while (left < right)
    do
      left = left + 1
    until (A[left] >= pivot)
    do
      right = right - 1
    until (A[right] <= pivot)
    if (left < right)
      swap(A[left],A[right])
    else
      return right
  end
  return right
end
```

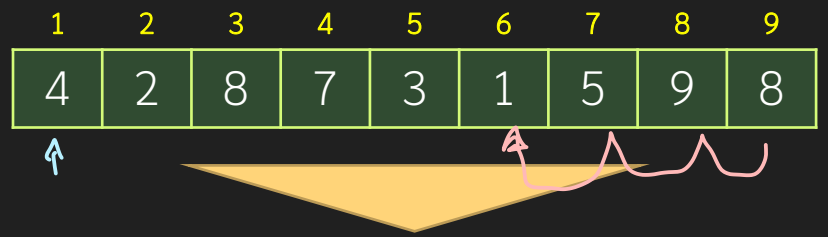
O(n)

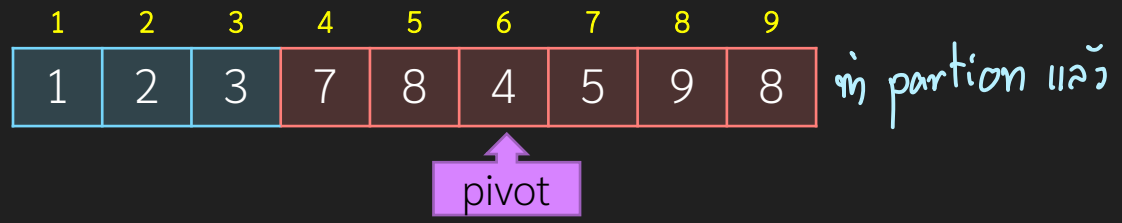
O(n)

→ O(n)

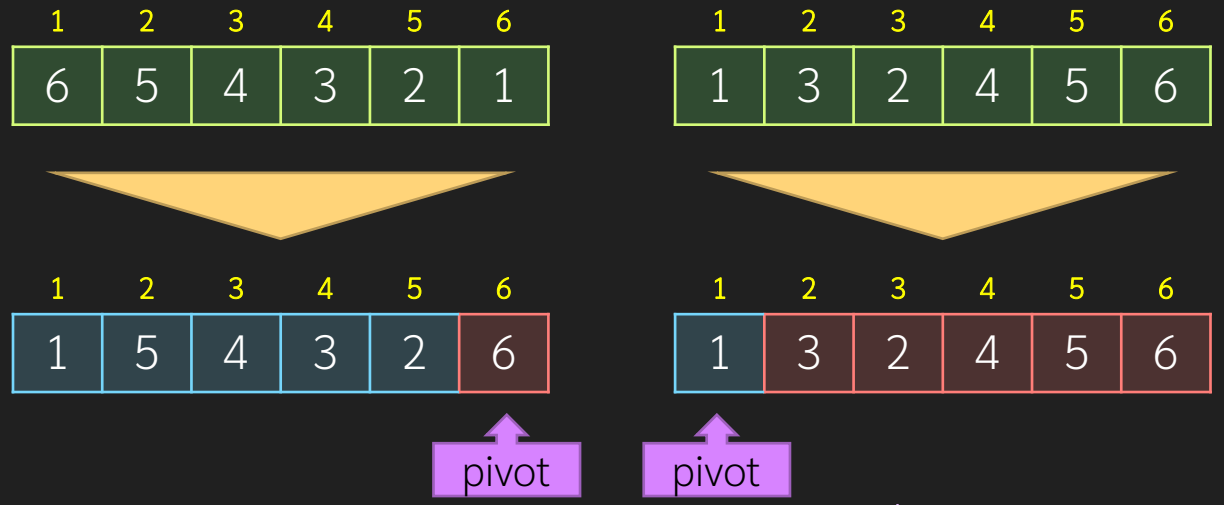
O(n)

Partitioning

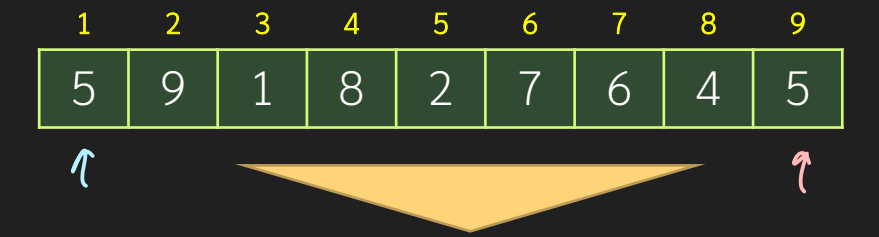


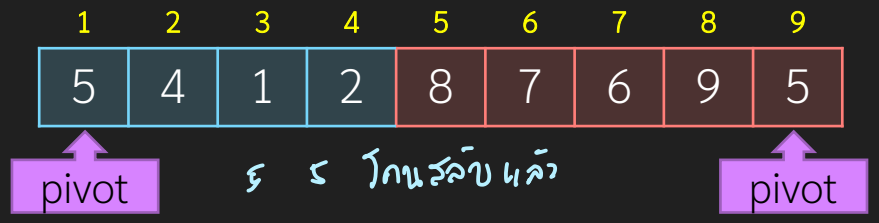


ที่ partion แล้ว

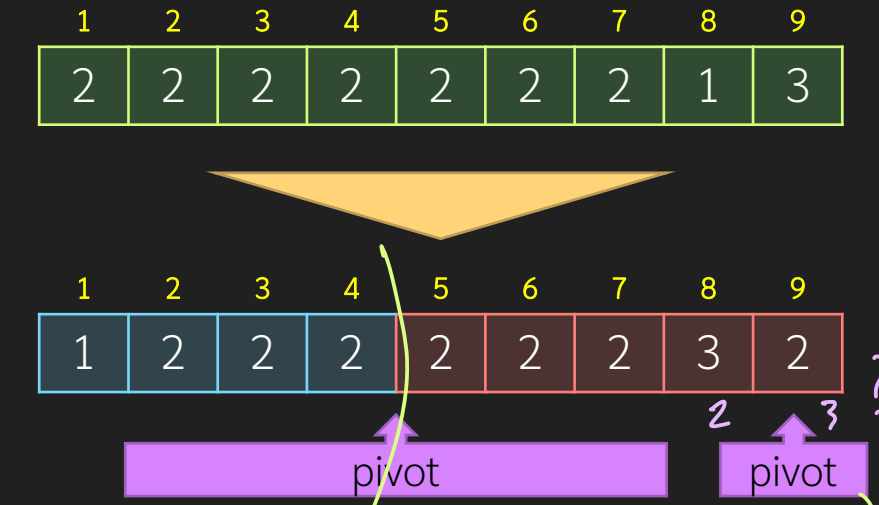


pivot อยู่ใกล้ที่สุดแล้ว





5 5 โดนสลับแล้ว



โดนto แบ่งไปด้วยรวย ถ้า แบ่งนี้

Analysis

$T(n) = T(k) + T(n-k) + O(n)$ (1 ชิ้น)

- We cannot directly use Master Method
  - Because we don't know that the subproblems actually equally divide
- Consider worst case and best case
- Worst case, each partition result in very imbalance subproblem
  - $T(n) = T(1) + T(n-1) + O(n)$
  - This is $O(n^2)$ worst case

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

$T(n-1) = T(n-2) + n \ldots$

- Best case, each partition is balanced
  - $T(n) = 2T(n/2) + O(n)$
  - This is $O(n \log n)$ ดีกว่า $n \log n$

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

Average Case

Let $f(n) = T_{avg}(n)$

↳ จะได้ไม่ต้องสับเยอะ

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

$$T_{avg}(n) = \frac{1}{n}\sum_{i=1}^{n-1}\left(T_{avg}(i) + T_{avg}(n-i)\right) + n$$

↳ เป็น n ไม่ได้ (pivot ไม่ชี้ตัวท้ายแล้ว)

$$= \frac{2}{n}\sum_{i=1}^{n-1} T_{avg}(i) + n$$

T(1) + T(4) → 2 3 / 3 2 / 4 1 → /4

n×

$$nf(n) = 2(f(1) + f(2) + \cdots + f(n-2) + f(n-1)) + n^2 \quad (1)$$

แทน n-1

$$(n-1)f(n-1) = 2(f(1) + f(2) + \cdots + f(n-2)) + (n-1)^2 \quad (2)$$

$$nf(n) - (n-1)f(n-1) = 2f(n-1) + n^2 - (n-1)^2 \quad (1)-(2)$$

$$nf(n) = (n+1)f(n-1) + 2n - 1$$

$$\frac{nf(n)}{n(n+1)} = \frac{(n+1)f(n-1)}{n(n+1)} + \frac{2n-1}{n(n+1)}$$

$$\frac{f(n)}{(n+1)} = \frac{f(n-1)}{n} + \frac{2n-1}{n(n+1)}$$

Average Case

$$\frac{f(n)}{(n+1)} = \frac{f(n-1)}{n} + \frac{2n-1}{n(n+1)}$$
$$\frac{f(n-1)}{(n)} = \frac{f(n-2)}{n-1} + \frac{2(n-1)-1}{(n-1)((n-1)+1)}$$
$$\cdots = \cdots$$
$$\frac{f(2)}{3} = \frac{f(1)}{2} + \frac{2(2-1)}{2(2+1)}$$

$$\frac{f(n)}{(n+1)} = \sum_{i=2}^{n} \frac{2i-1}{i(i+1)}$$
$$= 2\sum_{i=2}^{n} \frac{i}{i(i+1)} - \sum_{i=2}^{n} \frac{1}{i(i+1)}$$
$$= 2\sum_{i=2}^{n} \frac{1}{(i+1)} - \sum_{i=2}^{n} \frac{1}{i(i+1)}$$
$$\leq 2(\ln(n) + c)$$

something less than 0.5 because $\sum(\frac{1}{i(i+1)}) \neq 1$: $\frac{1}{2} + x$, $x = 0.5$

Partial sum harmonic series less than ln(k) + 1

$$\frac{f(n)}{(n+1)} = \ln(n) + c$$
$$f(n) = n\ln(n) + nc + c'$$
$$= \underline{\theta(n \log n)}$$

Quicksort's average case is θ(n log n)

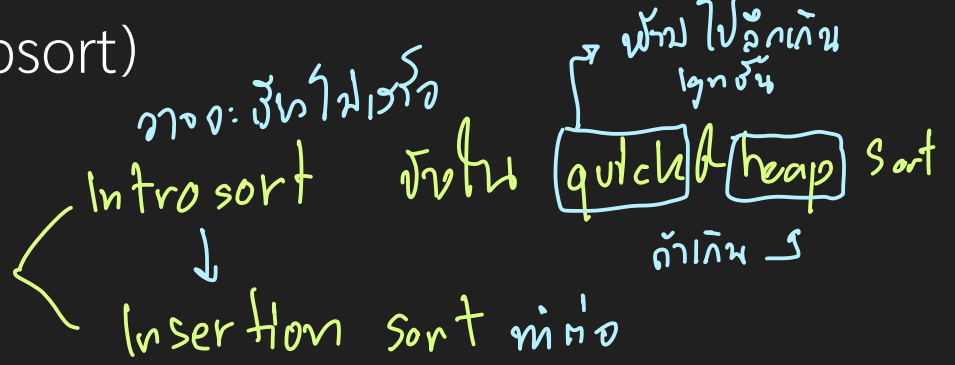
How likely is the worst case เกิดขึ้นบ่อยไหม?

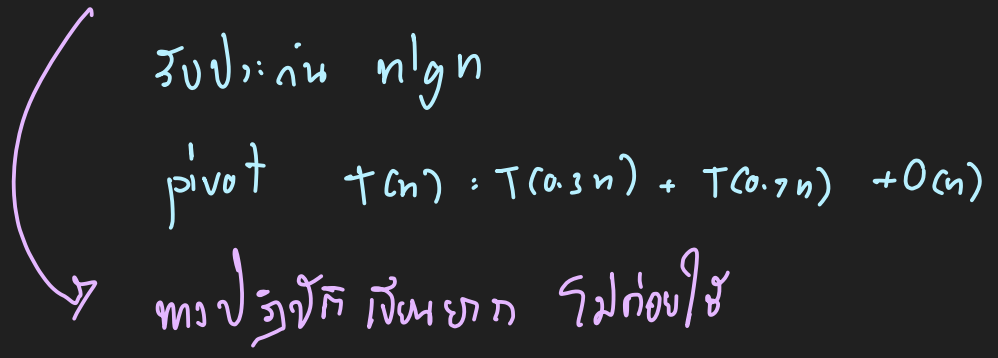
เกิดขึ้นง่ายมาก ถ้าข้อมูลเรียงไว้แล้ว แล้วเอา pivot เป็นตัวแรก

- If we pick pivot by the first element (or the last element) and the data is sorted (either ascending or descending), it's worst case
- Can be practically avoided by picking pivot at random แก้
  - With Hoare's algorithm, do not pick a pivot at the last element
    - ↳ อย่าเอาตัวท้าย

Dealing with quicksort uncertainty

- Sort of Gnu C++ use hybrid sort (similar to Timsort in Python)
  - Hybrid of Introsort follow by Insertion Sort
  - Introsort is a hybrid of Quicksort + Heapsort (when depth of recursion exceed some limit, we convert to heapsort)
- Better pivot selection
  - Median-of-median-of-five



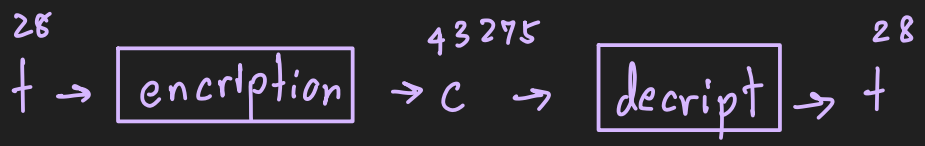


Modulo Exponentiation

Calculating $a^n \bmod k$

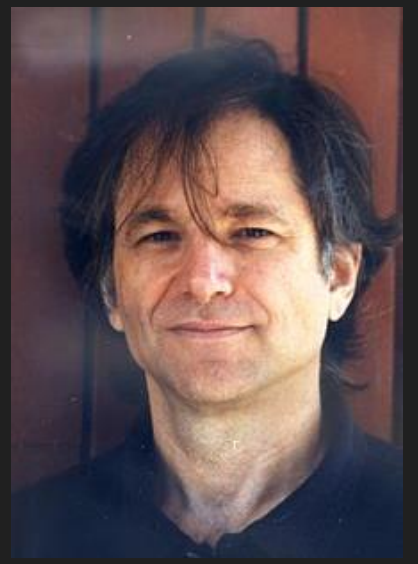
Problem

- Calculate a^n mod k
- Input:
  - Three positive integers a, n and k
- Output:
  - The value of a^n mod k
- Example instance
  - a = 2, n = 92, k = 10 output: 4



Usage

- Public key cryptography, especially RSA algorithm
- In RSA, we need 3 things, m, e and d
  - m is calculated from p*q where p and q are large prime number
  - e is any integer from 1 to LCM(p-1,q-1)
  - d is a modular inverse of e (in mod LCM(p-1,q-1))
    - Very easy to calculate d if we know e, p,q
    - Very hard to calculate d if we know only e and m (but not p, q)
- Encrypt a value t as $c = t^e \bmod m$
- Decrypt a value c by $t = c^d \bmod m$



Divide & Conquery an mod k

- Observe that $(a \cdot b) \bmod k = [(a \bmod k) \cdot (b \bmod k)] \bmod k$
- Lets n = x + y
  - Then, $a^n \bmod k = a^{(x+y)} \bmod k = (a^x \bmod k) * (a^y \bmod k) \bmod k$
- In other words, to calculate $a^n \bmod k$, we need to calculate
  - $a^x \bmod k$ and
  - $a^y \bmod k$
  - Now, let x = n/2

$$a^n = \begin{cases} a^x * a^x, & x \text{ is even} \\ a^x * a^x * a, & x \text{ is odd} \end{cases}$$

Pseudocode

```
def mod_expo(a,n,k)
  if n == 1
    return a mod k
  if n mod 2 == 0  วงต้น : คู่
    tmp = mod_expo(a,n/2,k) ยกกำลัง
    return (tmp * tmp) mod k
  else
    tmp = mod_expo(a,n/2,k)
    tmp = (tmp * tmp) mod k
    return (tmp * (a mod k)) mod k
  end
end
```

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

Example $2^{92} \bmod 10$

$2^{92} = 4951760157141521099596496896$

$2^{92} = 2^{46} \times 2^{46}$ $= 4 \times 4 \bmod 10 = 6$

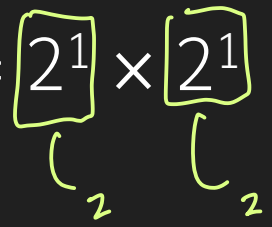
$2^{46} = 2^{23} \times 2^{23}$ $= 8 \times 8 \bmod 10 = 4$

$2^{23} = 2^{11} \times 2^{11} \times 2$ $= 8 \times 8 \times 2 \bmod 10 = 8$

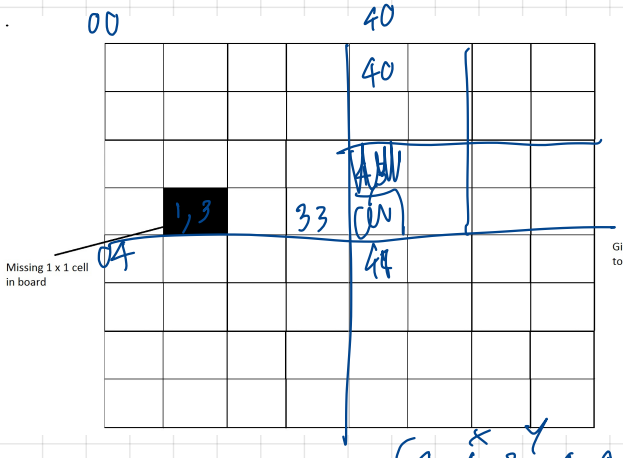
$2^{11} = 2^{5} \times 2^{5} \times 2$ $= 2 \times 2 \times 2 \bmod 10 = 8$

$2^{5} = 2^{2} \times 2^{2} \times 2$ $= 4 \times 4 \times 2 \bmod 10 = 2$

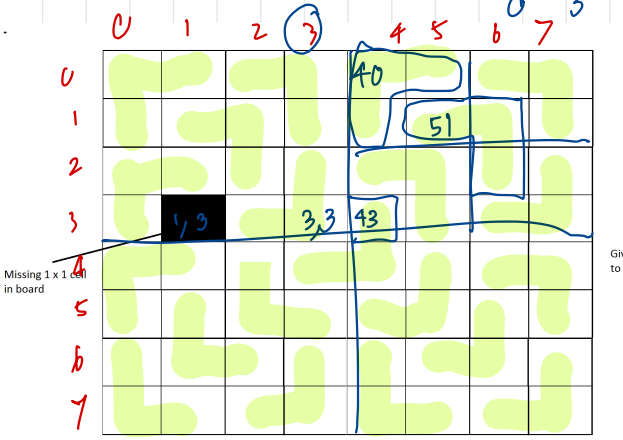
$2^{2} = 2^{1} \times 2^{1}$ $= 2 \times 2 \bmod 10 = 4$



03



(4, 4, 3, 4, 0)



(2, 5, 1, 4, 0)

Maximum Sum of Subarray

The problem

- Given array A[1..n] of numbers, may contain negative number
  - Find a non-empty subarray A[p..q] such that the summation of the values in the subarray is maximum
- Input:
  - A[1..n]
- Output:
  - p and q, where 1 <= p <= q <= n and summation of A[p..q] is maximum
- Example:
  - A = [1, 4, 2, 3] output: 1 and 4
  - A = [-2, -1, -3, -5] output: 2 and 2
  - A = [2, 3, -6, 4, -2, 3, -5, -4, 3] output: 4 and 6

Naïve $O(n^3)$

```
def mss_naive(A[1..n])
  max = A[1]
  for p from 1 to n
    for q from p to n
      sum = 0
      for j from p to q
        sum += A[j]
      if (sum > max)
        max = sum
  return max
end
```

- Try all possible $O(n^2)$ subarray
- Need O(n) per subarray to calculate the summation

$\Rightarrow O(n^3)$

Using prefix sum to reduce innermost loop

↳ O(n²)

```
def mss_prefix_sum(A[1..n])
  let S be an array [0..n]
  S[0] = 0;
  sum = 0;
  for i from 1 to n
    sum = sum + A[i]
    S[i] = sum


  max = A[1]
  for p from 1 to n
    for q from p to n
      sum = S[q] - S[p-1]
      if (sum > max)
        max = sum
  return max
end
```

- Let s[k] = $\sum_{i=1}^{k} A[i]$
  - Also S[0] = 0

S[1] : A[1] S[2] = A[1] + A[2]

- Calculating $\sum_{i=p}^{q} A[i]$ is just S[q] – S[p-1]

$a_1 + a_2 + \ldots a_q - a_1 + a_2 + \ldots + a_{p-1}$

$a_p - a_q$

	1	2	3	4	5	6	7	8
A	3	-4	1	2	-1	-1	5	-4

1+2+-1-1+5 = 6

	0	1	2	3	4	5	6	7	8
S	0	3	-1	0	2	1	0	5	1

5 - -1 = 6

D&C

- Try divide at the middle point
- Divide:
  - m = n/2
  - A[1..n] into A[1..m] and A[m+1..n]
- Conquer:
  - The answer from A[1..m] is p_1, q_1 in range [1..m]
  - The answer from A[m+1..n] is p_2, q_2 in range [m+1..n]
  - Bu we need to consider p in range [1..m] while q is in range [m+1..n]

D&C v0.1

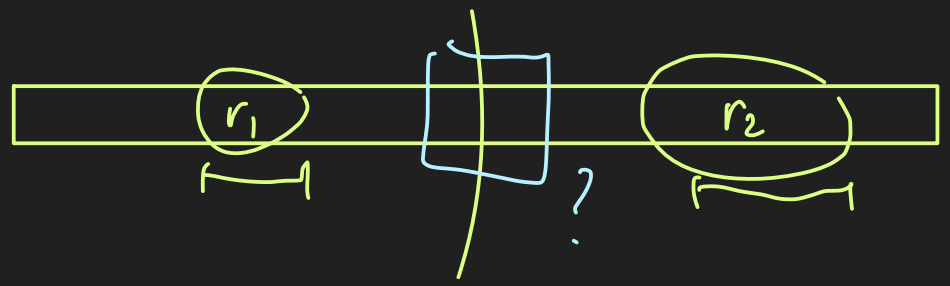
$T(n) : 2T(\frac{n}{2}) + O(n^2)$

$: O(n^2)$

```
def mss1(A[1..n],start,stop,S)
  if (start == stop)
    return A[start]
  m = (start+stop) / 2
  r1 = mss1(A,start,m)
  r2 = mss1(A,m+1,stop)
  r3 = A[start]
  for p from 1 to m
    for q from m+1 to n
      sum = S[q] - S[p-1]
      if (sum > r3)
        r3 = sum
  return max(r1,r2,r3)
end
```

- Notice that $\sum_1^n A[i] = \sum_1^{n-1} A[i] + A[n]$
- Hence, $S[i] = S[i-1] + A[i]$

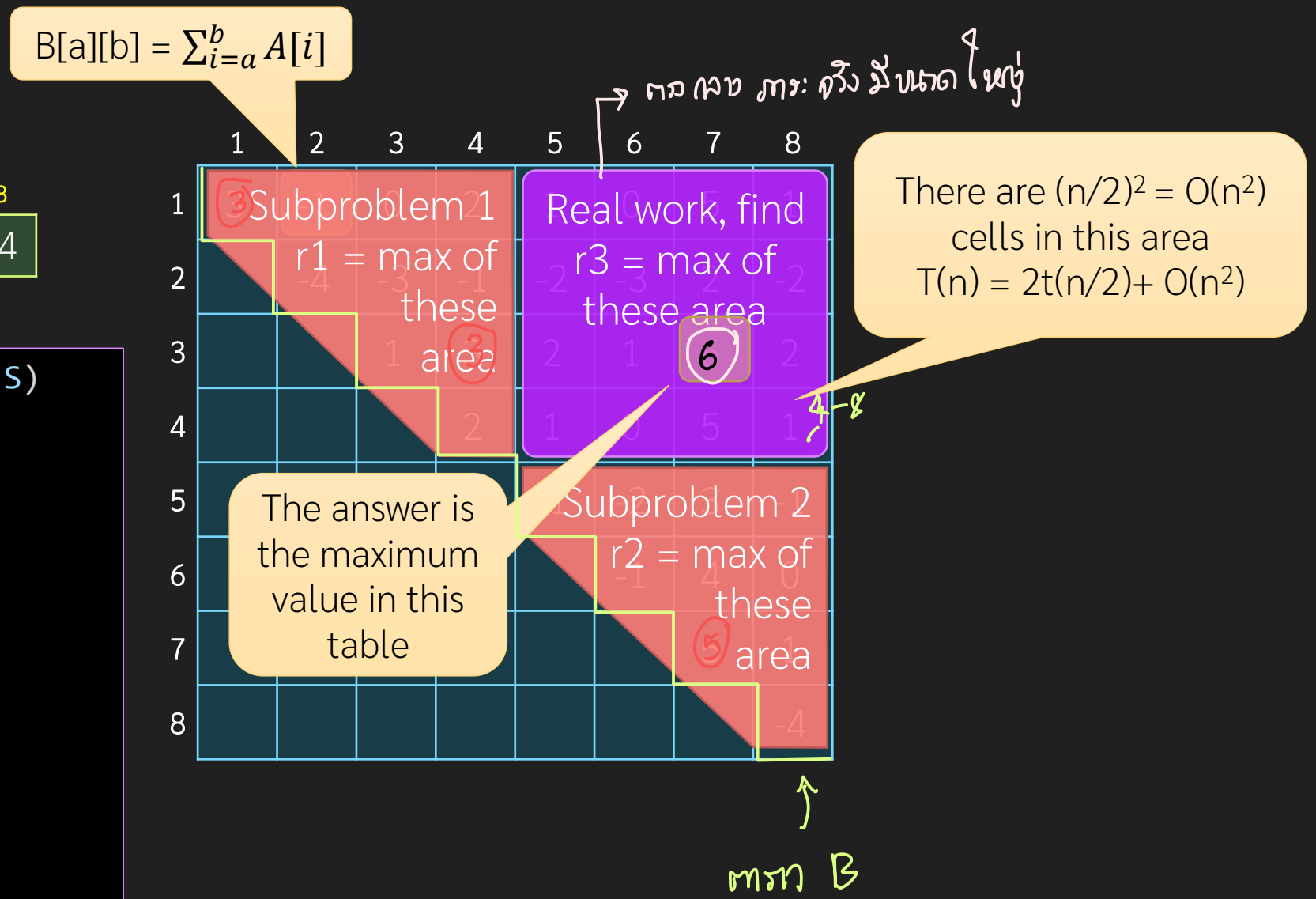
```
def mss1(A[1..n])
  let S be an array [0..n]
  S[0] = 0;
  for i from 1 to n
    S[i] =  S[i-1] + A[i]

  mss1(A,1,n,S)
end
```



Visualization

	1	2	3	4	5	6	7	8
A	3	-4	1	2	-1	-1	5	-4

```
def mss1(A[1..n],start,stop,S)
  if (start == stop)
    return A[start]
  m = (start+stop) / 2

  r1 = mss1(A,start,m)
  r2 = mss1(A,m+1,stop)

  r3 = A[start]
  for p from 1 to m
    for q from m+1 to n
      sum = S[q] - S[p-1]
      if (sum > r3)
        r3 = sum
  return max(r1,r2,r3)
end
```

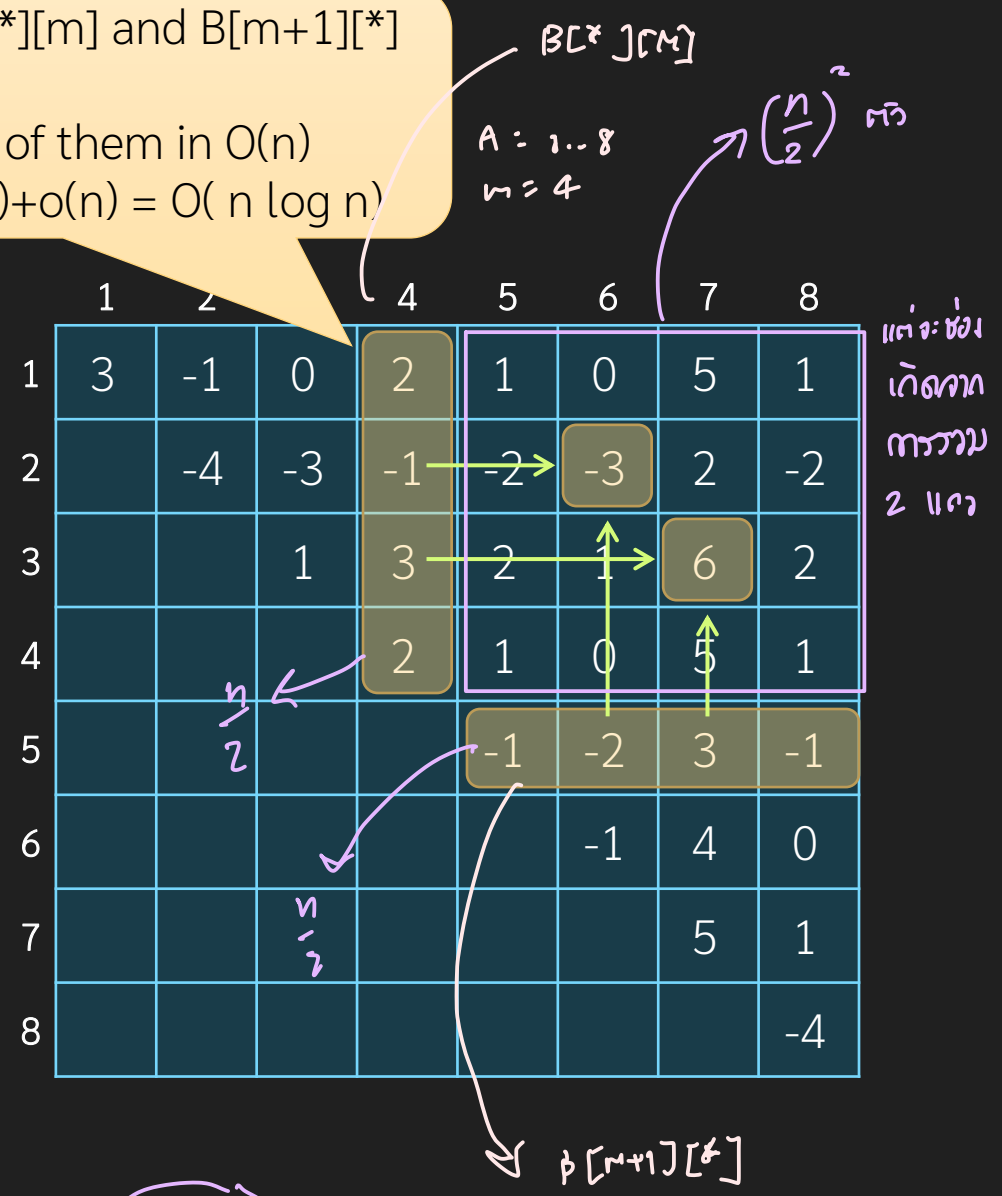


Actual Version

There are only n/2 B[*][m] and B[m+1][*]

We can find max of them in O(n)
Hence T(n) = 2T(n/2)+o(n) = O(n log n)

- Reduce real work from $O(n^2)$ to $O(n)$
- Notice that $B[a][b] = B[a][k] + B[k+1][b]$
- Real work find max when a is from 1..m and b = m+1..n
  - If we calculate B[*][m] and B[m+1][*]
  - Any B[a][b] in these range can be calculated from B[a][m] + B[m+1][b]
- Hence $\max_{\substack{1\le a\le m \\ m+1\le b\le n}} B[a][b] = \max_{1\le a\le m} B[a][m] + \max_{m+1\le b\le n} B[m+1][b]$

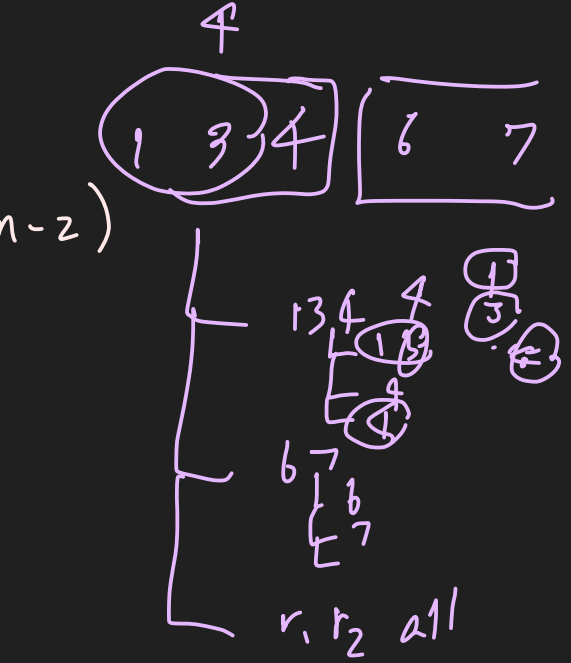


Pseudocode

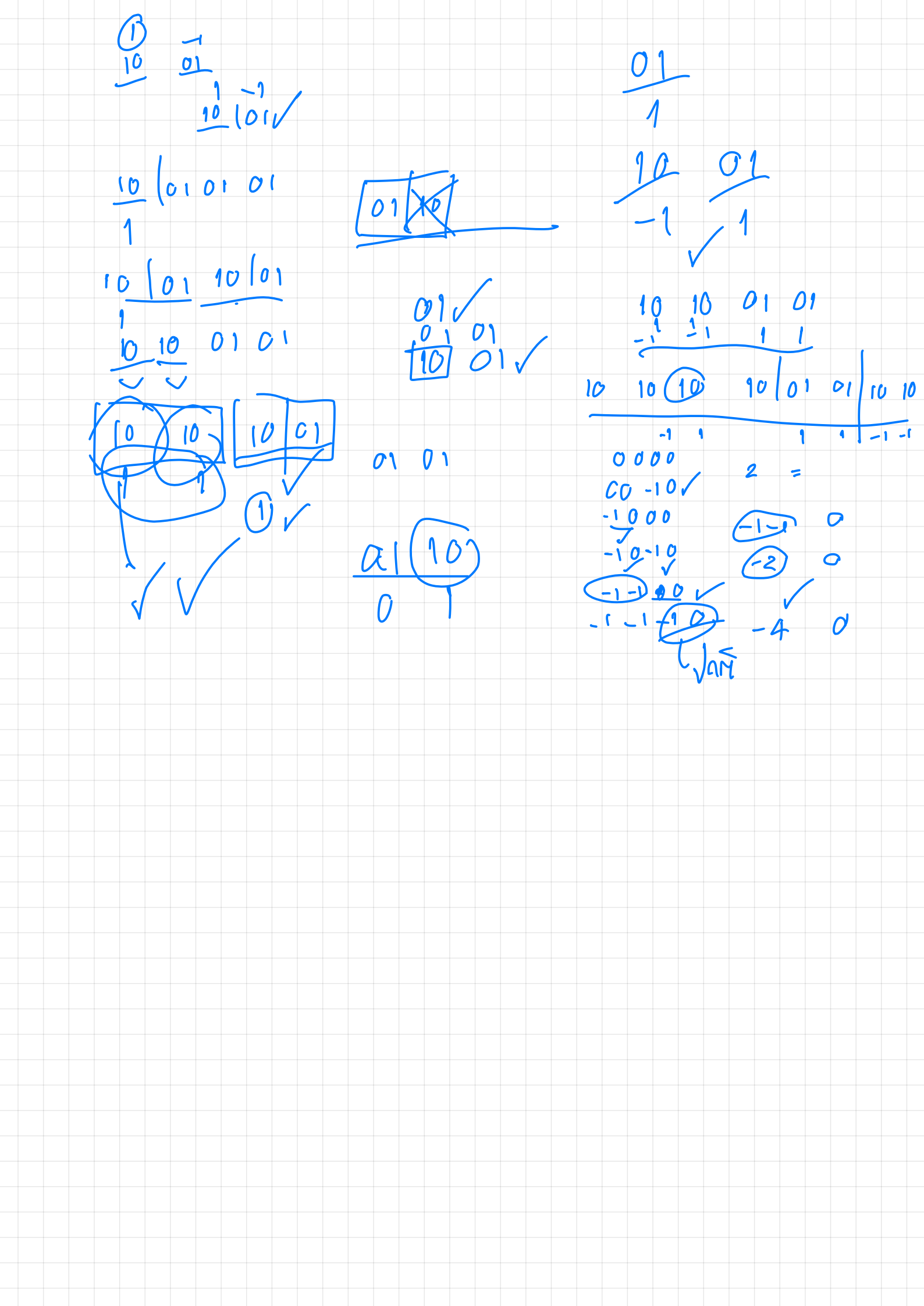
```
#this return sum A[a..b] in O(1)
def get_sum(S,a,b)
  return S[b] - S[a-1]
end
```

```
def mss(A,start,stop,S)
  if (start == stop)
    return A[start]
  m = (start+stop) / 2

  r1 = mss(A,start,m,S)
  r2 = mss(A,m+1,stop,S)

  #find max of B[start..m][m]
  max_sum_left = get_sum(S,m,m)
  for i in start to m-1
    max_sum_left = max(max_sum_left,get_sum(S,i,m))

  #find max of B[m+1..stop][stop]
  max_sum_right = get_sum(S,m+1,m+1)
  for j in m+2 to stop
    max_sum_right = max(max_sum_right,get_sum(S,m+1,j))

  r3 = max_sum_left + max_sum_right
  return max(r1,r2,r3)
end
```

- Real work part is O(n)

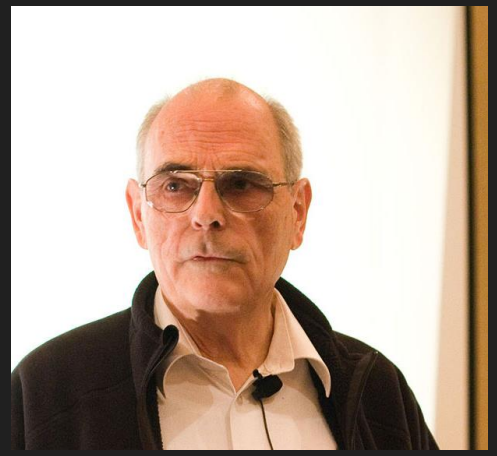


$2T(\frac{n}{2}) + O(n) \Rightarrow n \lg n$



Strassen’s Matrix Multiplication

The Problem



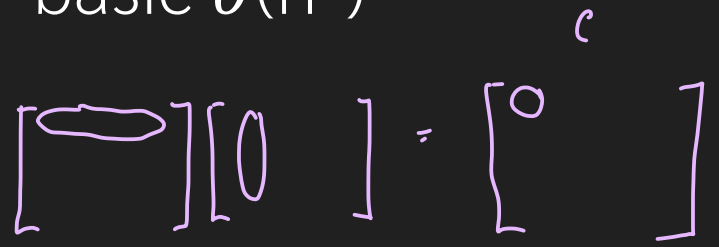
- Invented by Volker Strassen
- Input:
  - two square matrices of the same size
  - A[1..n][1..n] and B[1..n][1..n]
- Output:
  - The multiplication of A and B
  - C = AB

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Square matrix size n

Naïve Method

- basic $\theta(n^3)$



```
def multi(A[1..n][1..n],B[1..n][1..n])
  let C[1..n][1..n] be a matrix of size n * n
  for i in 1 to n
    for j in 1 to n
      sum = 0
      for k in 1 to n
        sum += a[i][k] * b[k][j];
      C[i][j] = sum
end
```

$O(n^3)$

ไอเดียเบื้องต้น

Block Multiplication

- Divide Matrix into blocks
  - $A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$ แบ่ง A ออกเป็น Matrix ย่อย
  - $B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$
  - $C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$
- Calculate multiplications of blocks

แต่ละอัน เกิดจากการคูณ 2 ครั้ง $\left(\frac{n}{2}\right)^2$

  - $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$
  - $C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$
  - $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$
  - $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$

$A =$ (8×8)

$$\begin{bmatrix} \overbrace{\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}}^{A_{1,1}} & \overbrace{\begin{matrix} a_{15} & a_{16} & a_{17} & a_{18} \\ a_{25} & a_{26} & a_{27} & a_{28} \\ a_{35} & a_{36} & a_{37} & a_{38} \\ a_{45} & a_{46} & a_{47} & a_{48} \end{matrix}}^{A_{1,2}} \\ \underbrace{\begin{matrix} a_{51} & a_{52} & a_{53} & a_{54} \\ a_{61} & a_{62} & a_{63} & a_{64} \\ a_{71} & a_{72} & a_{73} & a_{74} \\ a_{81} & a_{82} & a_{83} & a_{84} \end{matrix}}_{A_{2,1}} & \underbrace{\begin{matrix} a_{55} & a_{56} & a_{57} & a_{58} \\ a_{65} & a_{66} & a_{67} & a_{68} \\ a_{75} & a_{76} & a_{77} & a_{78} \\ a_{85} & a_{86} & a_{87} & a_{88} \end{matrix}}_{A_{2,2}} \end{bmatrix}$$

4 × 4

Matrix addition is $O(n^2)$

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

$= O(n^3)$

↳ บวกได้ แถว + คอลัมน์

กระทำการเรียกซ้ำ

C $\begin{bmatrix} C_{1,1} & \\ & \end{bmatrix}$

ลดการคูณ การเรียกซ้ำ

Strassen's Algorithm

ต่อไปนี้ matrix ย่อย ของผลเฉลยของ Matrix AxB

- Compute $M_1 \dots M_7$
  - $M_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$
  - $M_2 = (A_{2,1} + A_{2,2})(B_{1,1})$
  - $M_3 = (A_{1,1})(B_{1,2} - B_{2,2})$
  - $M_4 = (A_{2,2})(B_{2,1} - B_{1,1})$
  - $M_5 = (A_{1,1} + A_{1,2})(B_{2,2})$
  - $M_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$
  - $M_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$

สูตร ไม่ต้องจำ มีการคูณแค่ครั้งเดียว

- Note that each M can be computed by one single multiplication of n/2 * n/2 matrices, with 1 or 2 addition of a matrix

$M_1 = (A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{12}B_{11})$

$M_4 = A_{22}B_{21} - A_{22}B_{11}$

$M_5 = A_{11}B_{22} + A_{12}B_{22}$ ที่เหลือคัดมาหมด

$M_7 = A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22}$

- The result can be computed as
  - ไม่มีการคูณ
  - $C_{1,1} = M_1 + M_4 - M_5 + M_7$
  - $C_{1,2} = M_3 + M_5$
  - $C_{2,1} = M_2 + M_4$
  - $C_{2,2} = M_1 + M_2 + M_3 + M_6$
- Total of 7 matrix multiplications and 18 matrix addition

อันเก่า คูณ 8 บวก 4

Analysis

- $T(n) = 7T(n/2) + 18(n/2)^2$
  - Using Master's method
  - $a = 7, b = 2, f(n) = O(n^2)$
    - $c = \log_2 7 \approx 2.807$
  - Hence, $T(n) = O(n^{2.807})$

$O(n^2)$

จาก $O(n^3)$ ก็เร็วขึ้น

ใน ทางปฏิบัติ เขียนยาก

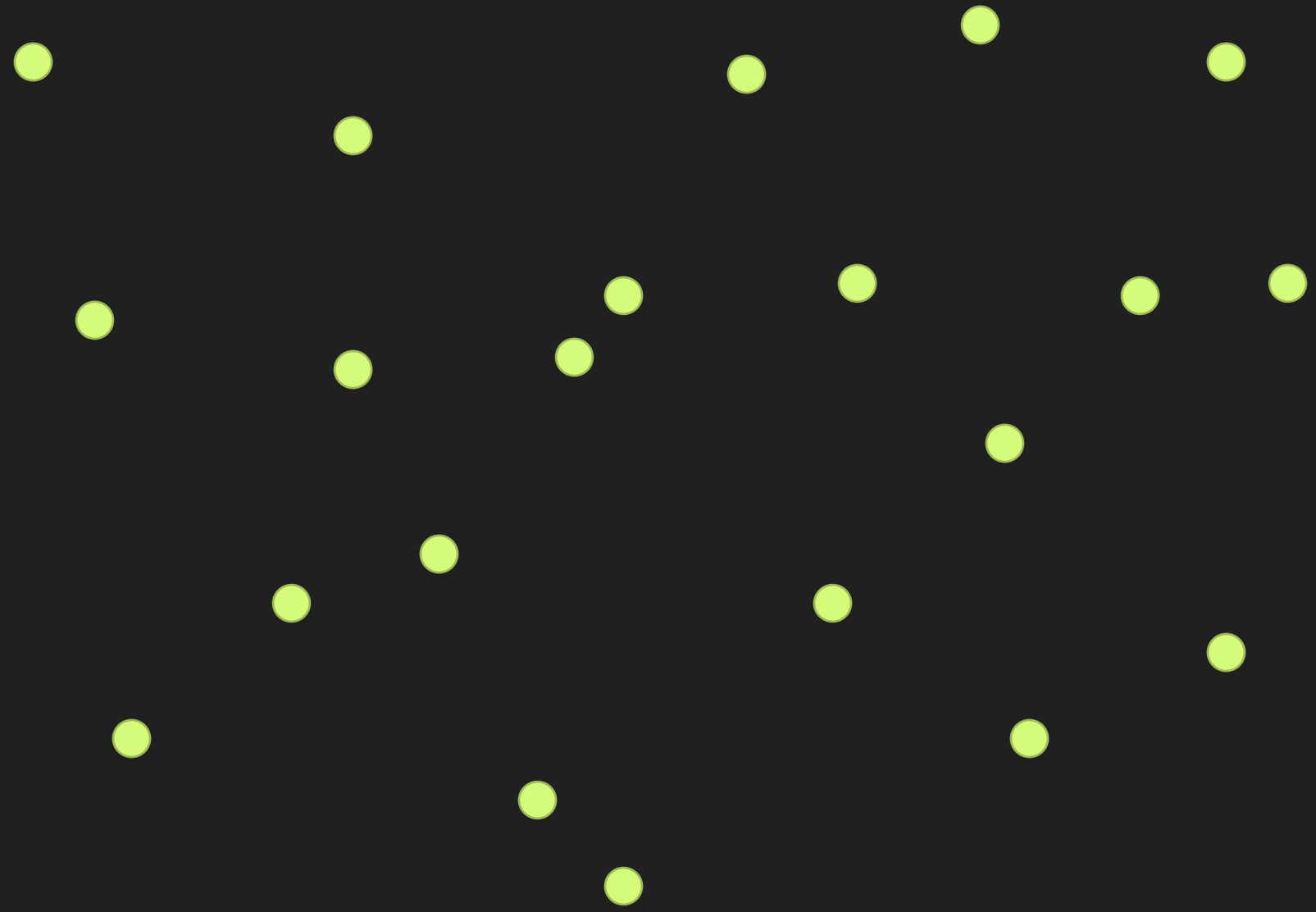
Year	Algorithm	Complexity
1969	Strassen	$O(n^{2.807})$
1987	Coppersmith–Winograd	$O(n^{2.375})$ เร็วขึ้น
2010	Improved CW (by Andrew Stothers)	$O(n^{2.374})$
2011	Further improvement of CW (by Virginia Vassilevska Williams)	$O(n^{2.3728642})$
2014	Improve over Williams' (by François Le Gall)	$O(n^{2.3728639})$

Closest Pair

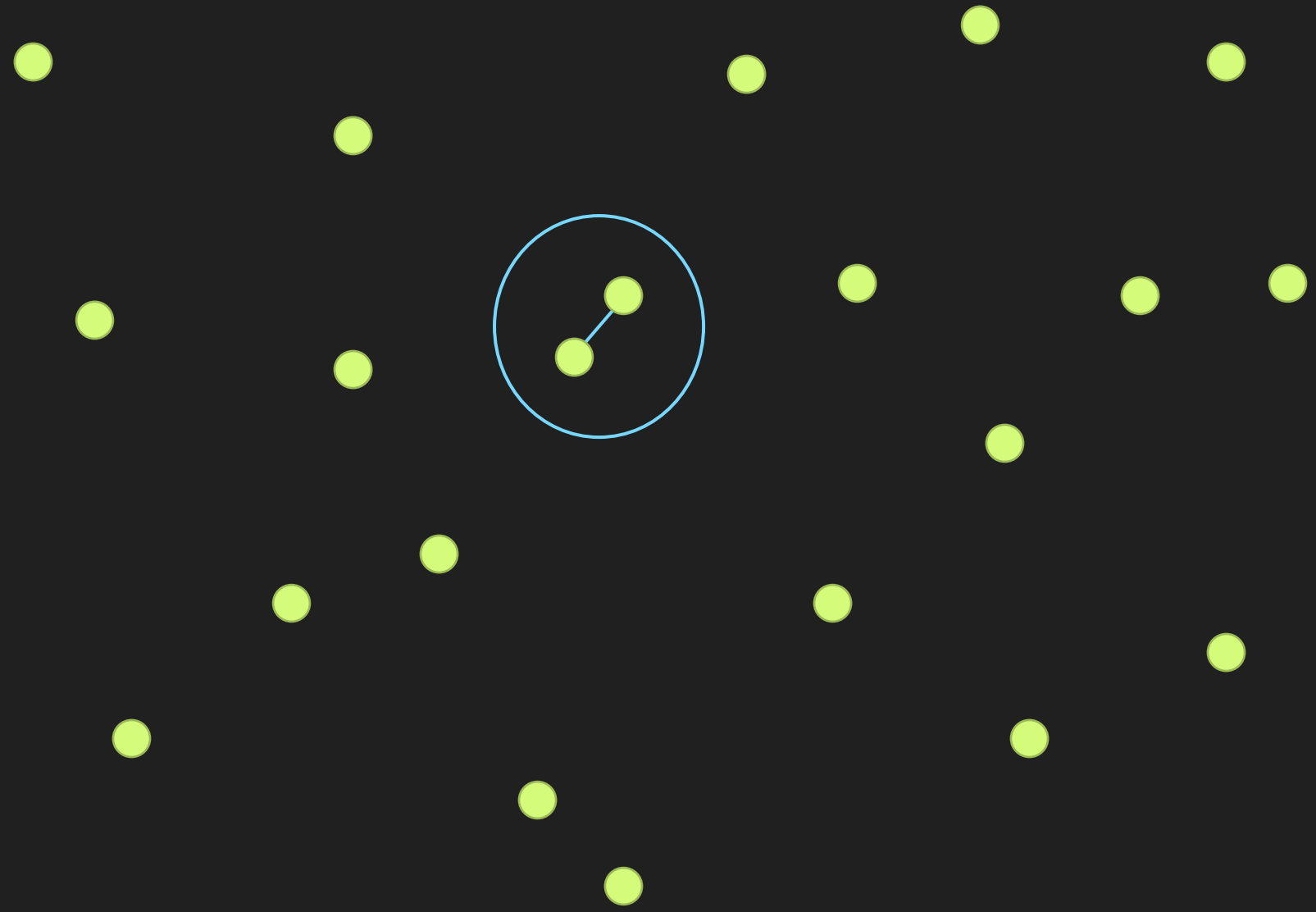
The Problem

- Input:
  - Coordinates of n points in 2D
    - $(x_1,y_1), (x_2,y_2), \ldots ,(x_n,y_n)$ คู่ที่ระยะห่างของจุดนั้นน้อยสุด
- 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 $n(n+1)/2$ pairs
  - Compute the distance of each pair
    - Takes $\Theta(1)$ for each pair

- In total, it is $\boxed{\Theta(n^2)}$

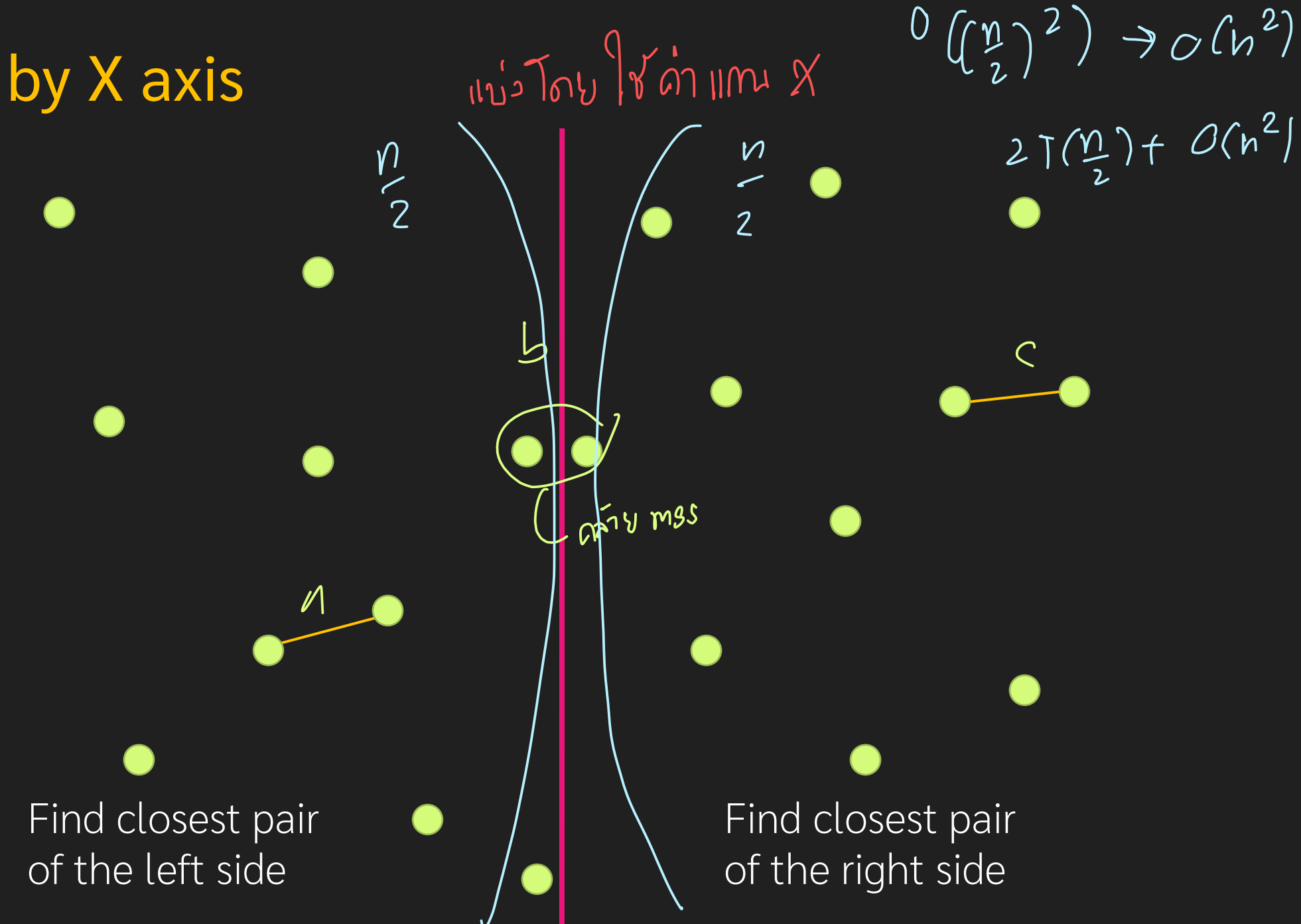
DC approach

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

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

$$= T(n \lg n)$$

Divide by X axis

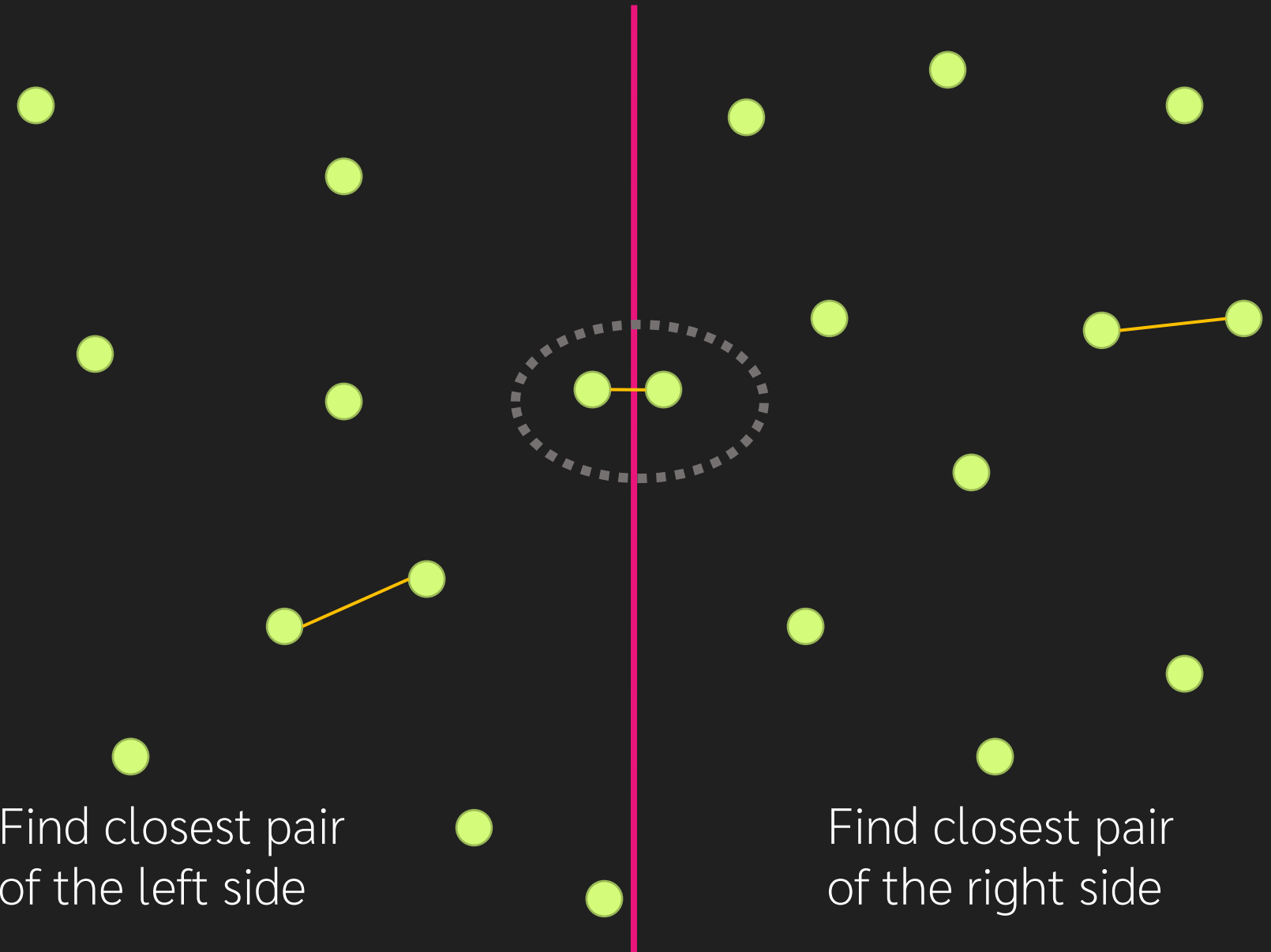


Conquer

แทนที่จะเช็กทุกคู่ ⇒ เช็กแค่บางคู่!

- Similar to 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 $(n/2)^2$ such pairs, if we simply consider everything, it would be $O(n^2)$, still quadratic running time
  - To get better complexity, we something better than $O(n^2)$ for the real work

Divide by X axis



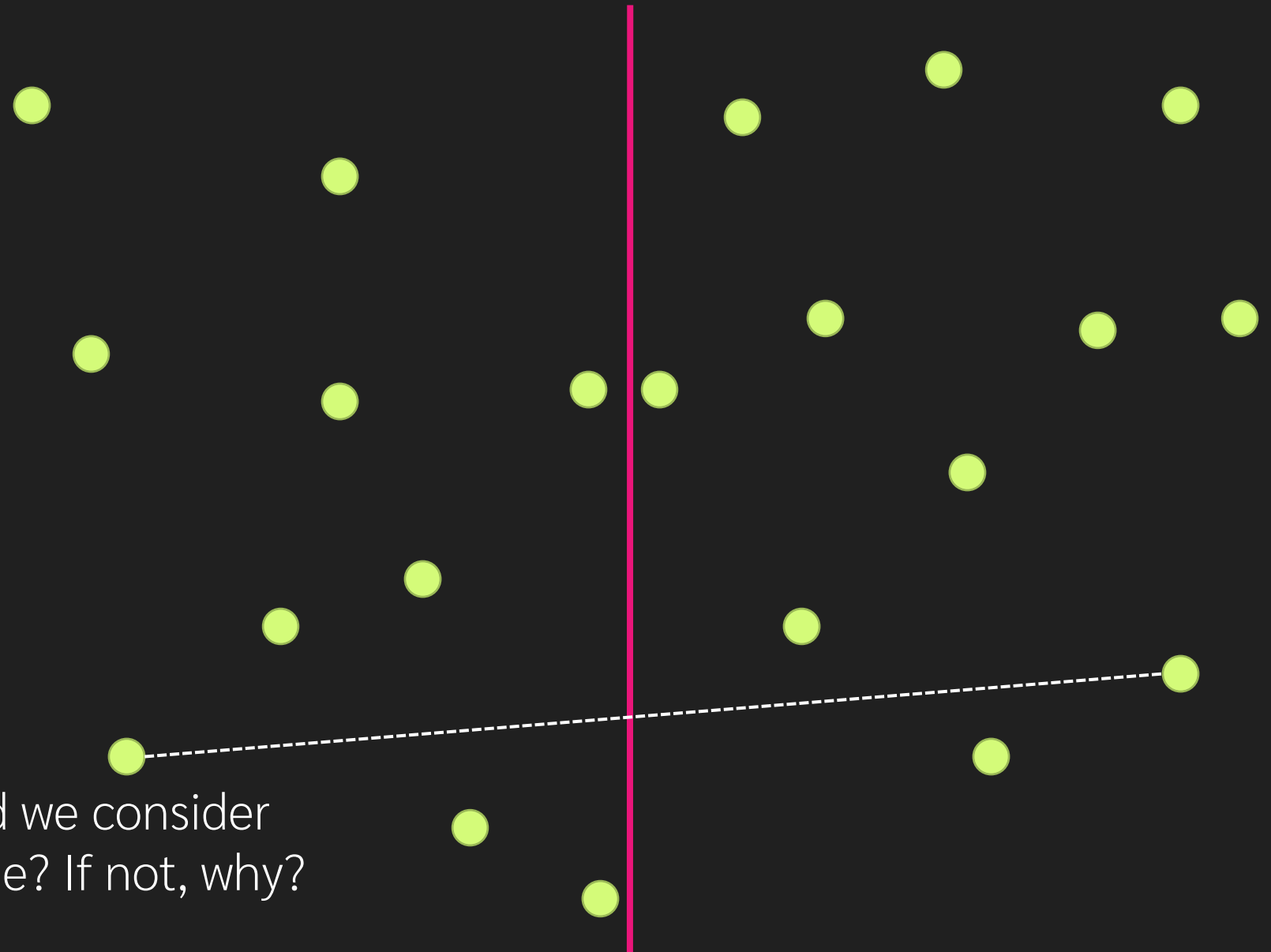
Find closest pair
of the left side

Find closest pair
of the right side

Find Closest Spanning Pair

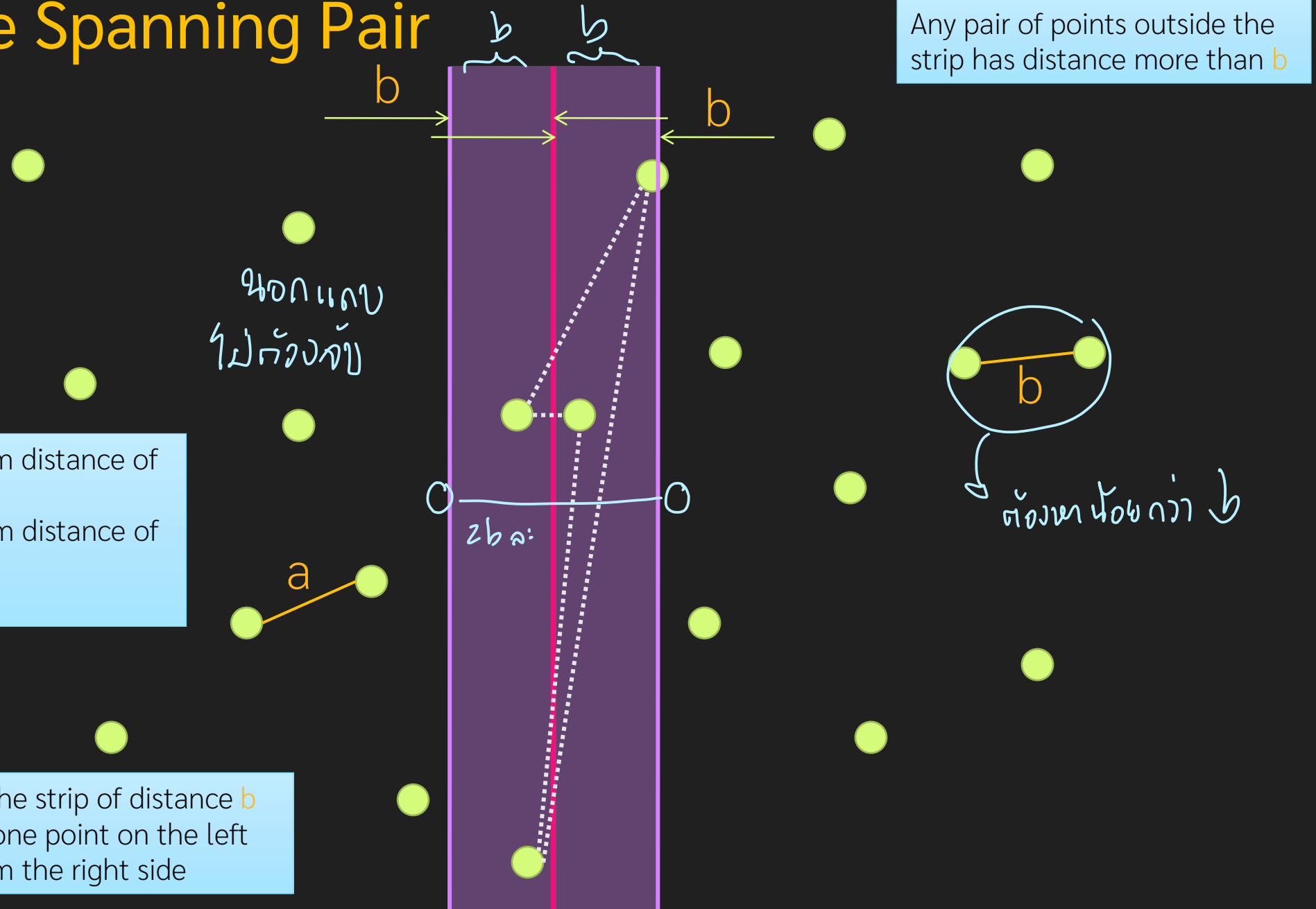
- Should consider only the nearer pairs
  - Skip pairs whose distance is definitely larger than b
  - Should not consider the pair on the far left with that on the far right

Find Closest Spanning Pair



Should we consider this one? If not, why?

Possible Spanning Pair

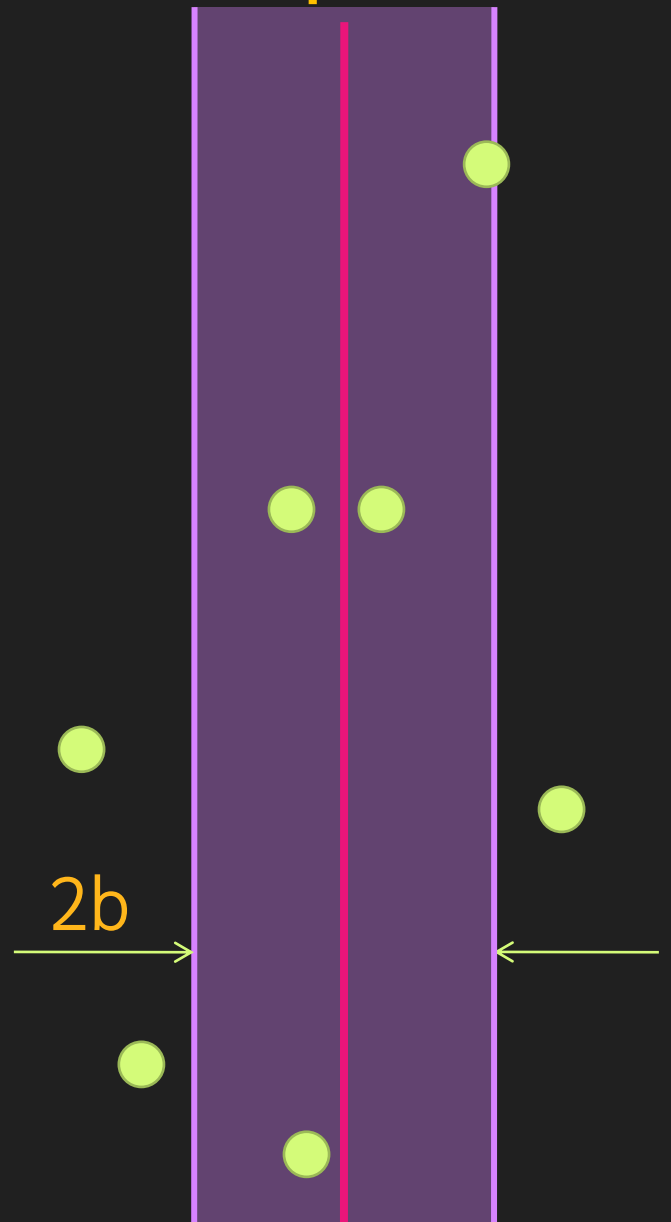


- Let a be the minimum distance of the left subproblem
- Let b be the minimum distance of the right subproblem
- Assume b < a

Any pair of points outside the strip has distance more than b

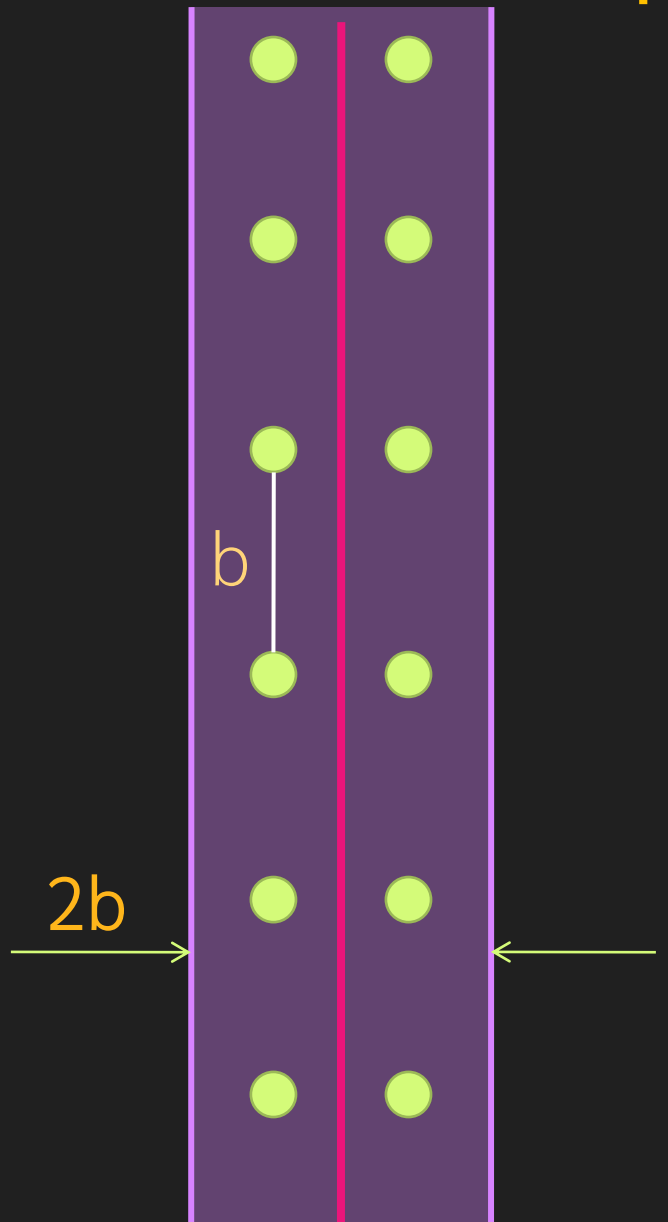
Consider only pairs in the strip of distance b from the division line, one point on the left side, another point from the right side

Point in Strips



- How many pairs in the strip?
  - Can it be more than **n**?

Pairs of Point in Strips can be is $O(N^2)$

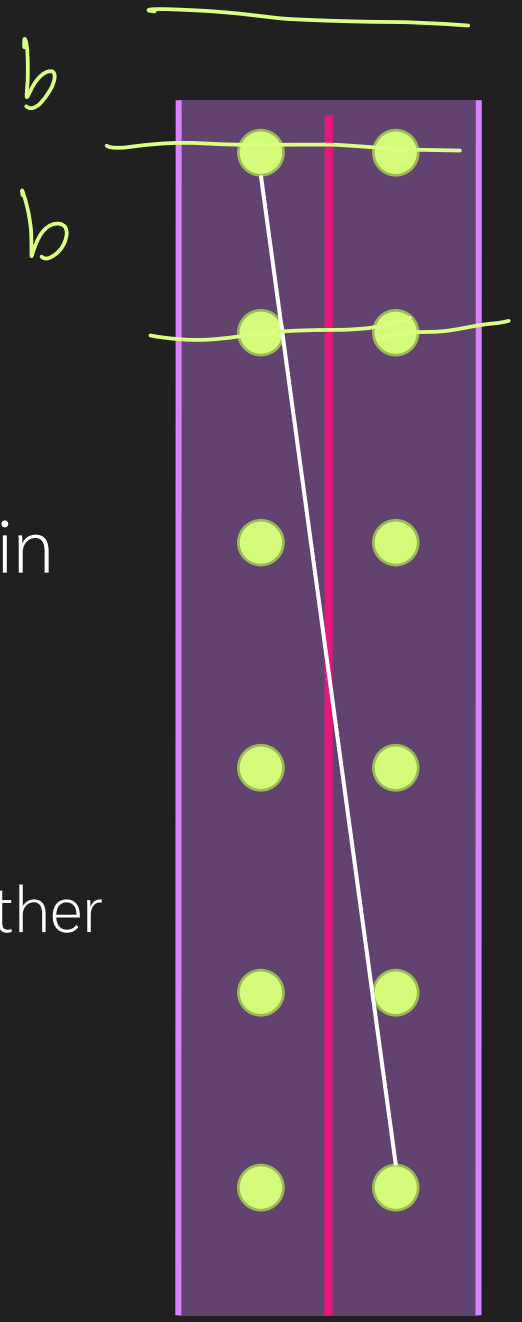


- Bad news
  - There can be as much as n/2 points on each side
  - Consider a set of vertically aligned point
    - Each are b unit apart
    - So, all points will be in the strip of 2b width
- The problem
  - If we check every pair of points, we still stuck with $O(n^2)$ time

ใช้เรื่องนี้ y ต่อ ได้มั้ย

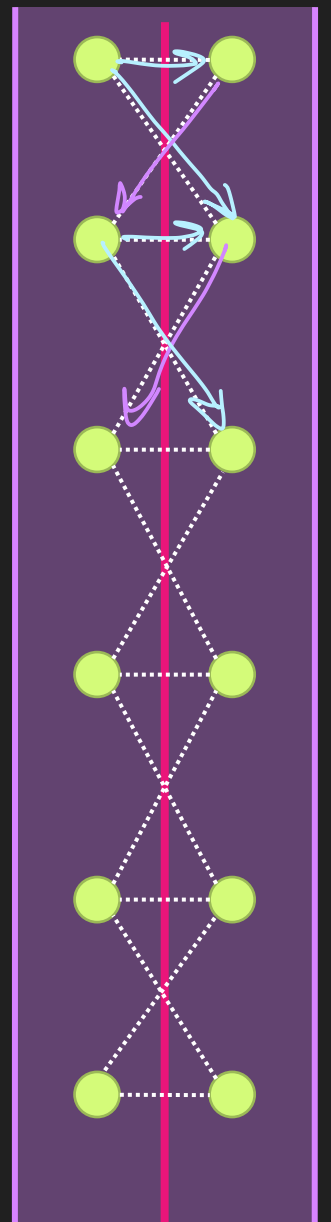
The Solution

- Think Vertically
- Do we have to check for every pair in the strip?
  - 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

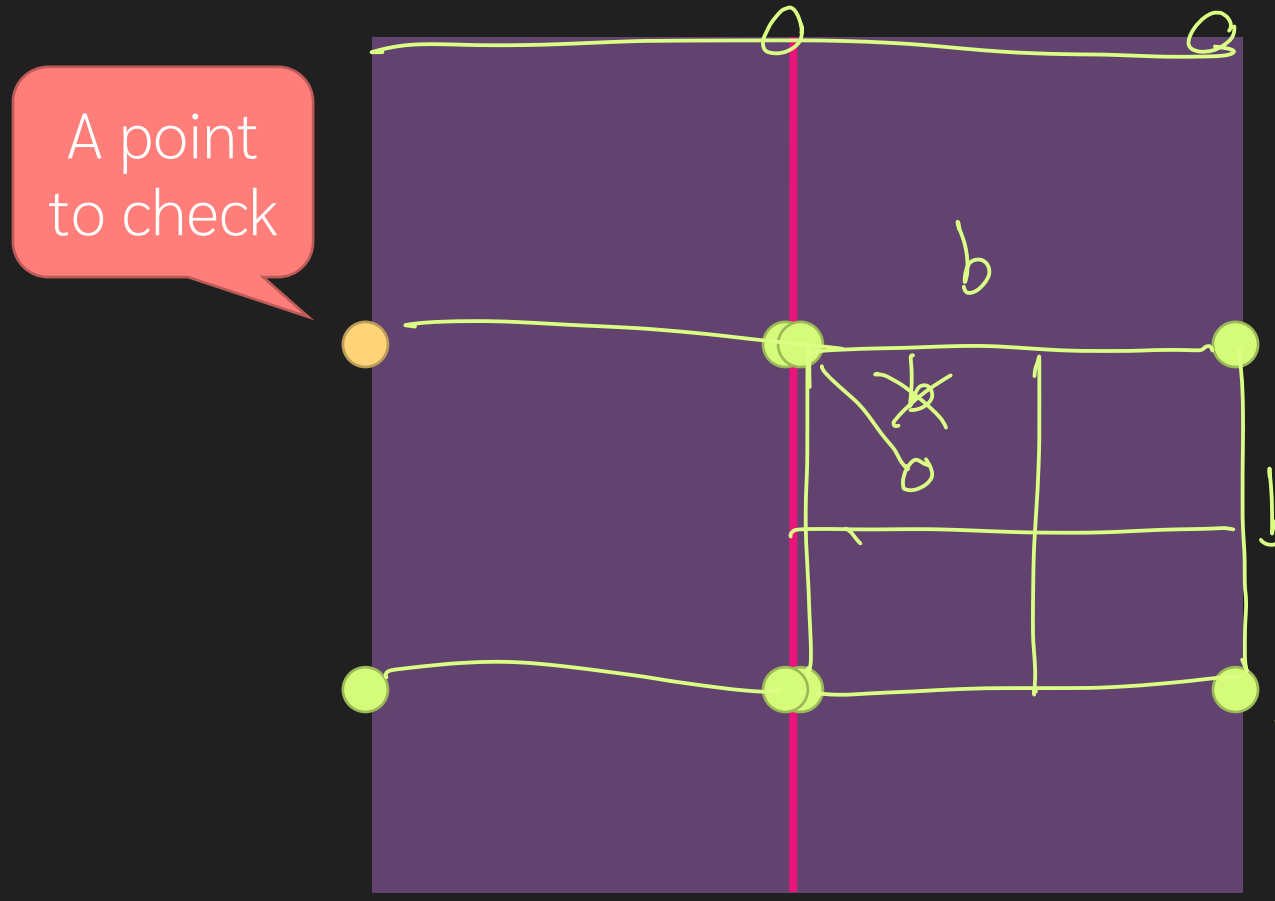


Question is still remains

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

$O(n \lg n)$

- How many pair to be checked?



There are, at most, 4 points for each starting point

$P_{x\,sort}$ แบ่งเป็น $P_{x\,sort\,L}$, $P_{x\,sort\,R}$

$CP(P_{x\,sort}, P_{y\,sort})$

$P_{y\,sort,L}$ $P_{y\,sort,R}$

จุดอื่นต้อง $\geq b$

Implementation Detail

เราต้องเรียงข้อมูลตามแกน y ก่อน

- For each point on one side, if we have to loops over every points on the other side to test whether the Y-value falls within range, it will still take O(n)
  - However, if the points are sorted by Y-value, we can scan from top to bottom and stop when difference of Y-value exceeds b

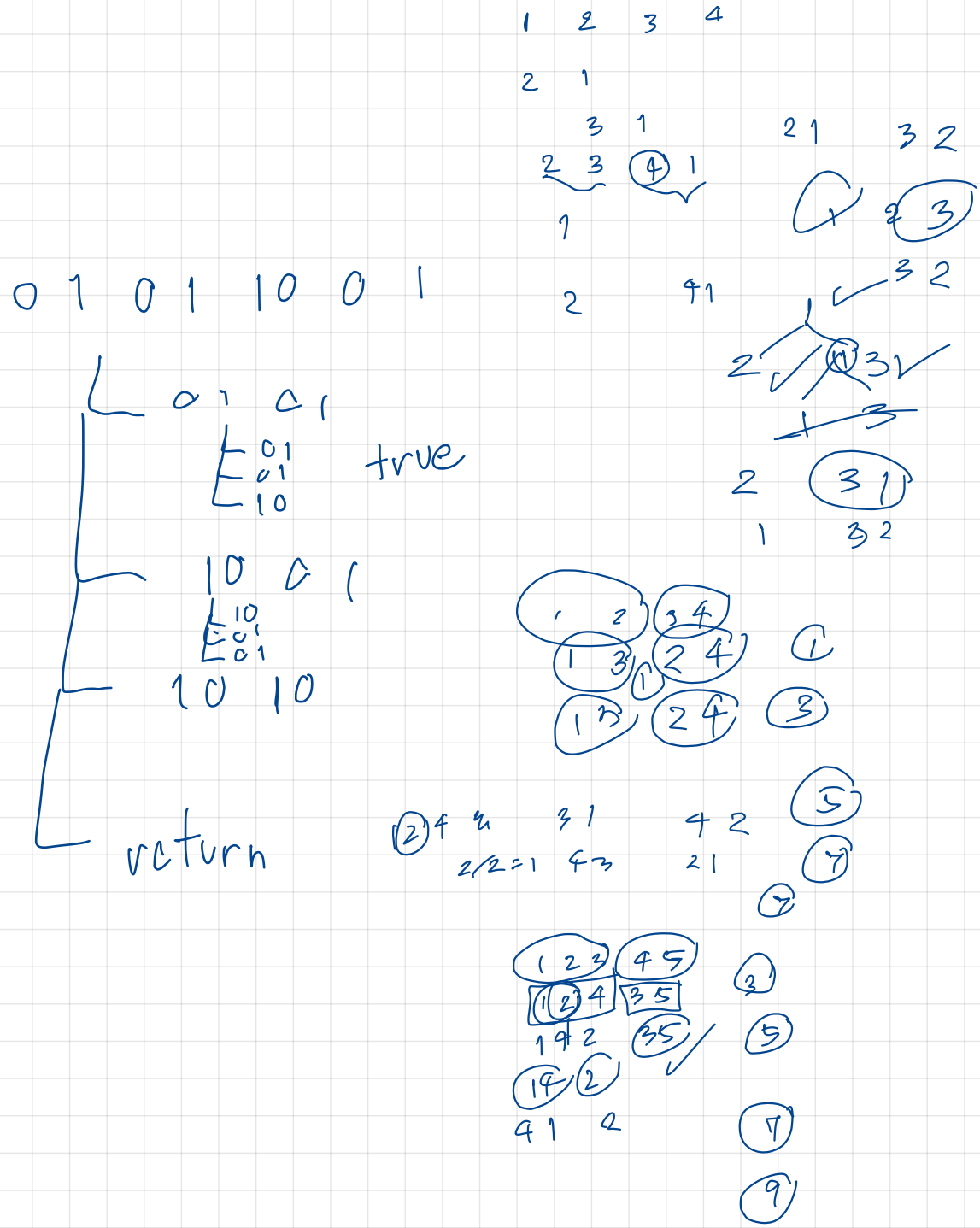
```
Let L[1..p] be points on the left strip, sorted by y-value
Let R[1..q] be points on the right strip, sorted by y-value
Let b be the width of the strip

result = (L[1],R[1])
min = distance(L[1],R[1])
right_idx = 1
for left_idx = 1 to p
  while right_idx < q && L[left_idx].y+b < R[right_idx].y
    d = distance(L[left_idx],R[left_idx])
    if d < min
      result = (left_idx,right_idx)
    right_idx  = right_idx + 1
#dit again, starting with right side
```

If we sort every time we do recursive call,
Real work will be O(n lg n)

That would result in $O(n \lg^2 n)$

Better Approach

- Point must be sorted in x-value so that dividing can be done in $O(1)$
- Point must also be sorted in y-value, so that we can stop checking points in the strip
- Sorting points only at the beginning
  - 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 $O(n)$


true
return