เอกสารประกอบการอบรม

การแบ่งแยกเพื่อเอาชนะ
Divide and Conquer

ค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย 2 2/2567

ศูนย์โรงเรียนสามเสนวิทยาลัย - มหาวิทยาลัยธรรมศาสตร์

ระหว่างวันที่ 10 มีนาคม – 26 มีนาคม 2568

สาขาวิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี

มหาวิทยาลัยธรรมศาสตร์

Problem Solving Paradigms

คำถามชวนคิด

ให้ Implement โปรแกรม/คิดวิธีการของปัญหา

การหาจำนวนที่มากที่สุด/น้อยที่สุดใน A เมื่อเรากำหนด A ={10, 7, 3, 5, 8, 2, 9}

- **Complete search (Brute-force)**:
  - ถ้าเราใช้วิธีการวนลูปเช็คทุกตัวใน List จะใช้เวลา => O(n)
- **Divide and Conquer: (Today's topic !)**
  - จาก T1 หากเราแก้ปัญหาด้วย Divide and Conquer จะใช้เวลา => O(n log n)

Problem Solving Paradigms

- ลองนึกภาพว่าเราแก้ปัญหาต่อไปนี้ด้วย Brute force เมื่อเรากำหนด Array A มีสมาชิก n <= 10K (สมมติ A={10, 7, 3, 5, 8, 2, 9}):
  - T1: การหาจำนวนที่มากที่สุด/น้อยที่สุดใน A (10, 2)
  - T2: การหาสมาชิกทีน้อยที่สุดลำดับที่ k ใน A (k=2; ans=3)
  - T3: หาผลต่างที่มากที่สุดของเลขสองจำนวนใน A (8)
  - T4: หา Longest increasing subsequence ใน A ({3,5,8,9})
- **Complete search (Brute-force)**:
  - T1 => ถ้าเราใช้วิธีการวนลูปเช็คทุกตัวใน List => O(n)
  - T2 => ถ้าเราหา Medium (k=n/2) เราอาจจะใช้เวลา => $O(n^2)$
  - T3 => ถ้าหา Minimum จากทุก ๆ คู่ที่เป็นไปได้ => $O(n^2)$
  - T4 => ถ้า Generate ทุก ๆ Subsequences ที่เป็นไปได้แล้วหา Len ที่ยาวที่สุดจะใช้เวลา $O(2^n)$
- **Divide and Conquer: (Today's topic !)**
  - จาก T1 และ T2 หากเราแก้ปัญหาด้วย Divide and Conquer จะใช้เวลา => O(nlogn) และ O(n) ตามลำดับ
- **Greedy:**
  - สำหรับ T3 หากเราหาได้ว่าผลต่างระหว่าง 2 ตัวเลขที่กว้างที่สุดใน A คือ ผลต่างระหว่าง min(A) – max(A) ดังนั้นเราจะใช้เวลา O(n) + O(n) จาก Task แรก
- **Dynamic Programming:**
  - T4 เราสามารถใช้เทคนิค DP ในการแก้ปัญหาใน $O(n^2)$ หรือใช้ Greedy ในเวลา O(nlogk)

Complete Search (Review)

- Complete search:
  - A.k.a. Brute force หรือ recursive backtracking ที่ลงไปค้นหาทั้ง **Search space**
  - ในระหว่างค้นหาเราอาจจะเลือกที่จะไม่สนใจบางส่วนของ Search space ได้ เรียกว่าการ **Pruning** ซึ่งไม่มีผลต่อความถูกต้องของผลลัพธ์
  - เมื่อพิจารณาดูแล้วไม่มี Algorithms ใดเลยที่ fit กับปัญหาที่กำลังจะแก้ ให้พิจารณา Complete search
  - ข้อดีคือ (1) **ง่าย** (2) **ไม่มีทางที่คำตอบจะผิดได้เลย** (เพราะค้นทุกทางที่เป็นไปได้)
    - แต่จากที่เห็นจากตัวอย่าง บางปัญหาจะมี Algorithm ที่ดีกว่า Complete search แต่ Complete search มักจะ**ใช้เวลานานเกินไป** (Time Limit Exceeded)
    - เราอาจจะใช้ Complete search ในการ Generate Combinatoric objects ของ n เล็ก ๆ เพื่อดู Pattern ของปัญหาเพื่อจะช่วยเราคิดว่า จะใช้ Algorithm ที่ดีกว่าอันไหนดี (ในกรณีที่นึกไม่ออก)
    - ใช้ในการ Verify คำตอบของ Algorithm ที่ซับซ้อน
    - Complete search อาจจะใช้แก้ปัญหาที่ยากได้ (เนื้อหาอยู่ใน Lecture อื่น)
  - ถ้าพิจารณาแล้วพบว่าขนาดของปัญหาไม่ใช่ Challenge หลักของปัญหาที่กำลังแก้ then.. Go ahead!

Divide and Conquer

Divide – Conquer - Combine

```
DQ ( P) {
  if ( P is trivial ) return Solve ( P )
  Divide P into P_1, P_2, … , P_k
  for ( i =1to k )
    S_i = DQ ( P_i )
  S = Combine ( S_1, S_2, …, S_k)
  return S
}
```

Divide and Conquer

- **Divide-and-conquer** is probably the best-known general algorithm design technique.
- Divide-and-conquer algorithms work according to the following general plan:

DIVIDE

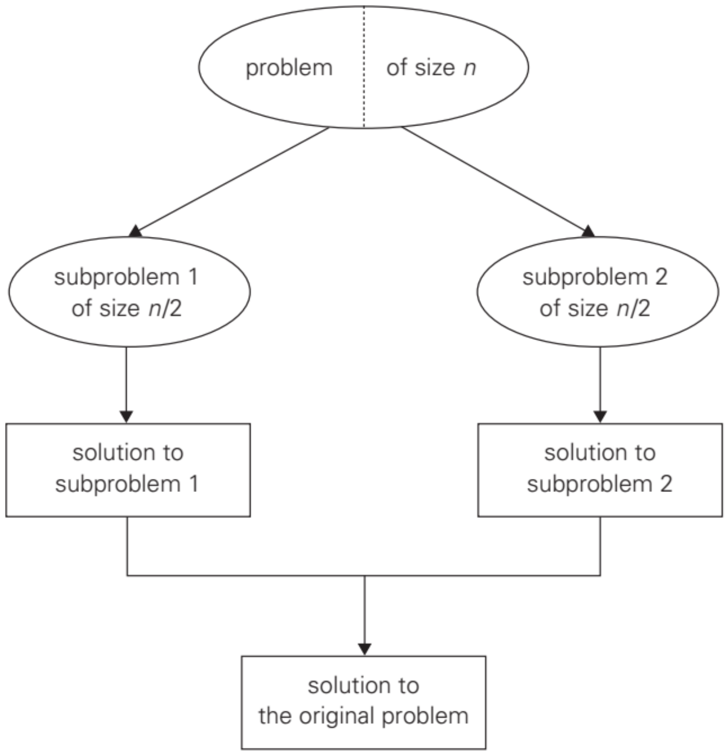
Divide the problem into a number of subproblems that are smaller instances of the same problem.

CONQUER

Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

COMBINE

Combine the solutions to the subproblems into the solution for the original problem.



Divide and Conquer

- Let us consider the problem of computing the sum of n numbers.

1	2	3	4	5	6	7	8	9

If n > 1, we can divide the problem into two instances: first $\lfloor n/2 \rfloor$ numbers and the remaining $\lceil n/2 \rceil$ numbers

1	2	3	4

5	6	7	8	9

1	2

3	4

5	6

7	8	9

… Do it recursively…

“bottom out”

Divide and Conquer

- In the most typical case of divide-and-conquer a problem's instance of size n is divided into two instances of size $n/2$.
- More generally, an instance of size n can be divided into b instances of size n/b, with a of them needing to be solved. ($constants\ a \ >= 1, b \ > \ 0$)
- Assume that n is a power of b, i.e. $n = b^p$
- Thus, the running time $T(n)$ can be computed as:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \ , \\ aT(n/b) + D(n) + C(n) & \text{otherwise .} \end{cases}$$

- If we take $D(n)$ time to divide the problem into subproblems and $C(n)$ time to combine the solutions to the subproblems into the solution to the original problem

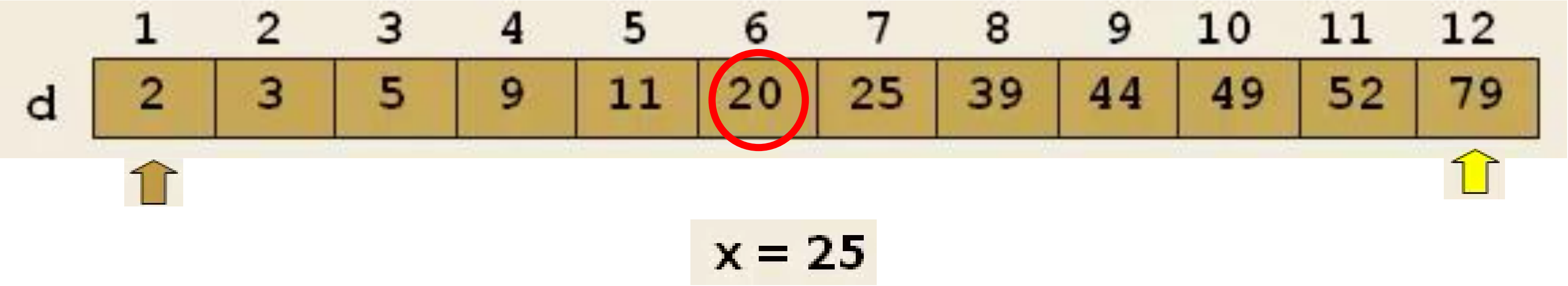
Binary Search

- input : x และ $D =< d_1, d_2, d_3, \dots, d_n >$

  $d_1 \le d_2 \le d_3 \le \cdots \le d_n$

  x และ d_i เป็นจำนวนจริง

- output : ค่า k ที่ $d_k = x$ ถ้าหาไม่พบ คืน -1



Binary Search

```
bsearch( d[1…n], x, left, right ) {
    if ( left > right ) return -1
    mid = ⌊(left + right)/2⌋
    if ( x == d[mid] ) return mid
    if ( x < d[mid])
        return bsearch( d, x, left, mid – 1)
    else
        return bsearch( d, x, mid + 1, right)
}
```

ลองพิจารณาหา t(m) ที่เป็นเวลาการทำงานในการเรียก bsearch
โดยสมมติให้ข้อมูลมี m จำนวน

คำถามชวนคิด

ให้ Implement โปรแกรม/คิดวิธีการของปัญหา

สมมติให้ A เป็น array เก็บข้อมูลแตกต่างกันที่เรียงลำดับแล้ว

อยากทราบว่า ช่องใดใน A ที่ A[k] มีค่าเป็น k

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
-4	-3	-1	0	1	2	6	8	11	13	14	19	20	24	26	35	48	49	80	90	92

คำถามชวนคิด

ให้ Implement โปรแกรม/คิดวิธีการของปัญหา

สมมติให้ A เป็น array n+1 ช่อง

A เก็บจำนวนเต็มมีค่า 1 ถึง n (แสดงว่ามีค่าที่ซ้ำ)

จงหาว่า ค่าใดซ้ำใน A

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	10	19	15	14	9	13	12	11	8	7	3	4	6	16	17	20	2	5	18	16

Integer multiplication

A x B :

$$
\begin{array}{r}
\times \quad 1003 \\
0410 \\
\hline
0000 \\
1003 \\
4012 \\
0000 \\
\hline
0411230 \\
\hline
\end{array}
$$

$$1003 = 10\times10^2 + 03 \ , \ 0410 = 04\times10^2 + 10$$

$$(10\times10^2 + 03)\times(04\times10^2 + 10)$$

$$
\begin{aligned}
&= 10\times04\times10^4 + (10\times10 + 03\times04)\times10^2 + 03\times10 \\
&= 400000 \quad + \quad 11200 \quad + 30 \\
&= 411230
\end{aligned}
$$

Integer multiplication

A x B :

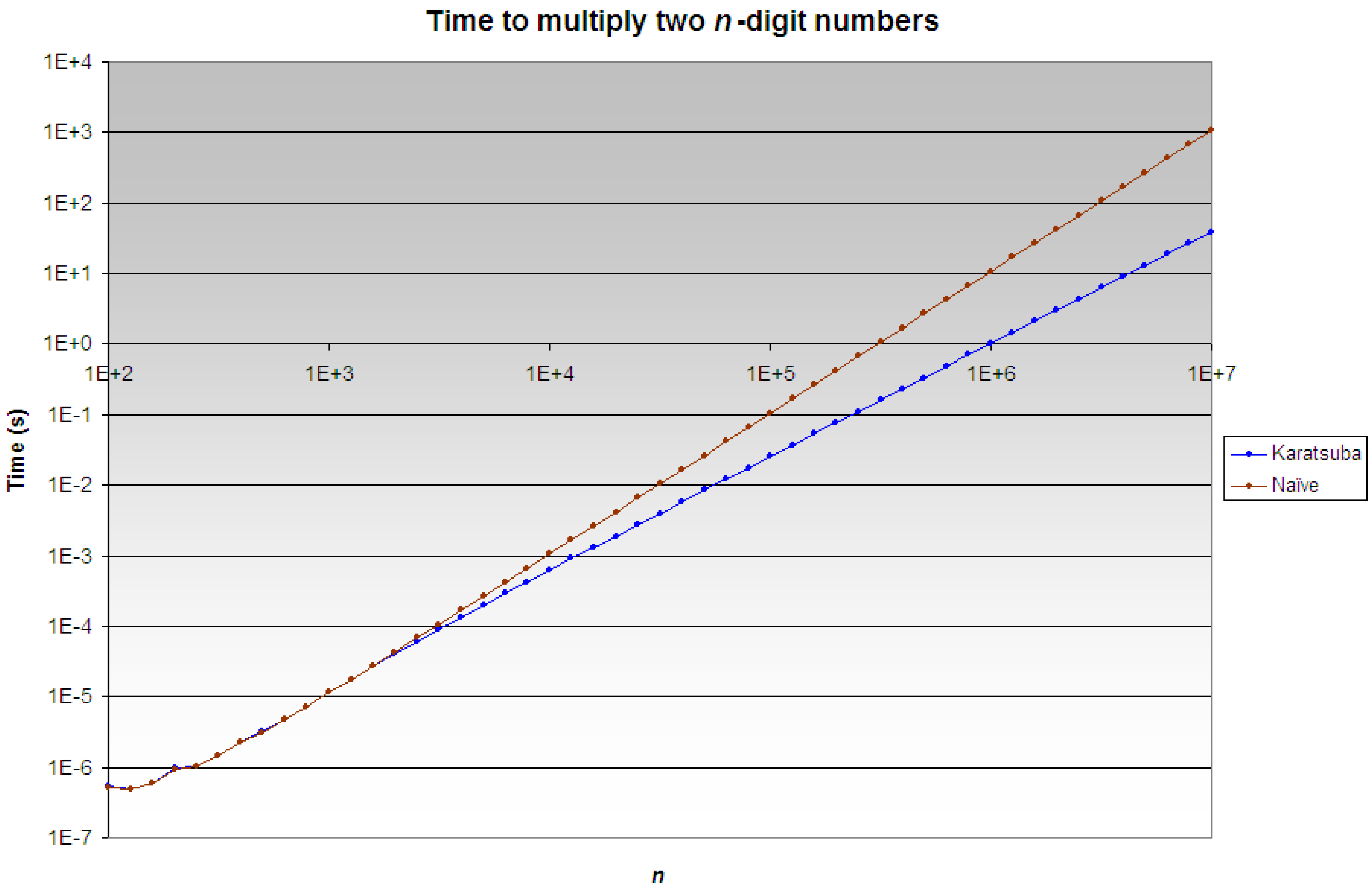
	$n/2$ digits	$n/2$ digits
$A =$	A_L	A_R
$B =$	B_L	B_R

$$A \times B = (A_L 10^{n/2} + A_R) \times (B_L 10^{n/2} + B_R)$$
$$= A_L \times B_L 10^{n} + (A_L \times B_R + A_R \times B_L) 10^{n/2} + A_R \times B_R$$

$t(n) = 4t(n/2) + \Theta(n)$
Master method : $c = \log_2 4 = 2$,
$n = O(n^{2-\varepsilon})$ ❶ $\rightarrow t(n) = \Theta(n^2)$

☹

Integer multiplication



https://www.nayuki.io/res/karatsuba-multiplication/benchmark.png

Matrix multiplication

```
MatrixMult( A[1..n][1..n], B[1..n][1..n] ){
  C = new array[1..n][1..n]
  for ( i = 1; i <= n; i++ ) {
    for(j= 1; j<= n; j++){
      C[i][j] = 0
        for ( k = 1; k <= n; k++ ) {
          C[i][j] += A[i][k] * B[k][j]
        }
    }
  }
return C
}
```

Matrix multiplication: Divide & Conquer

$$A = \left[\begin{array}{ccc|cc} a_{1,1} & \cdots & a_{1,n/2} & \cdots & a_{1,n} \\ \vdots & \boxed{A_{1,1}} & \vdots & \boxed{A_{1,2}} & \vdots \\ a_{n/2,1} & \cdots & a_{n/2,n/2} & \cdots & a_{n/2,n} \\ \hline \vdots & \boxed{A_{2,1}} & \vdots & \boxed{A_{2,2}} & \vdots \\ a_{n,1} & \cdots & a_{n,n/2} & \cdots & a_{n,n} \end{array}\right]$$

$$\left[\begin{array}{c:c} A_{11} & A_{12} \\ \hdashline A_{21} & A_{22} \end{array}\right]$$

Matrix multiplication: Divide & Conquer

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

$$\begin{aligned}
C_{1,1} &= A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} \\
C_{2,1} &= A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} \\
C_{1,2} &= A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\
C_{2,2} &= A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2}
\end{aligned}$$

$t(n) = 8t(n/2) + \Theta(n^2)$

Master method : $c = \log_2 8 = 3$

$n^2 = O(n^{3-\varepsilon})$ ❶ → $\Theta(n^3)$

☹

Matrix multiplication: Strassen (1969)

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

$$\begin{aligned}
M_1 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\
M_2 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
M_3 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \\
M_4 &= (A_{11} + A_{12}) \times B_{22} \\
M_5 &= A_{11} \times (B_{12} - B_{22}) \\
M_6 &= A_{22} \times (B_{21} - B_{11}) \\
M_7 &= (A_{21} + A_{22}) \times B_{11}
\end{aligned}$$

$$\begin{aligned}
C_{11} &= M_1 + M_2 - M_4 + M_6 \\
C_{12} &= M_4 + M_5 \\
C_{21} &= M_8 + M_7 \\
C_{22} &= M_2 - M_3 + M_5 - M_7
\end{aligned}$$

$t(n) = 7t(n/2) + \Theta(n^2)$ ☺

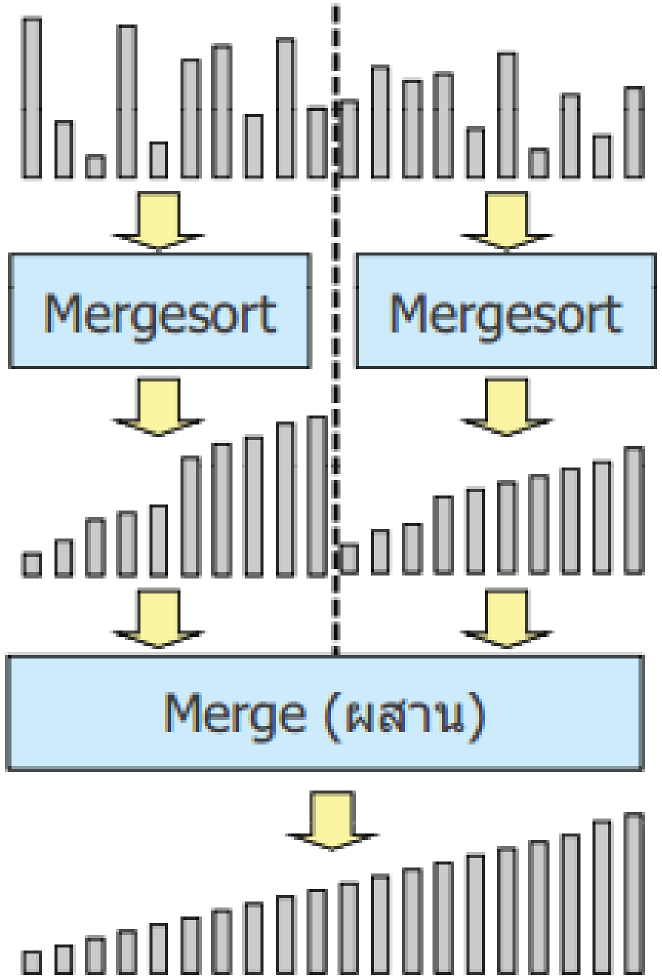
Master method : $c = \log_2 7$

$n^2 = O(n^{c-\varepsilon})$ $t(n) = \Theta(n^{\log_2 7})$

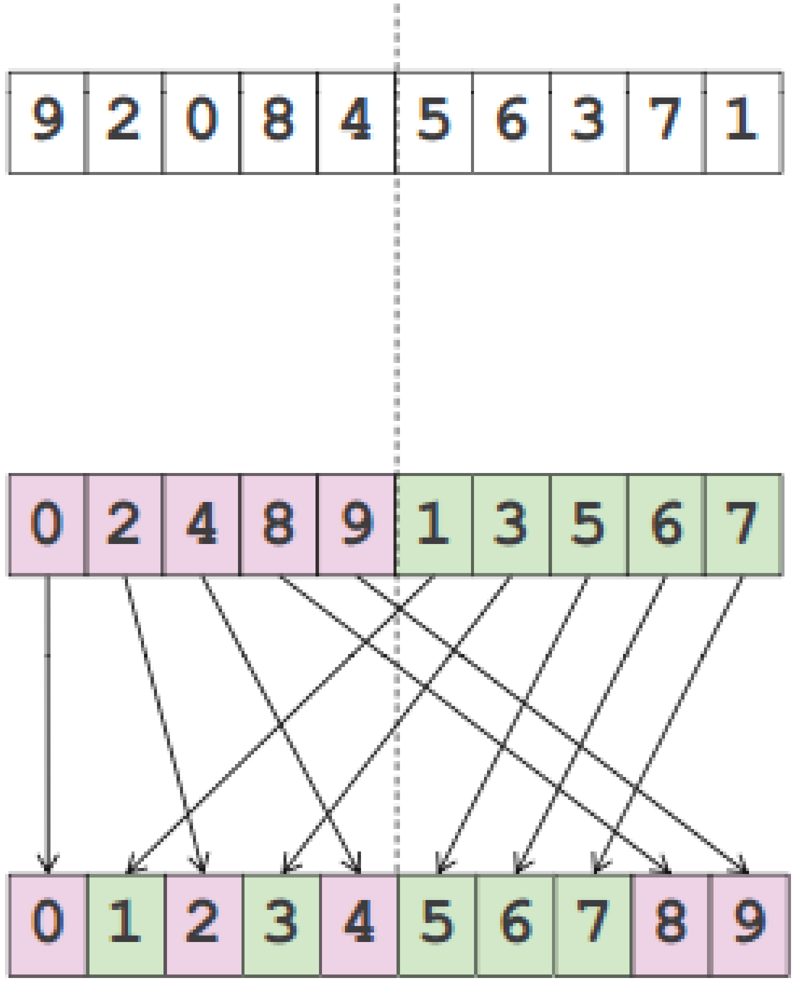
1969 : $O(n^{2.81})$ Strassen

1987 : $O(n^{2.376})$ Coppersmith–Winograd

Merge Sort



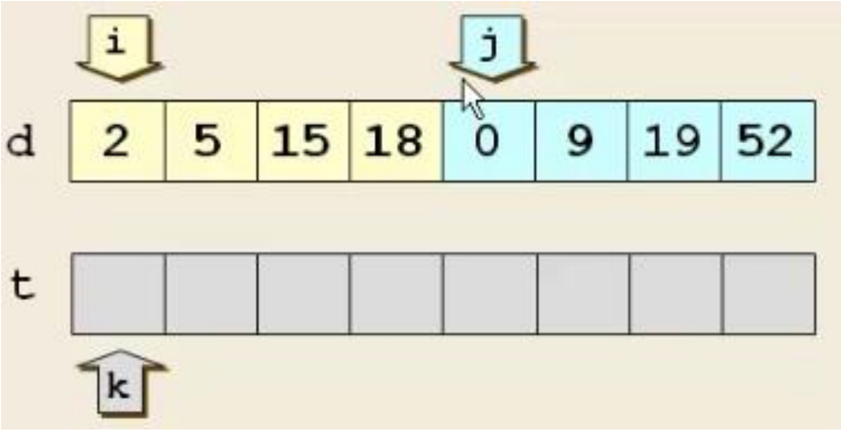
Mergesort
Mergesort
Merge (ผสาน)



9 2 0 8 4 5 6 3 7 1
0 2 4 8 9 1 3 5 6 7
0 1 2 3 4 5 6 7 8 9

Merge Sort

```
mergeSort( d[1..n], left, right ) {
   if (left >= right) return
   mid = ⌊(left + right) / 2⌋;
   mergeSort(d, left, mid);
   mergeSort(d, mid + 1, right);
   merge(d, left, mid, right);
}
```



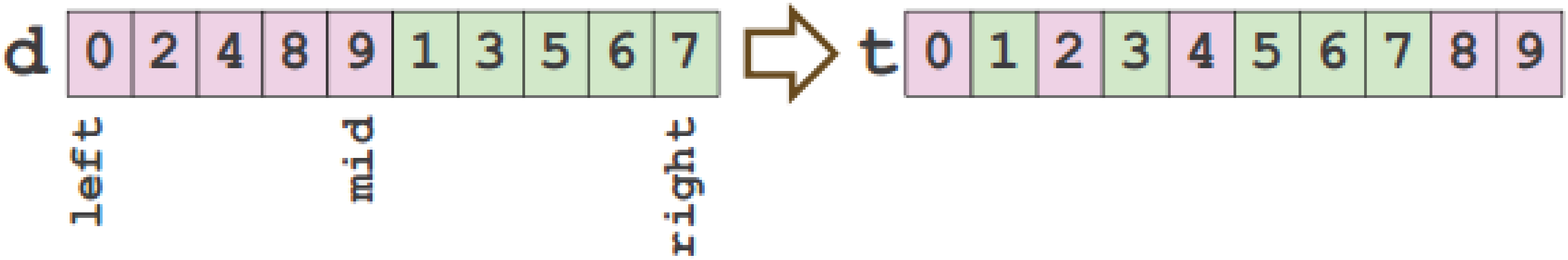
ให้ $t(n)$ คือเวลาในการ mergesort ข้อมูลจำนวน n ตัว

$$\boxed{t(n) = 2t(n/2) + (\text{เวลาในการ merge})}$$

Merge: #cmps

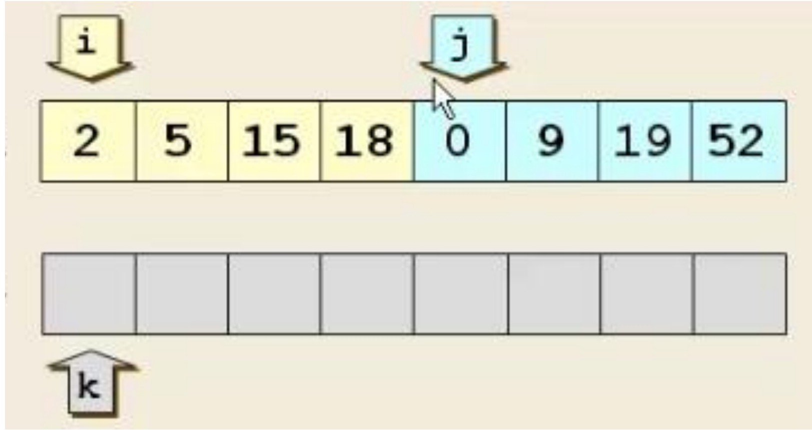
```
merge( d[1..n], left, mid, right ) {
   create t[left..right]
   i = left, j = mid+1;
   for (k = left; k <= right; k++) {
     if (i > mid)    {t[k] = d[j++]; continue}
     if (j > right) {t[k] = d[i++]; continue}
     t[k] = (d[i] < d[j]) ? d[i++] : d[j++]
   }
   for (k = left; k <= right; k++) d[k] = t[k]
}
```

$$n/2 \le \#cmps \le n-1$$



Merge Sort

```
mergeSort( d[1..n], left, right ) {
  if (left >= right) return
  mid = ⌊(left + right) / 2⌋;
  mergeSort(d, left, mid);
  mergeSort(d, mid + 1, right);
  merge(d, left, mid, right);
}
```



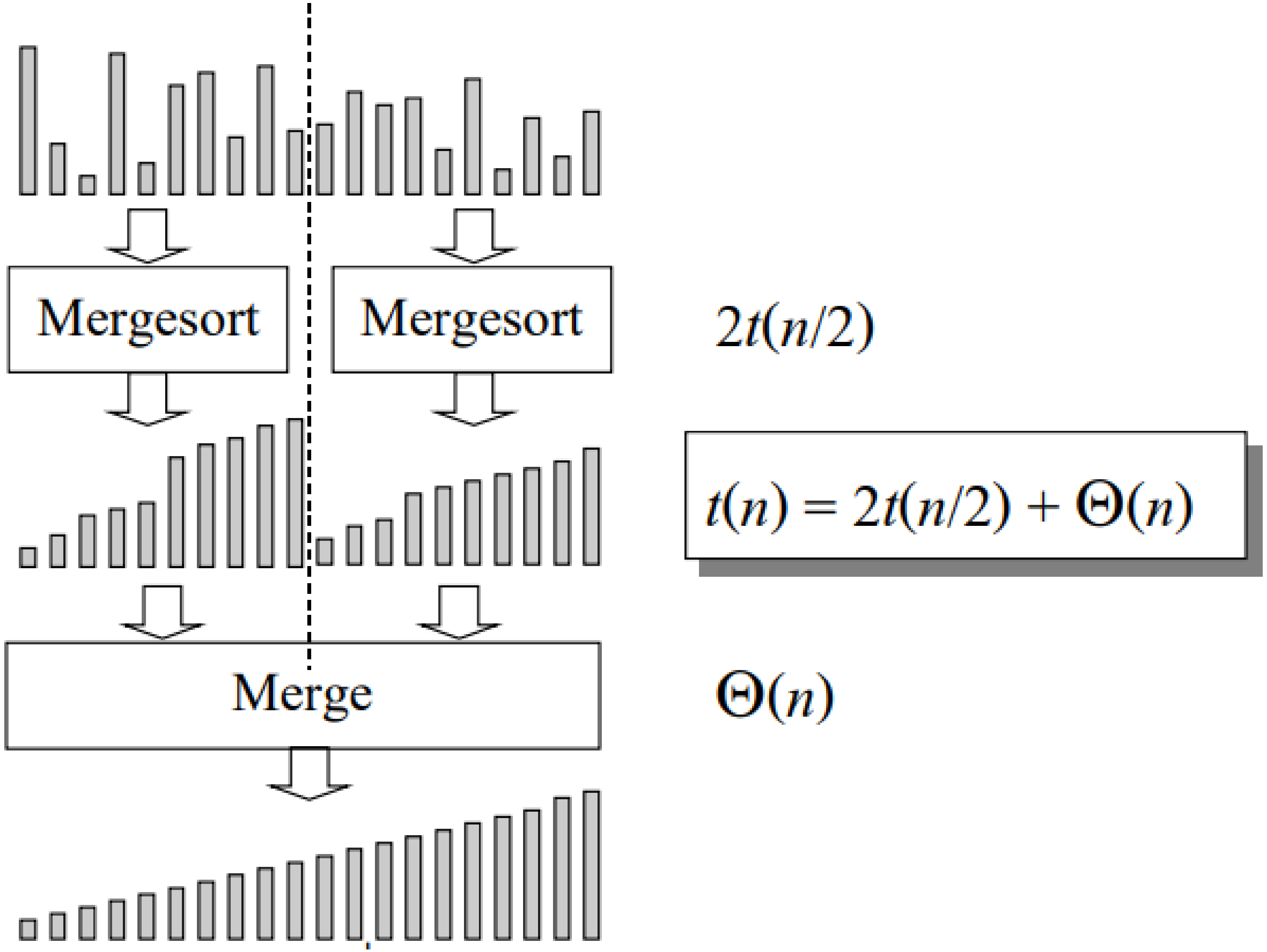
ให้ $t(n)$ คือ เวลา ในการ mergesort ข้อมูลจำนวน n ตัว

$$t(n) = \boxed{2t(n/2)} + \boxed{\Theta(n)}$$

master method : $n = \Theta(n^{\log_2 2})$

ได้ $t(n) = \Theta(n \log n)$

Merge Sort

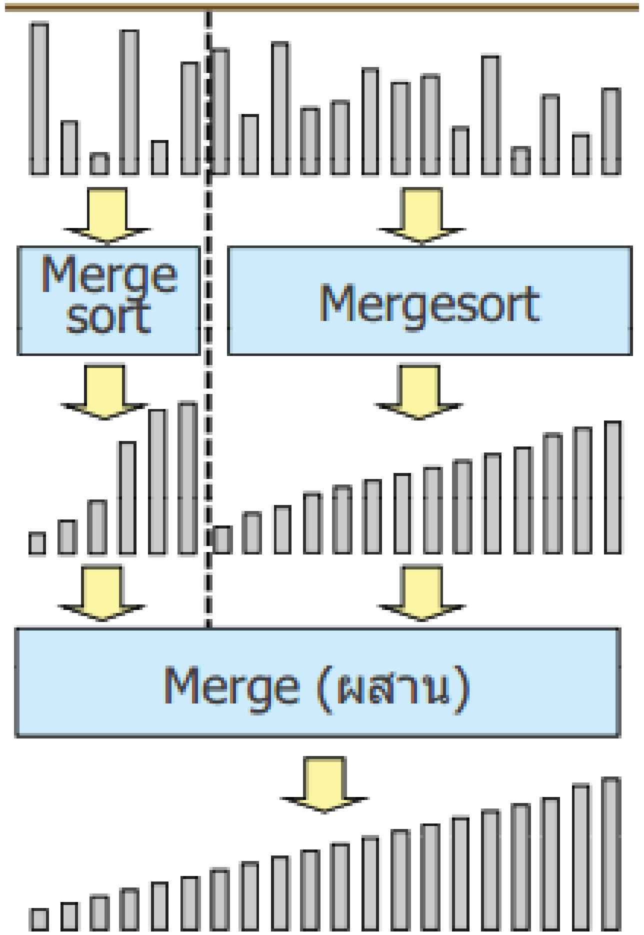


คำถามชวนคิด

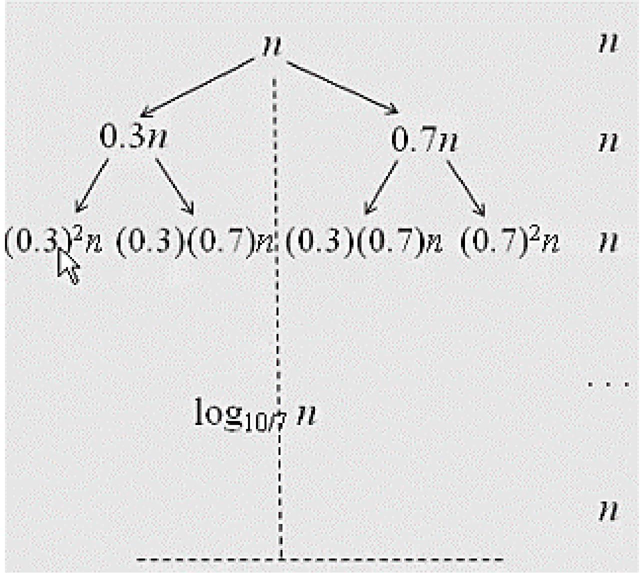
Merge sort: ถ้าแบ่ง 30-70

t(n) จะลดลงหรือเพิ่มขึ้นหรือเท่าเดิม?

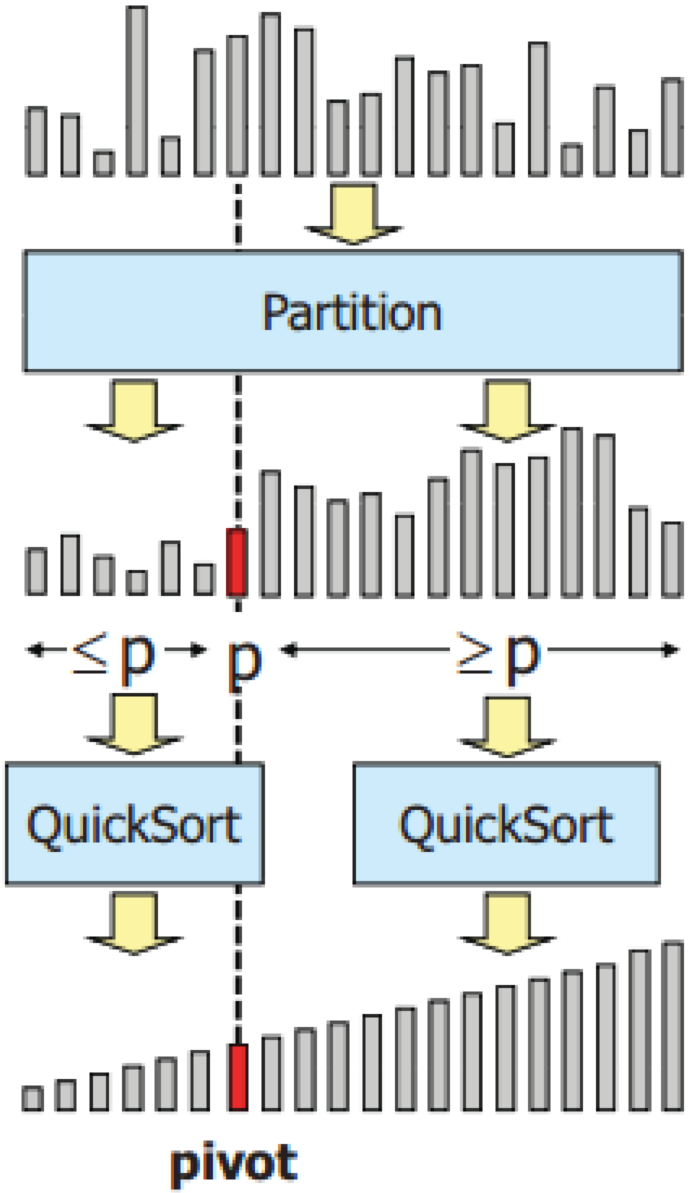
Merge Sort: ถ้าแบ่ง 30 - 70

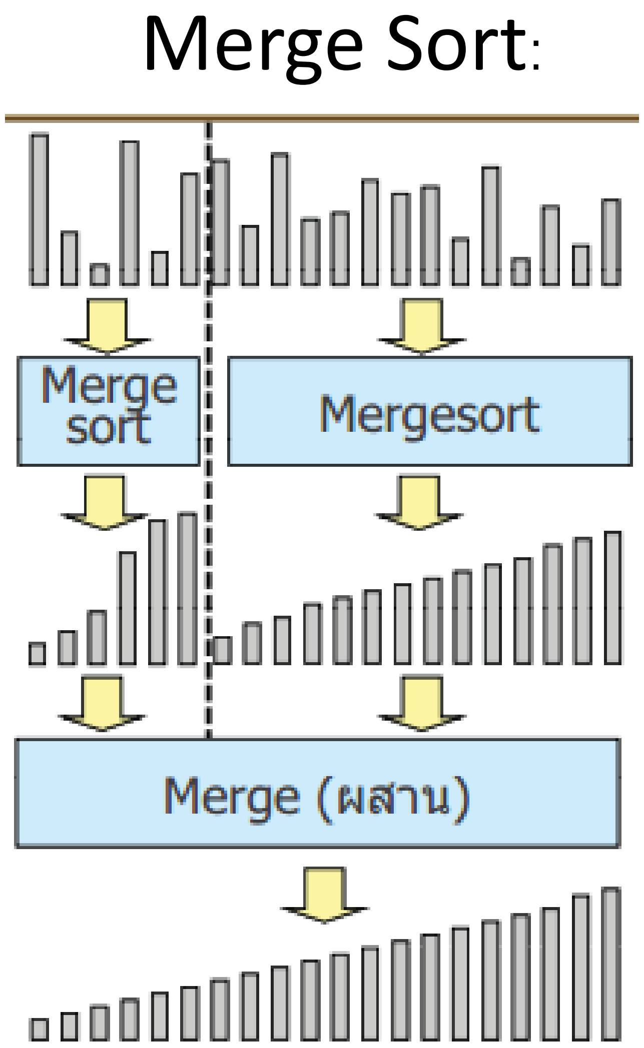


$$t(n) = t(0.3n) + t(0.7n) + \Theta(n)$$



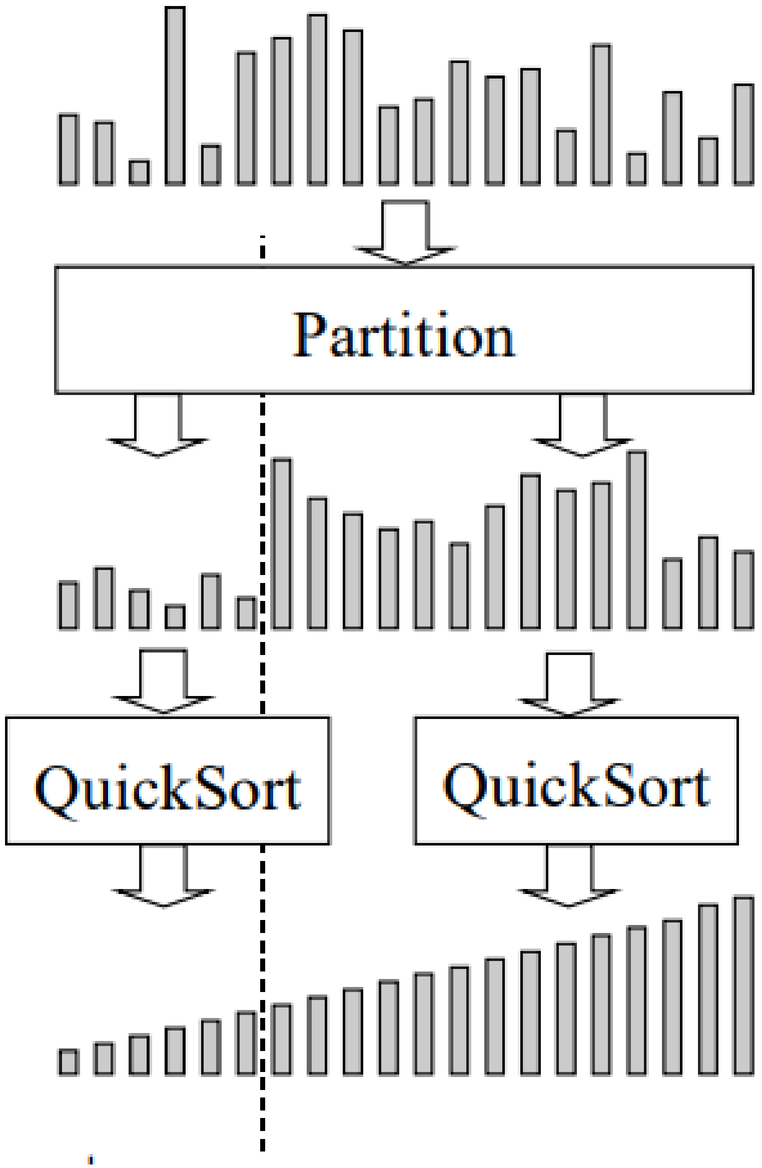
Quick Sort


Partition
≤p
p
≥p
QuickSort
QuickSort
pivot


Merge Sort:
Merge sort
Mergesort
Merge (ผสาน)

Quick Sort

```
quickSort(d[1..n], left, right) {
  if (left >= right) return
  j = partition(d, left, right)
  quickSort(d, left, j - 1)
  quickSort(d, j + 1, right)
}
```

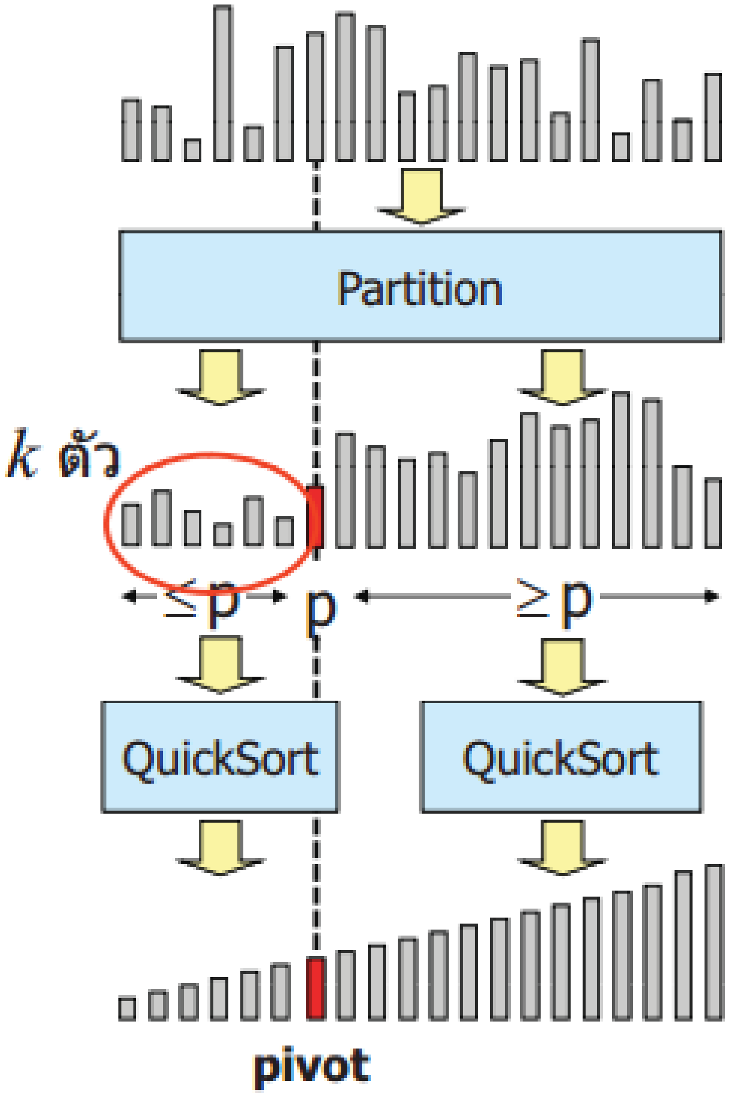


Quick Sort

```
partition( d[1..n], left, right ) {
  p = d[left]
  i = left, j = right + 1
  while (i < j) {
    while (d[--j] > p) ;
    while (d[++i] < p) if (i == right) break
    if (i < j) d[i] ↔ d[j]
  }
  d[left] ↔ d[j]
  return j
}
```

Quick Sort

ให้ $c(n)$ คือ #cmps ในการ quicksort ข้อมูลจำนวน n ตัว



$$n - 1$$

$$c(n) = c(k) + c(n-k-1) + n-1$$

$$c(k) + c(n-k-1)$$

Quick Sort: กรณีเร็วสุด

หลัง partition จำนวนข้อมูลของทั้งสองเท่ากัน

$$
\begin{aligned}
c(n) &= c(k) + c(n-k-1) + (n-1) \\
c_{min}(n) &= c_{min}(n/2) + c_{min}(n-n/2-1) + (n-1) \\
&= 2c_{min}(n/2) + (n-1) \\
&= \Theta(n \log n)
\end{aligned}
$$

Quick Sort: กรณีเฉลี่ย

$$c(n) = c(k) + c(n-k-1) + (n-1)$$

$$c_{\text{avg}}(n) = \frac{1}{n}\sum_{k=0}^{n-1}\Big(c_{\text{avg}}(k) + c_{\text{avg}}(n-k-1)\Big) + (n-1)$$

$$= \frac{2}{n}\sum_{k=0}^{n-1} c_{\text{avg}}(k) + (n-1)$$

$$nc_{\text{avg}}(n) = 2\sum_{k=0}^{n-1} c_{\text{avg}}(k) + n(n-1)$$

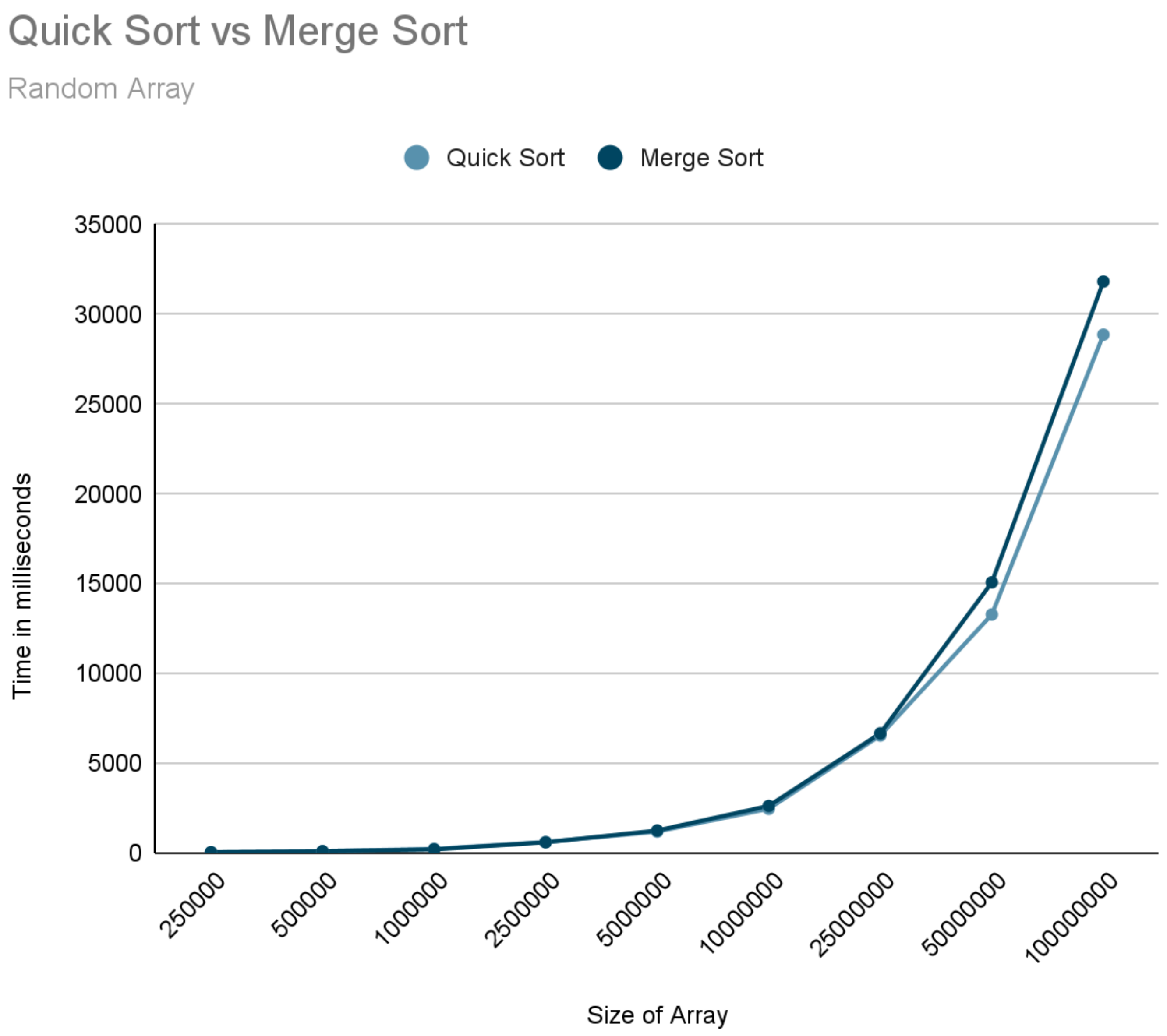
$$(n-1)c_{\text{avg}}(n-1) = 2\sum_{k=0}^{n-2} c_{\text{avg}}(k) + (n-1)(n-2)$$

$$nc_{\text{avg}}(n) = (n+1)c_{\text{avg}}(n-1) + 2(n-1)$$

Quick Sort: เวลาการทำงาน

- **Quicksort**
  - กรณีช้าสุด : $\Theta(n^2)$
  - กรณีเร็วสุด : $\Theta(n \log n)$
  - กรณีเฉลี่ย : $\Theta(n \log n)$
- **เพื่อหลีกเลี่ยงกรณีช้าสุด เลือก pivot แบบสุ่ม**
  - มีโอกาสสูงที่ทำให้ quicksort ทำงานในเวลา $\Theta(n \log n)$

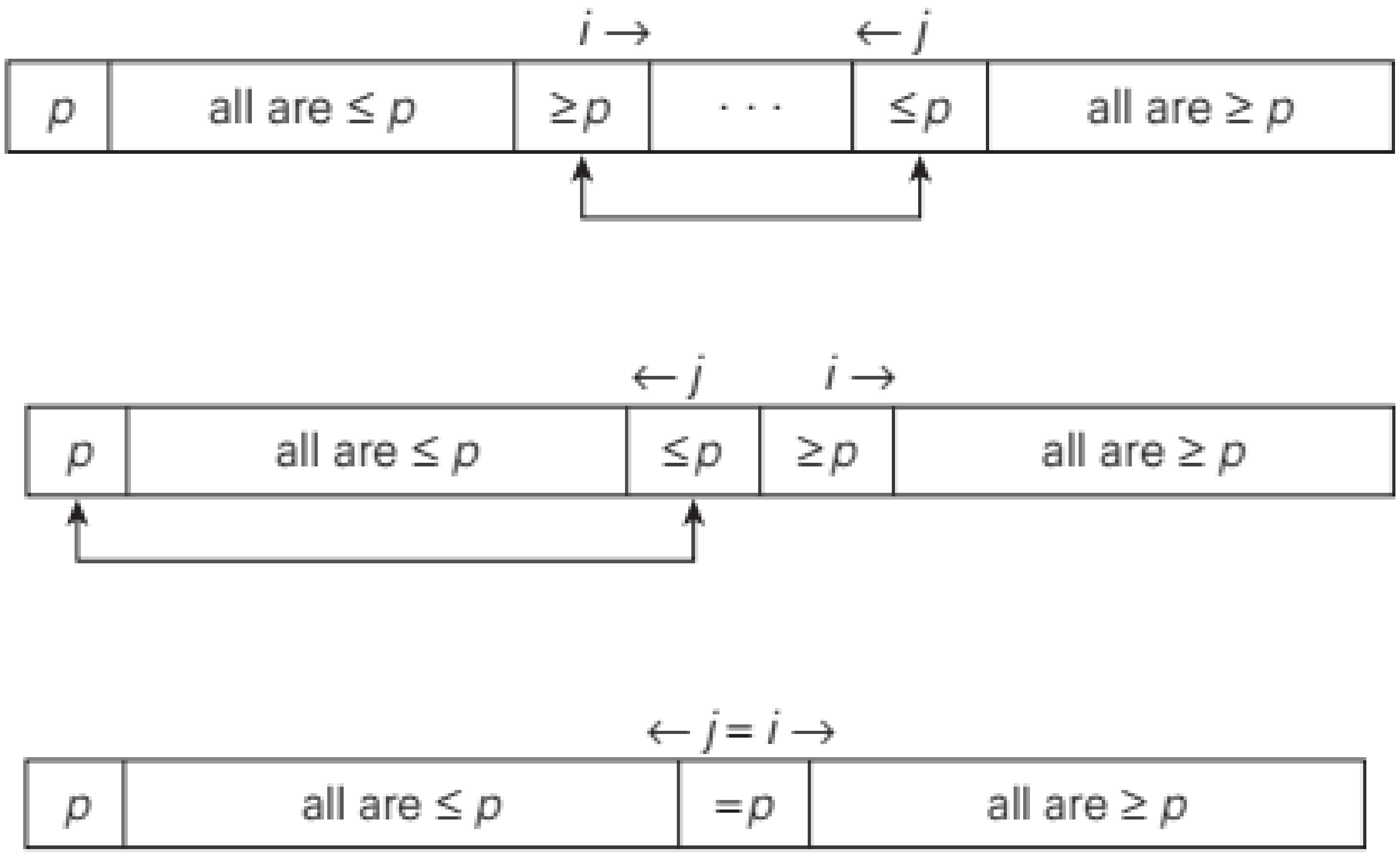
```
partition( d[1..n], left, right ) {
  k = random(left, right)
  d[k] ↔ d[left]
  p = d[left]
  i = left, j = right + 1
  ...
```



https://www.interviewkickstart.com/learn/merge-sort-vs-quicksort-performance-analysis

Quick Sort

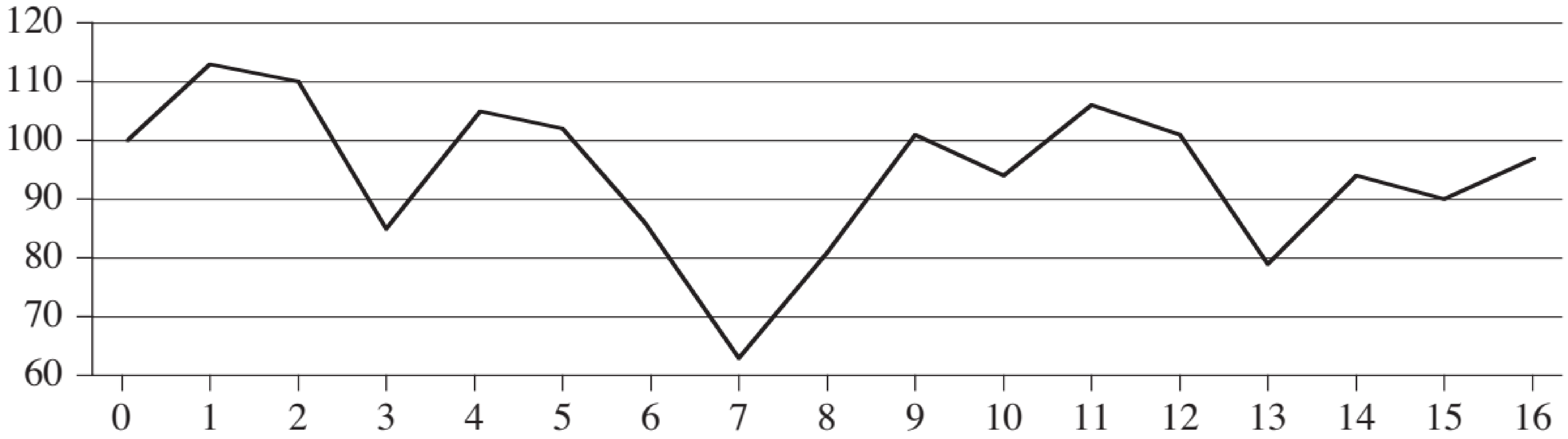
$$\underbrace{A[0]\dots A[s-1]}_{\text{all are } \le A[s]} \quad A[s] \quad \underbrace{A[s+1]\dots A[n-1]}_{\text{all are } \ge A[s]}$$



The Maximum-subarray Problem

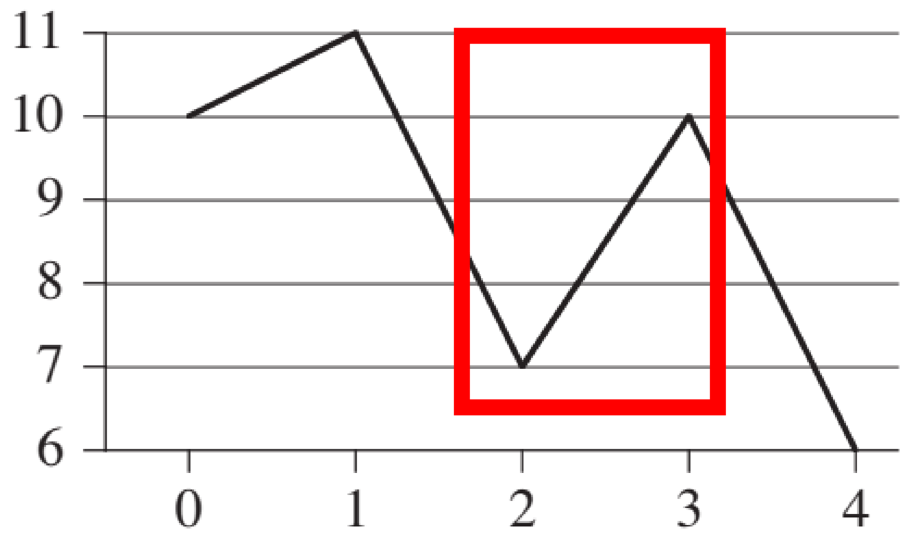
- Suppose that you have been offered the opportunity to invest in the Volatile Chemical Corporation.
  - The stock price of the Volatile Chemical Corporation is rather volatile.
  - You are allowed to buy one unit of stock only one time and then sell it at a later date, buying and selling after the close of trading for the day.
  - To compensate for this restriction, you are allowed to learn what the price of the stock will be in the future.
  - **Your goal is to maximize your profit.**

The Maximum-subarray Problem



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

The Maximum-subarray Problem



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

An example showing that the maximum profit does not always start at the lowest price or end at the highest price. Again, the horizontal axis indicates the day, and the vertical axis shows the price. Here, the maximum profit of $3 per share would be earned by buying after day 2 and selling after day 3. The price of $7 after day 2 is not the lowest price overall, and the price of $10 after day 3 is not the highest price overall.

Come on… Think about brute force…

Brute force

- We can easily devise a brute-force solution to this problem: just try every possible pair of buy and sell dates in which the buy date precedes the sell date.
- For n days, it has $\binom{n}{2}$ pairs of dates that you can buy the share.
- So $\binom{n}{2}$ is $\Theta(n^2)$

Is there any better way to solve this?

Transformation

- Instead of looking at the daily prices, let us instead consider the daily change in price, where the change on day i is the difference between the prices after day $i-1$ and after day i.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

- we now want to find the nonempty, contiguous subarray of A whose values have the largest sum. We call this contiguous subarray the **maximum subarray**.
  - From the above example, you would want to buy the stock just **before day 8** (that is, after day 7) and sell **it after day 11**, earning a profit of \$43 per share.
- Although computing the cost of one subarray might take time proportional to the length of the subarray, when computing all $\Theta(n^2)$ subarray sums, **we can organize the computation so that each subarray sum takes $O(1)$ time**, given the values of previously computed subarray sums, so that the brute-force solution takes $\Theta(n^2)$ time.

Non-negative array?

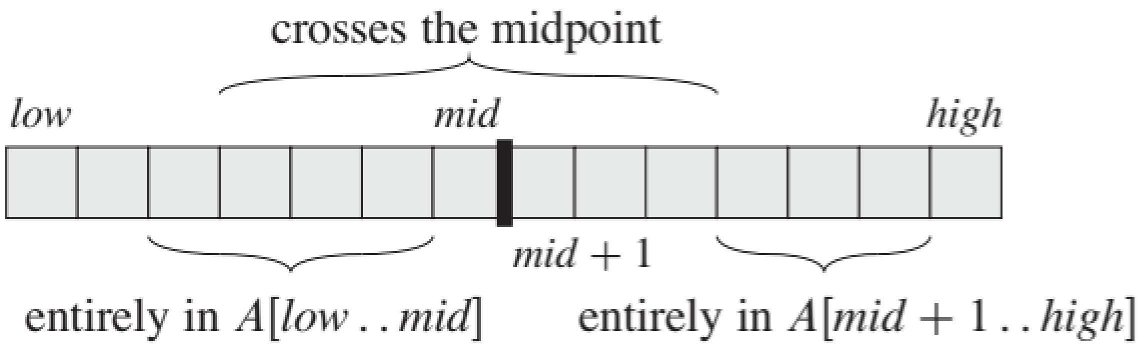
- What if all elements in the array are all non-negative numbers?

A solution using divide-and-conquer

- Divide-and-conquer suggests that we divide the subarray into two subarrays of as equal size as possible.
- Suppose we want to find the **maximum subarray** $A = [low \ldots high]$,
- First, we find the midpoint, say mid, of the subarray, and consider the subarrays $A[low \ldots mid]$ and $A[mid + 1 \ldots high]$.

A solution using divide-and-conquer

- Consider the following figure:

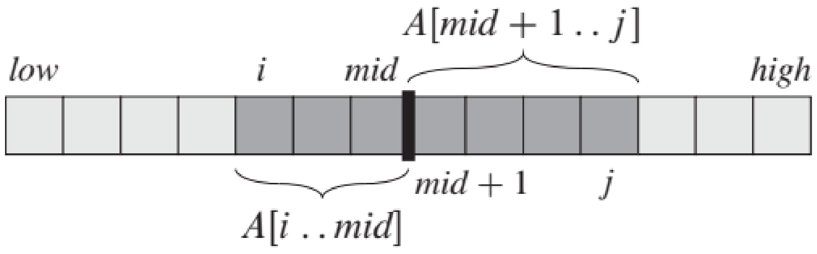


(a)

- Any contiguous subarray $A[i \ldots j]$ of $A[low \ldots high]$ must lie in exactly one of the following places:
  - entirely in the subarray $A[low \ldots mid]$, so that $low \leq i \leq j \leq mid$,
  - entirely in the subarray $A[mid + 1 \ldots high]$, so that $mid < i \leq j \leq high$,
  - crossing the midpoint, so that $low \leq i \leq mid < j \leq high$.

A solution using divide-and-conquer

- We can easily find a maximum subarray crossing the midpoint in time **linear** in the size of the subarray $A[low \dots high]$.
- This problem is **not a smaller instance** of our original problem, because it has the added restriction that the subarray it chooses must cross the midpoint.



(b)

Any subarray crossing the midpoint is itself made of two subarrays $A[i \dots mid]$ and $A[mid + 1 \dots j]$, where $low \le i \le mid$ and $mid \le j \le high$.
Therefore, we just need to find maximum subarrays of the form $A[i \dots mid]$ and $A[mid + 1 \dots j]$, and then combine them.

A solution using divide-and-conquer

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

A solution using divide-and-conquer

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])          // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

Complexity O(n log n)

แบบฝึกหัด

- เขียนโปรแกรมเพื่อหา Maximum subarray ด้วย Divide and Conquer Approach

Better algorithms for maximum subarray

- The Kadane’s Algorithm in O(n).
  - https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/
- Not covered in here.

Interesting Usage of Binary Search

- Binary Search: เช็คตรงกลาง Sorted array ว่าตรงกับ Search key ที่ต้องการหรือไม่
  - ถ้าพบ => หยุดการทำงาน
  - ถ้าไม่พบ => ค้นหาต่อ (ทางซ้ายหรือขวา) จนกว่าจะไม่มีสมาชิกใน Array ให้หาอีกต่อไป
  - Complexity => O(log n)
  - Prerequisite: <u>Static Sorted</u> Array

Interesting Usage of Binary Search

- **Bisection Method**
- หลักการของ Binary search อาจจะใช้ในการหา Root ของ Function ที่ยากจะคำนวณ
- เช่น คุณซื้อของสิ่งหนึ่งด้วยเงินกู้ และต้องผ่อนทุกๆ เดือน เดือนละ d บาท ทั้งหมด m เดือน สมมติว่ารถยนต์ราคา v บาทและธนาคารเก็บดอกเบี้ย i% ของทุกสิ้นเดือนที่คุณยังไม่ได้ชำระ คำถาม => เงินจำนวนเท่าใดที่คุณต้องจ่ายต่อเดือน ?
  - **d=576.19**, m=2, v=1000, i=10%

    ยอดที่ยังไม่ได้จ่าย **v=1000; 1.1** คือ 110**%**

    จำนวนเงินที่ชำระเดือนนี้
  - 1 เดือนผ่านไปยอดกู้ของคุณคือ **1000*1.1** – **576.19** = **523.81**
  - 2 เดือนผ่านไปยอดกู้ของคุณคือ **523.81*1.1** – **576.19** ≈ **0**
  - สิ่งที่เราต้องการวางแผนคือค่า d หามาจากไหน?

Interesting Usage of Binary Search

- **Bisection Method (ต่อ)**
- เริ่มต้นกำหนดค่า d ให้อยู่ในช่วงที่ต้องการ [a … b]
  - a = 0.01, b = 1000*1.1 (สมเหตุสมผลไหม ?)

จ่ายขาดไป ต้องจ่ายเพิ่ม

54.9895 คำนวณจาก **f**

จ่ายเกิน

กำลังดี

a	b	d $= \frac{a+b}{2}$	status: $f(d, m, v, i)$	action
0.01	1100.00	550.005	undershoot by 54.9895	increase d
550.005	1100.00	825.0025	overshoot by 522.50525	decrease d
550.005	825.0025	687.50375	overshoot by 233.757875	decrease d
550.005	687.50375	618.754375	overshoot by 89.384187	decrease d
550.005	618.754375	584.379688	overshoot by 17.197344	decrease d
550.005	584.379688	567.192344	undershoot by 18.896078	increase d
567.192344	584.379688	575.786016	undershoot by 0.849366	increase d
…	…	…	a few iterations later …	…
…	…	576.190476	stop; error is now less than ϵ	answer = 576.19

Transform and Conquer

Transform and Conquer

- Transform & Conquer is a general algorithm design technique which works in two stages.
  - STAGE 1: (Transformation stage): The problem’s instance is modified, more amenable to solution
  - STAGE 2: (Conquering stage): The transformed problem is solved

Transform and Conquer

- This group of techniques solves a problem by a transformation to
  - a simpler/more convenient instance of the same problem (instance simplification)
  - a different representation of the same instance (representation change)
  - a different problem for which an algorithm is already available (problem reduction)

Searching with presorting

- Problem: Search for a given K in A[0..n-1]
- Presorting-based algorithm:
  - Stage 1 Sort the array by an efficient sorting algorithm
  - Stage 2 Apply binary search

Efficiency: $\Theta(n\log n) + O(\log n) = \Theta(n\log n)$

Element Uniqueness with presorting

- Presorting-based algorithm
  - Stage 1: sort by efficient sorting algorithm (e.g. merge sort)
  - Stage 2: scan array to check pairs of adjacent elements

    Efficiency: $\Theta(n\log n) + O(n) = \Theta(n\log n)$
- Brute force algorithm
  - Compare all pairs of elements

    Efficiency: $O(n^2)$
- Another algorithm? Hashing

Searching Problem

- Problem: Given a (multi)set S of keys and a search key K, find an occurrence of K in S
- Searching must be considered in the context of:
  file size (internal vs. external)
  dynamics of data (static vs. dynamic)
- Dictionary operations (dynamic data):
  find (search)
  insert
  delete

References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.
- การออกแบบและวิเคราะห์อัลกอริทึม สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ สมชาย ประสิทธิ์จูตระกูล (2549)