# Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281

ผู้สอน: ตอน 1 ผศ. เบญจมาศ ปัญญางาม
ตอน 2 ผศ. ดร. จักริน ชวชาติ

บทที่ 5

อัลกอริทึมแบ่งแยกและเอาชนะ

(Divide and Conquer algorithms Part1)





## Divide-and-conquer paradigm

- Divide-and-conquer.
  - Divide up problem into several subproblems (of the same kind).
  - Solve (conquer) each subproblem recursively.
  - Combine solutions to subproblems into overall solution.
- Most common usage.
  - Divide problem of size n into two subproblems of size n/2. O(n)time
  - Solve (conquer) two subproblems recursively.
  - Combine two solutions into overall solution. O(n)time
- Consequence.
  - Brute force: $\Theta(n^2)$.
  - Divide-and-conquer: O(n log n).





## Divide and conquer (D&C) Algorithm

- Divide and conquer Algorithm
  - Decrease and conquer
  - Merge sort
  - Closest pairs
  - Multiplication in sub-quadratic time
  - Quicksort expected running time analysis
  - Probabilistic selection
  - Deterministic selection





## Divide and Conquer (D&C) Algorithm

An important algorithm design paradigm.

It works by recursively

- Divide: Divide a given problem into a number of subproblems
- Conquer: Recursively solve each sub-problem
- Combine: Combine the solutions of the sub-problems into the solution of the original problem.

Solutions can also be implemented by a non-recursive algorithm that stores the partial sub-problems in some explicit data structure, such as a stack, queue, or priority queue.

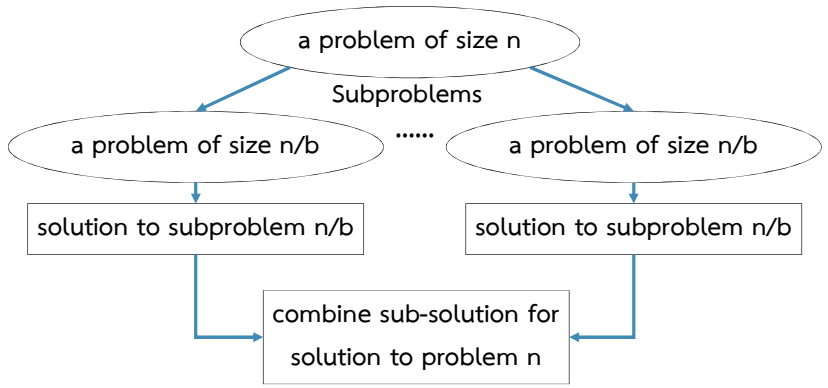
## Does Divide and Conquer Work Always?

- It's not possible to solve all the problems with Divide and Conquer techniques.
- As per the definition of D&C the recursion solves the subproblems which are of the same type.
- For all problems it is not possible to find the subproblems which are the same size.
- D&C is not a choice for all problems





## Divide and Conquer Visualization








## Divide and Conquer Visualization

```
DivideAndConquer(P){
    if(small(P)) //P is very small so that a solution obvious
        return solution(P);
    divide the problem P into k subproblems P1, P2, ... ,Pk :
    return (Combine(DivideAndConquer(P1),
        DivideAndConquer(P2),...,  DivideAndConquer(Pk)))
}
```





## Decrease and conquer

- One variation of divide and conquer
- A solution of problem depends on only one subproblem.
- Two advantages :
  - some problems do not need to solve all subproblems, and have a simpler conquer strategy. They can be generally solved with tail recursion.
  - Analysis of these problems is simpler than divide and conquer
- Example: Binary search tree

# Decrease and conquer

- Recursive binary search : Find an element in a sorted array
  - Divide : Check middle element
  - Conquer : Recursively search 1 subarray.
  - Combine: No Need

Decrease and conquer
"A solution of problem depends on only one subproblem"

```
BinarySearchRecursive(A[left..right])
1.  if (left > right) return(-1)
2.  m=(left + right)/2
3.  if x == A[m] return m;
4.  If x < A[m]
5.     return(BinarySearchRecursive (A[left..m-1]))
6.  else
7.     return(BinarySearchRecursive (A[m...right]))
```

Time complexity analysis: $T(n) = 1T(n/2) + O(1)$

#subproblems | Subproblem size | Cost divide & combine





# Merge sort analysis

The idea of Merge sort :

- Divide: Divide the n-element sequence into 2 subsequences of n/2 elements each.
- Conquer : Recursively sort 2 subarrays.
- Combine : Merge the 2 sorted subarrays





# Mergesort analysis

Merge_Sort(A,p,r)

1. If p < r then
2. q= $\lfloor (p+r)/2 \rfloor$
3. Merge_Sort(A,p,q)
4. Merge_Sort(A,q+1,r)
5. Merge(A,p,q,r)

```
Merge(A,p,q,r)
1. i = p, j = q + 1, n = r - p + 1
2. for k = 1 to n
3.    if ((A[i]< A[j]) or (j > r)) and (i≤q)
4.       B[k] = A[i]
5.          i = i +1
6.    else
7.          B[k] = A[j]
8.          j = j +1
9. for k = 0 to n – 1
10.   A[p + k] = B[k]
```

An algorithm contains a recursive call

Running time : Described by a recurrence.

$$T(n) = \begin{cases} 1 & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + n & if\ n > 1 \end{cases}$$

$t_B$ $t_A$





# Mergesort analysis

- Time complexity:

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

#subproblems | Subproblem size | Cost divide & combine

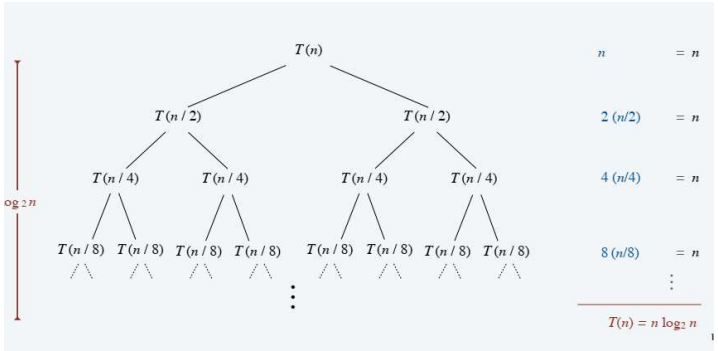
Solving recurrence relations Using the master method

a=2, b=2 $n^{\log_b a} = n^{\log_2 2} = n$

Is f(n)=n=O($n^{\log_b a-\epsilon}$) = O($n^{1-\varepsilon}$) , $\varepsilon$ >0 ? -> No,

Is f(n) = n = $\Theta(n^{\log_b a})$= $\Theta$(n) -> Yes ->Case 2 is selected.

Thus, T(n)= $\Theta(n^{\log_b a} \log n) = \Theta(n \log n)$








## Mergesort analysis

Time Analysis : T(n)= $\Theta$( n log n)

- Use Log(n) steps for merging
- Each merging step has complexity O(n)
  - If the array is already sorted, then only n – 1 comparisons are needed (it still needs time (n log n) because of the swapping, and it stills needs space 2n)

Memory :

- Needs extra memory ! ( ≈ 2n memory cells)

Stability :

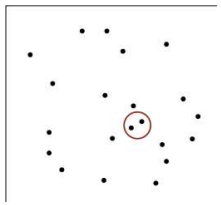
- For sorting algorithm, stability is the property that the order of equal elements is not changed
- Like insertion sort, merge sort is a stable sort
- Unlike Quicksort





## Closest pair of points

- Closest pair problem. Given n points in the plane, find a pair of points the with the smallest Euclidean distance between them.
- Fundamental geometric primitive.
  - Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
  - Special case of nearest neighbor, Euclidean MST, Voronoi. (fast closest pair inspired fast algorithms for these problems)







## Closest pair of points: Brute force

Brute force->Check all pairs.

BruteForceClosestPoints(P)  // P is list of points

1. dmin = ∞
2. for i= 1 to n-1 do
3. for k = i +1 to n do
4. d = sqrt ($(x_i - x_k)^2 + (y_i - y_k)^2$)
5. if d < dmin then
6. dmin = d, pos1 = i, pos2 = k
7. return pos1, pos2

Running time $\theta(n^2)$.

# Closest pair of points: Brute force

Pros and Cons of Brute Force

- Strengths:
  - Simplicity and Wide applicability
  - Yields reasonable algorithms for some important problems and standard algorithms for simple computational tasks
- Weaknesses:
  - Rarely produces efficient algorithms
  - Some brute force algorithms are in feasibly slow
  - Not as creative as some other design techniques





# Closest pair of points: D&C method

Divide-and-conquer method:

- Want to be lower than $O(n^2)$,
  - expect O(n log n).
- Need T(n)=2T(n/2)+O(n).
- How?
  - Divide : into 2 subsets (according to x-coordinate)
  - Conquer: recursively on each half.
  - Combine: select closer pair of the above.

One point from the left half and the other from the right may have closer distance.

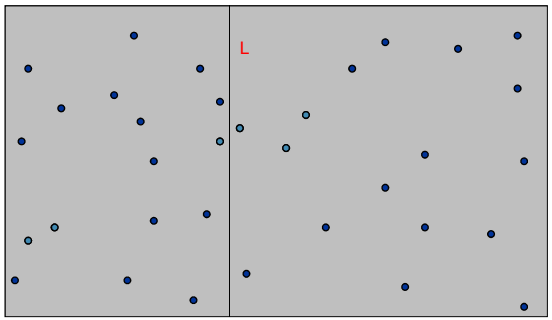




# Closest pair of points: D&C method

Algorithm.

- Divide: draw vertical line L so that roughly ½n points on each side.



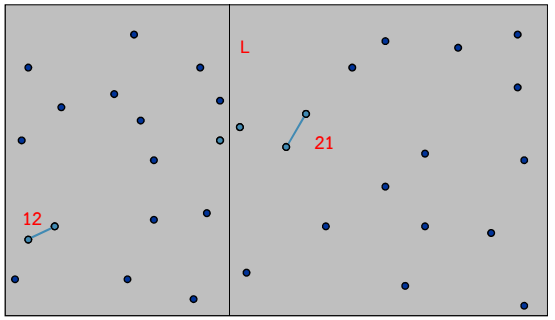





# Closest pair of points: D&C method

Algorithm.

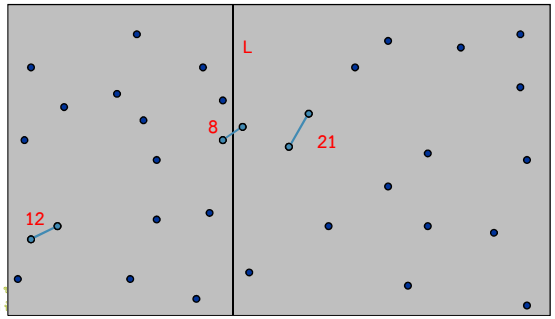
- Divide: draw vertical line L so that roughly ½n points on each side.
- Conquer: find closest pair in each side recursively.



# Closest pair of points: D&C method

Algorithm.

- Divide: draw vertical line L so that roughly ½n points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.

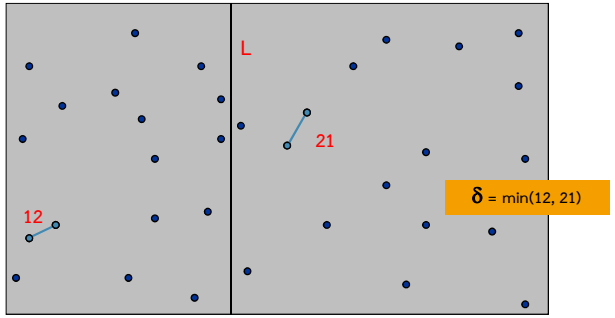







# How to find closest pair with one point in each side?

- Find closest pair with one point in each side, assuming that distance < $\delta$.

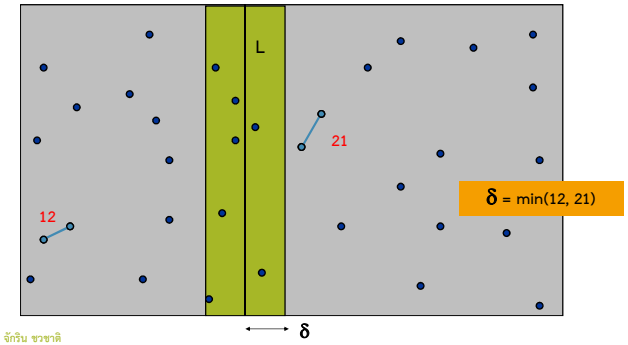







# How to find closest pair with one point in each side?

- Find closest pair with one point in each side, assuming that distance < $\delta$.
  - Observation: only need to consider points within $\delta$ of line L

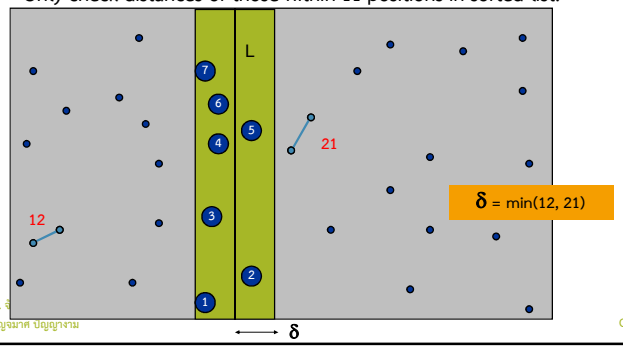







# How to find closest pair with one point in each side?

- Find closest pair with one point in each side, assuming that distance < $\delta$.
  - Observation: only need to consider points within $\delta$ of line L
  - Sort points in 2 $\delta$ -strip by their y coordinate.
  - Only check distances of those within 11 positions in sorted list!



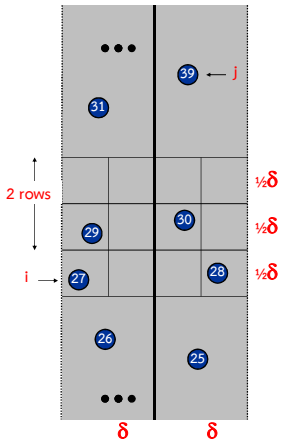
## How to find closest pair with one point in each side?

Def. Let $s_i$ be the point in the $2\delta$ -strip, with the $i^{th}$ smallest y-coordinate.

Claim. If $|i - j| \geq 12$, then the distance between $s_i$ and $s_j$ is at least $\delta$ .

Proof.

- No two points lie in same ½$\delta$-by-½$\delta$ box.
- Two points at least 2 rows apart have distance $\geq$ 2(½ $\delta$). ▪




- Fact. Still true if we replace 12 with 7.





Closest-Pair(p1, ..., pn) {

Compute separation line L such that half the points are on one side and half on the other side. O(nlogn)

$\delta_1$ = Closest-Pair(left half)

$\delta_2$ = Closest-Pair(right half) 2T(n/2)

$\delta$ = min($\delta$ 1, $\delta$ 2)

Delete all points further than $\delta$ from separation line L O(n)

Sort remaining points by y-coordinate. O(nlogn)

Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than $\delta$, update $\delta$. O(n)

return $\delta$.

}





## Closest Pair of Points: Analysis

- Running time.
- Q. Can we achieve O(n log n)?
- A. Yes. Don't sort points in strip from scratch each time.
  - Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
  - Sort by merging two pre-sorted lists.