演算法 排序

- 選擇排序 (selection sort) 、氣泡排序 (bubble sort) 、插入排序 (insertion sort)
- 合併排序 (merge sort)
- 快速排序 (quick sort)
- 比較排序演算法的時間下限
- 計數排序 (counting sort)

何謂排序？

- 輸入的n個數字序列 $<A_0, A_1, \ldots, A_{n-1}>$
  輸出結果有一定的順序 $<A'_0, A'_1, \ldots, A'_{n-1}>$
  例如「由小到大」： $A'_0 \leqq A'_1 \leqq \ldots \leqq A'_{n-1}$

- 範例
  - 輸入序列 <28，31，17，35，9，22>
  - 排序結果 <9，17，22，28，31，35>

最糟和平均時間複雜度列表 (4.5)

排序演算法	最糟執行時間	平均執行時間
選擇排序	$\Theta(n^2)$	$\Theta(n^2)$
氣泡排序	$\Theta(n^2)$	$\Theta(n^2)$
插入排序	$\Theta(n^2)$	$\Theta(n^2)$
合併排序	$\Theta(n\log n)$	$\Theta(n\log n)$
堆積排序	$\Theta(n\log n)$	$\Theta(n\log n)$
快速排序	$\Theta(n^2)$	$\Theta(n\log n)$

$$T(n) \in \Theta(n^d) \quad \Rightarrow \quad T(n) \in O(n^d)$$

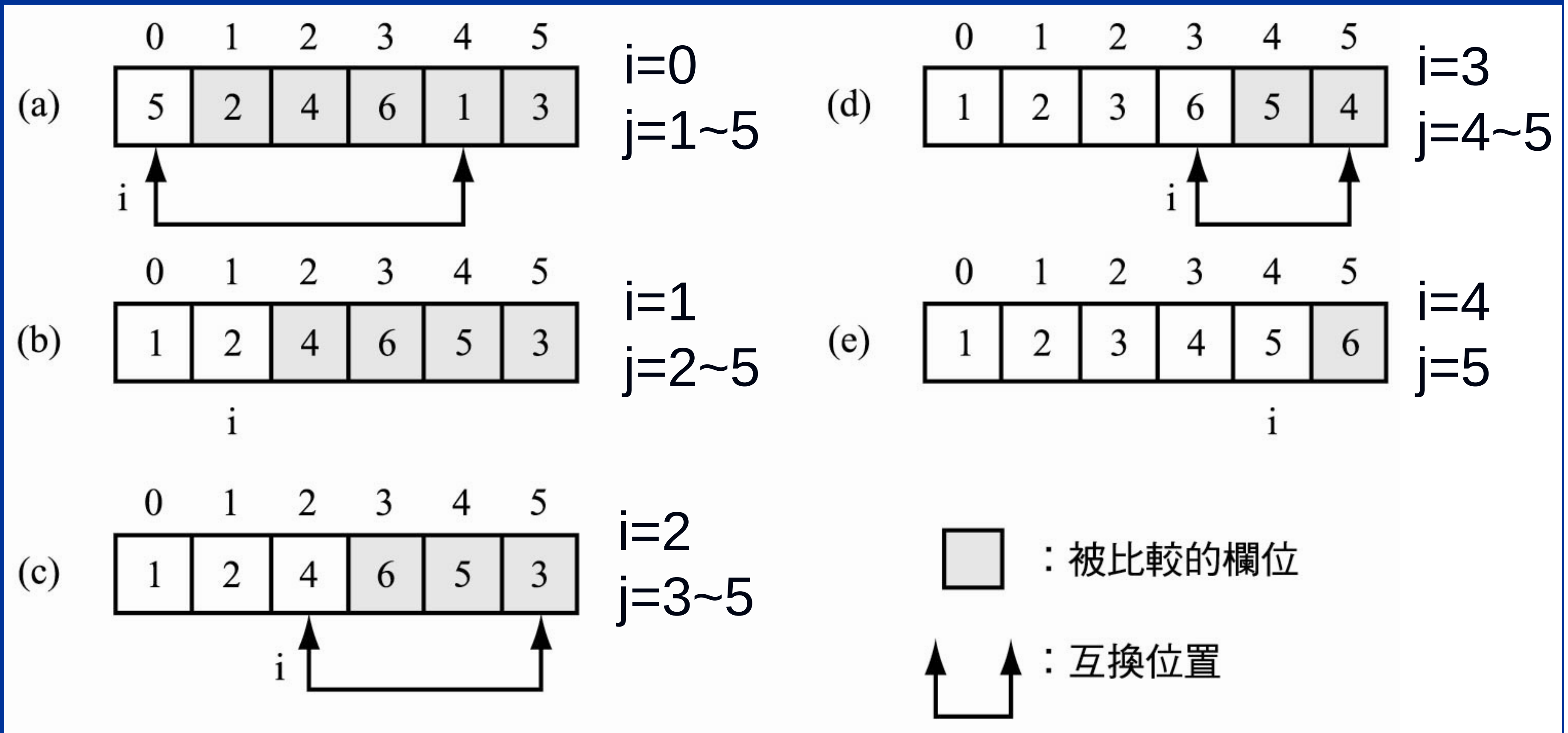
基本的排序演算法

- 選擇排序、氣泡排序、插入排序
  - 「直覺」、實作簡單
  - 適合小數目元素的排序

- 時間複雜度 $\Theta(n^2)$
  空間複雜度爲 $\Theta(1)$

選擇排序法

從全部未排序好的元素中，挑出最小的一個放在最左邊



選擇排序法

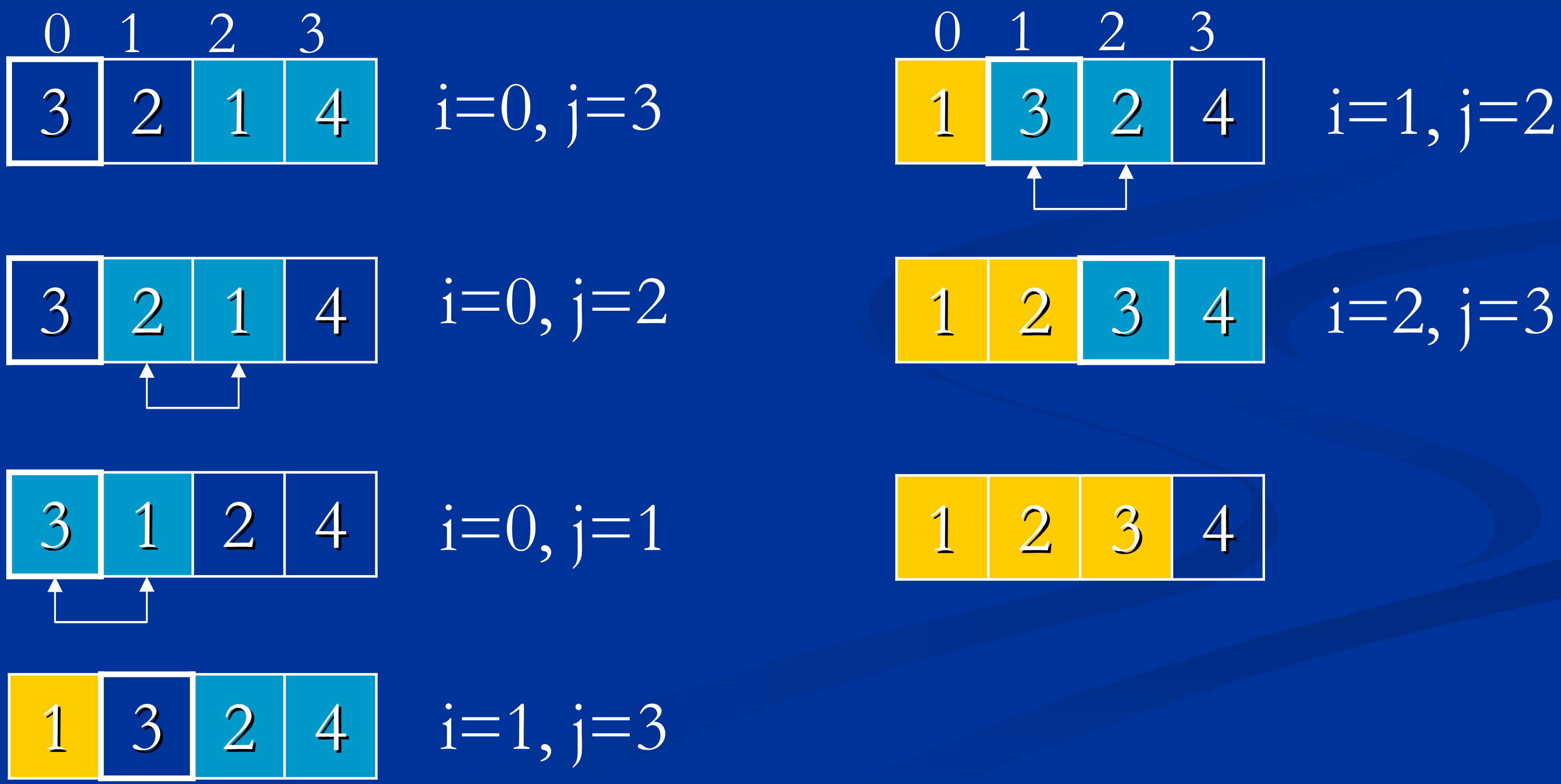
```
void SelectionSort(int A[ ],int n){
  int i, j, selected;
  for(i=0;i<=n-2;i++){       selected=i;
     for(j=i+1;j<=n-1;j++)
          if (A[j]<A[selected]) selected=j;
     if (i != selected) swap(A, i, selected);
  }
}
```

時間複雜度 $\Theta(n^2)$

空間複雜度爲 $\Theta(1)$

氣泡排序法

- 從全部未排序好的元素中，相鄰元素兩兩相比，較小的往前移, 最小的就會移到最邊邊



氣泡排序法

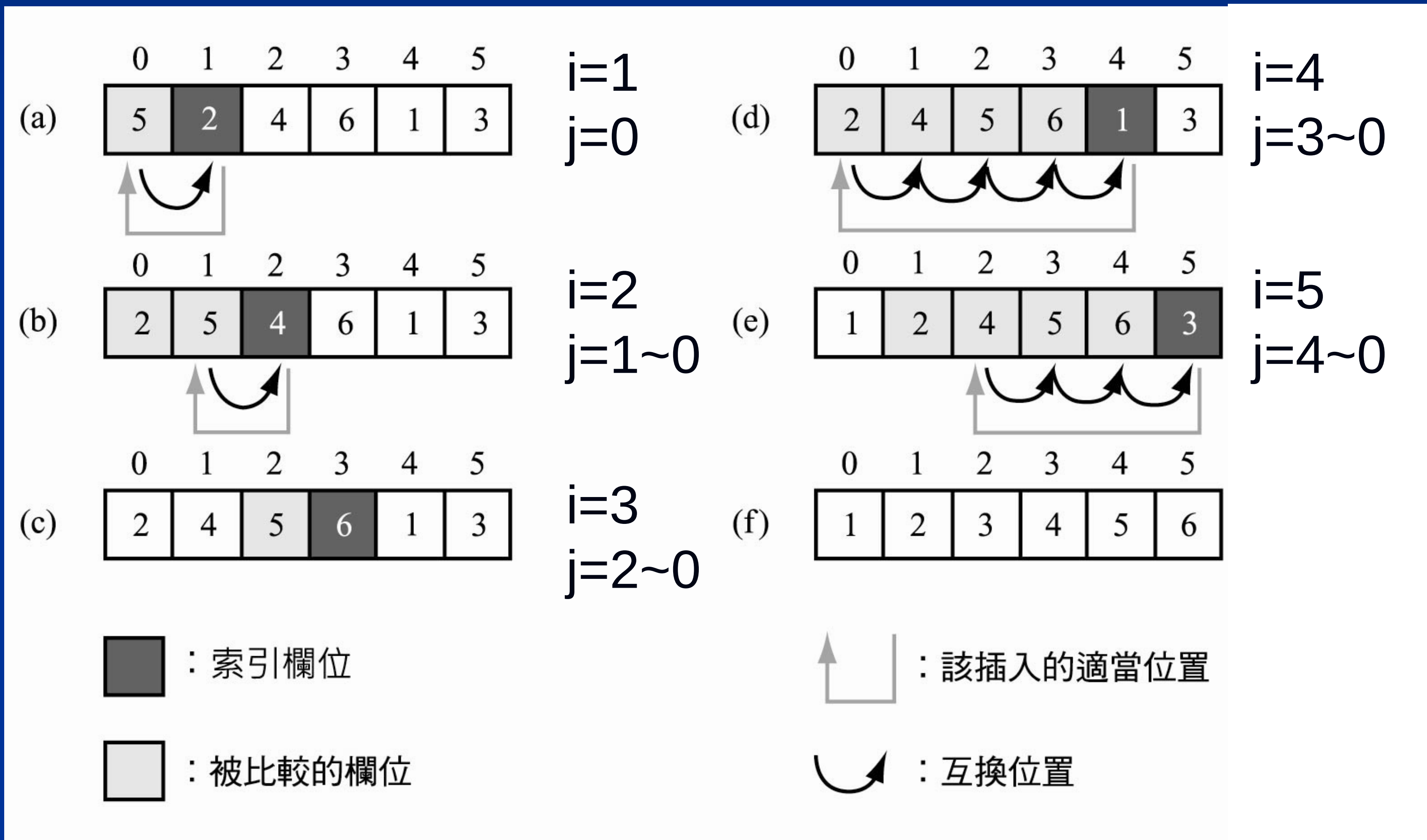
```
void BubbleSort(int A[ ],int n){
   int i, j;
   for(i=0;i<=n-2;i++)
        for(j=n-1;j>=i+1;j--)
                if (A[j]<A[j-1]) swap(A, j, j-1);
}
```

時間複雜度 $\Theta(n^2)$

空間複雜度爲 $\Theta(1)$

範例 5-1 插入排序法

由左到右，一次插入一個數字到適當的位置



由演算法設計程式

i=1
j=0

i=2
j=1~0

i=3
j=2~0

i=4
j=3~0

i=5
j=4~0

```
n=6;

for(i=1;i<=n-1;i++) {   // i 由 1 到 n-1
      temp=A[i];
      for(j=i-1;j>=0;j--)  // j 由 i-1 到 0
            if (temp<A[j])
                  A[j+1]=A[j];
            else
                  break;
      A[j+1]=temp;
}
```

插入排序法

```
void InsertionSort(int A[ ],int n){
        int i,j,temp;
        for(i=1;i<=n-1;i++) {
                temp=A[i];
                for(j=i-1;j>=0;j--)
                     if (temp<A[j]) A[j+1]=A[j];
                     else  break;
                A[j+1]=temp;
        }
}
```

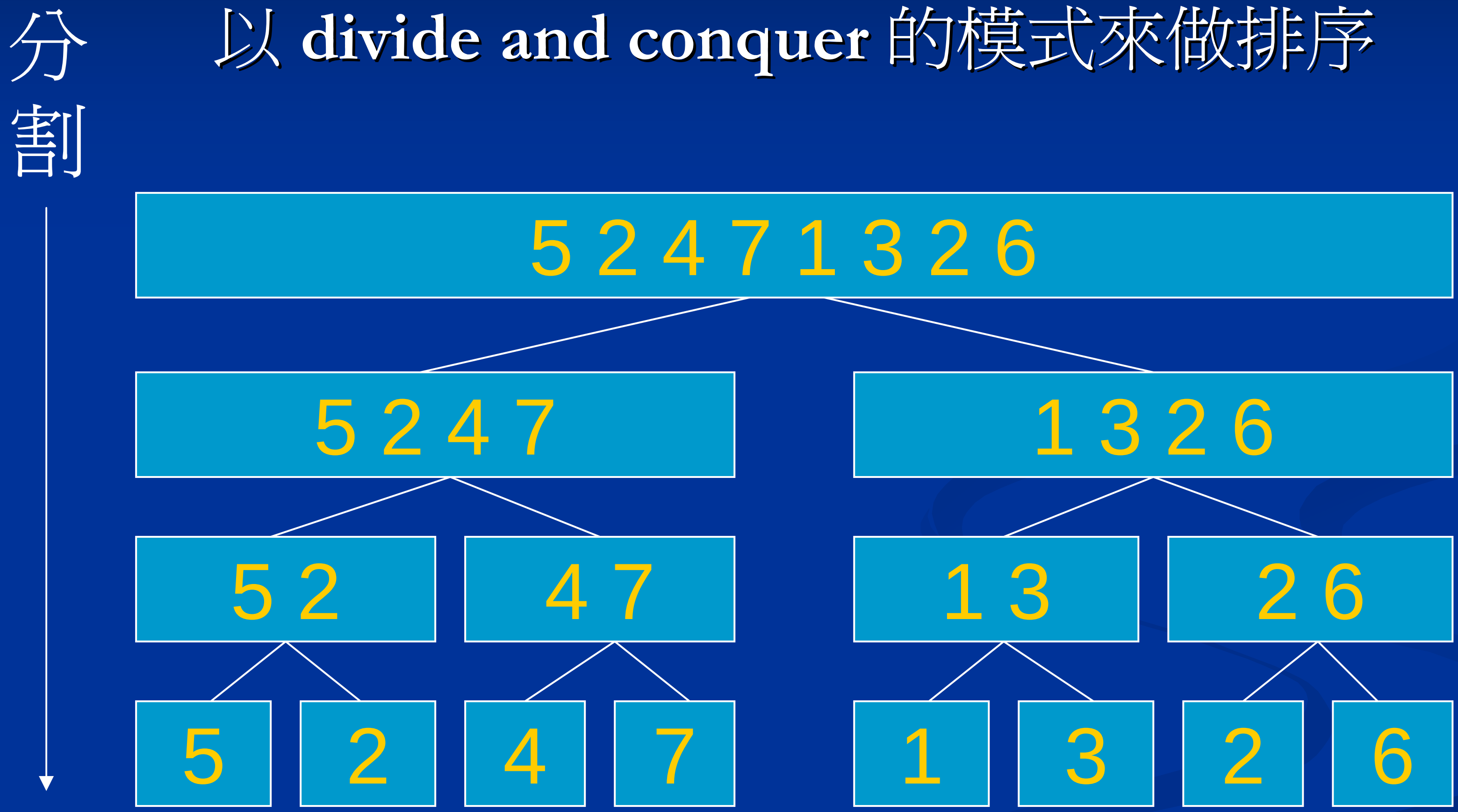
插入排序法

- 最遭情況需要進行 C(n) 個比較
  $C(n)=(n-1) + (n-2) + \ldots +1 = \frac{1}{2}(n-1)n$
  $= \frac{1}{2}(-n+n^2)$
  平均需要 $C(n) \approx n^2/4$
  (j 迴圈平均只需要進行一半)

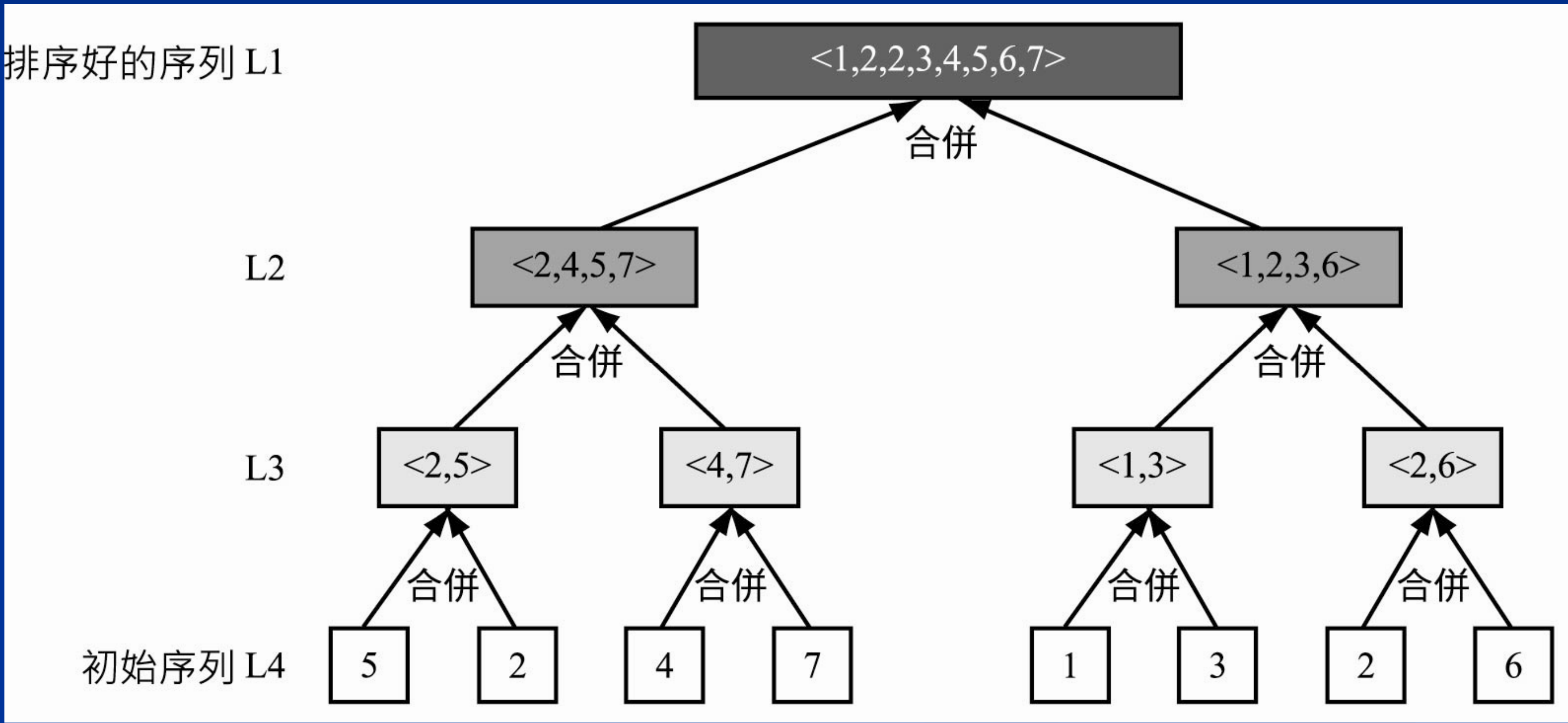
- 最遭情況與平均時間複雜度 $\Theta(n^2)$
  空間複雜度 $\Theta(1)$

合併排序

以 divide and conquer 的模式來做排序



合併排序

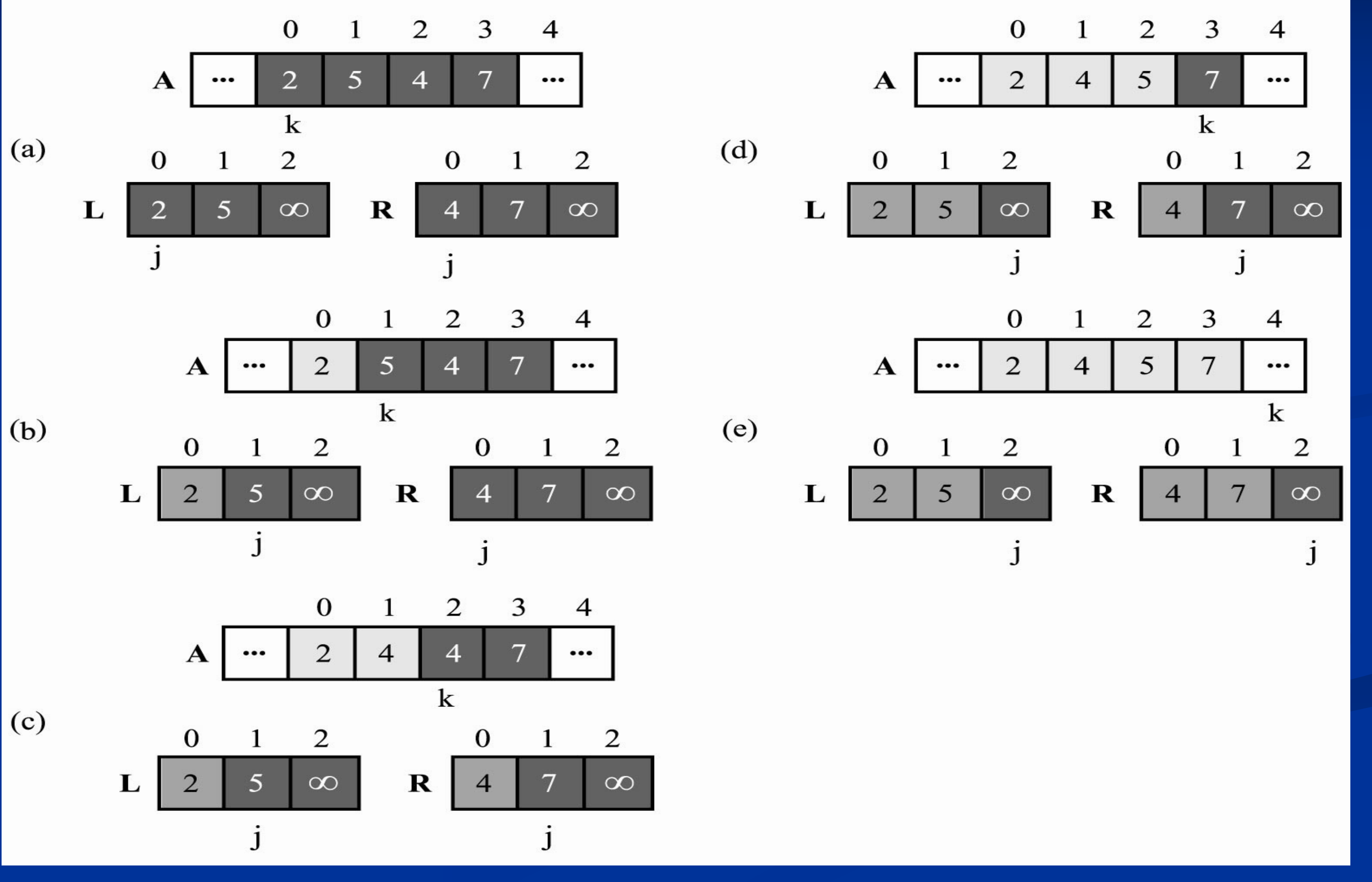


排序好的序列 L1
<1,2,2,3,4,5,6,7>
合併
L2
<2,4,5,7>
<1,2,3,6>
合併
合併
L3
<2,5>
<4,7>
<1,3>
<2,6>
合併
合併
合併
合併
初始序列 L4
5
2
4
7
1
3
2
6

合併排序

```
void MergeSort (int A[], int p, int r) {
   int q;
   if (p<r){
        q=(p+r)/2;
        MergeSort(A, p, q);
        MergeSort(A, q+1, r);
        Merge(A, p, q, r);
   }
}
```

合併之處理



```
void Merge(int A[], int p, int q, int r){
  int i, j, k, n1, n2, *L, *R;
    n1=q-p+1; n2=r-q;
    L=(int*)malloc(sizeof(int)*(n1+1));
    R=(int*)malloc(sizeof(int)*(n2+1));
    for (i=0;i<n1;i++) L[i]=A[p+i];
    for (j=0;j<n2;j++) R[j]=A[q+j+1];
    L[n1]=0x7FFFFFFF; R[n2]=L[n1];
  for(k=p,i=0,j=0; k<=r; k++) {
    if (L[i]<R[j])  { A[k]=L[i]; i++;}
    else { A[k]=R[j]; j++;}
  }  free(L); free(R);
}
```

配置並設定
左右陣列

範例 5-2 合併排序

問題規模　資料結構: 陣列

輸入:隨機產生數值

```
main(){
  int n=10;  int A[n];   int k;
  srand(time(0));
  for(k=0; k<n; k++) { A[k]=rand()%n;
      printf("%d ", A[k]);
  } printf("\n");
  MergeSort(A, 0, n-1);
  for(k=0; k<n; k++) printf("%d ", A[k]);
}
```

演算法

輸出:排序之結果

測試結果

參考範例 5-2

- n=5　　3 0 4 3 2
  0 2 3 3 4
- n=10　　1 8 0 3 4 8 7 2 2 8
  0 1 2 2 3 4 7 8 8 8
- n=15　　2 2 2 5 5 0 9 10 1 4 1 13 5 14 1
  0 1 1 1 2 2 2 4 5 5 5 9 10 13 14

合併排序的複雜度

- 時間複雜度

$$T(n)=\begin{cases}\Theta(1) & \text{如果} n=1\\ 2T\left(\dfrac{n}{2}\right)+\Theta(n) & \text{如果} n>1\end{cases}$$

依據支配理論解析此遞迴公式，T(n) ∈ Θ(n*log*n)

T(n) ∈ O(nlogn)

- 空間複雜度

每次呼叫Merge()最多需要Θ(n)，離開就歸還，

此整體遞迴堆疊空間爲 S(n)∈Θ(n)

參考範例 5-2

效能測量

- 設定較大的 n 值來測試演算法

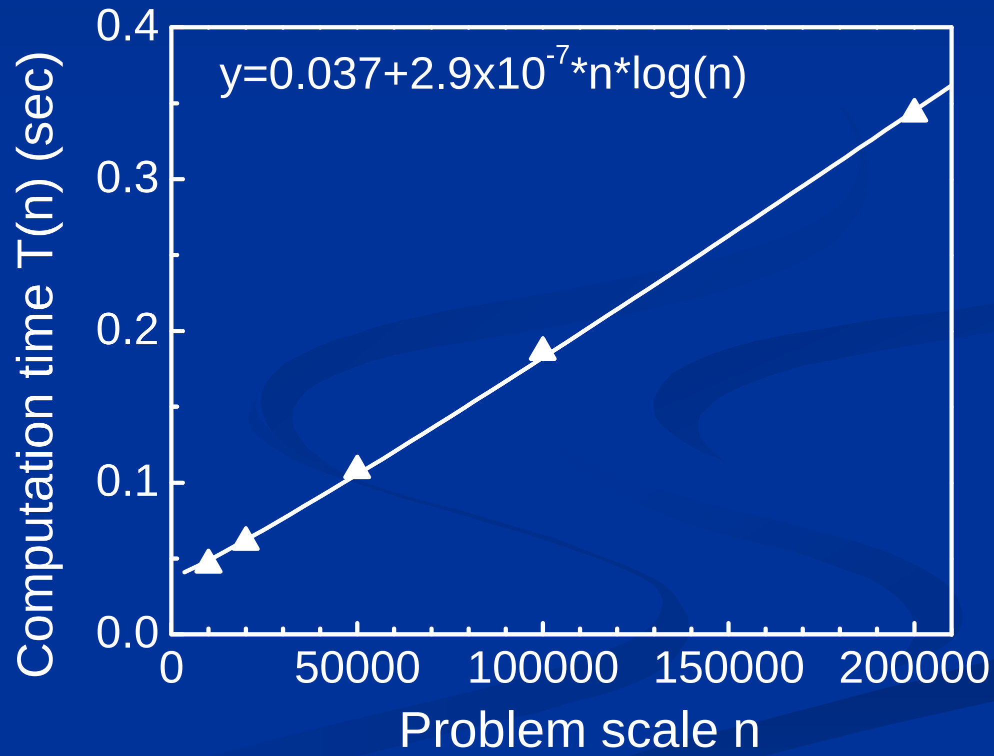
n=10000 0.046s

n=20000 0.061s

n=50000 0.108s

n=100000 0.186s

n=200000 0.343s



在地排序 (in place sorting)

- 排序的過程中，只有常數個數的輸入元素曾經被複製到另外取得的空間。
- 氣泡、選擇、插入排序使用一個元素 ***temp***的空間來做兩個元素位置的互換，屬於在地排序。
- 合併排序：需要另外取得最大到 n (L、R兩陣列)的空間來做合併，不屬於在地排序。

  →當 n 非常大時，須使用虛擬記憶體 →耗費時間

快速排序

- Divide and conquer, 在地排序
- 分割：將陣列 A[p...r] 分割成左、中、右三段
  - 使用 Partition() 找出 q 值使得

    左 A[p...q-1] ≤ A[q] ≤ A[q+1...r]
  - 中段 A[q] 又稱爲中樞 **(pivot)** ，左、右兩段可能長度爲 0
- 處理：遞迴排序 A[p...q-1] 和 A[q+1...r]
- 合併：不需要合併, A[p...r] 即排序完成

快速排序

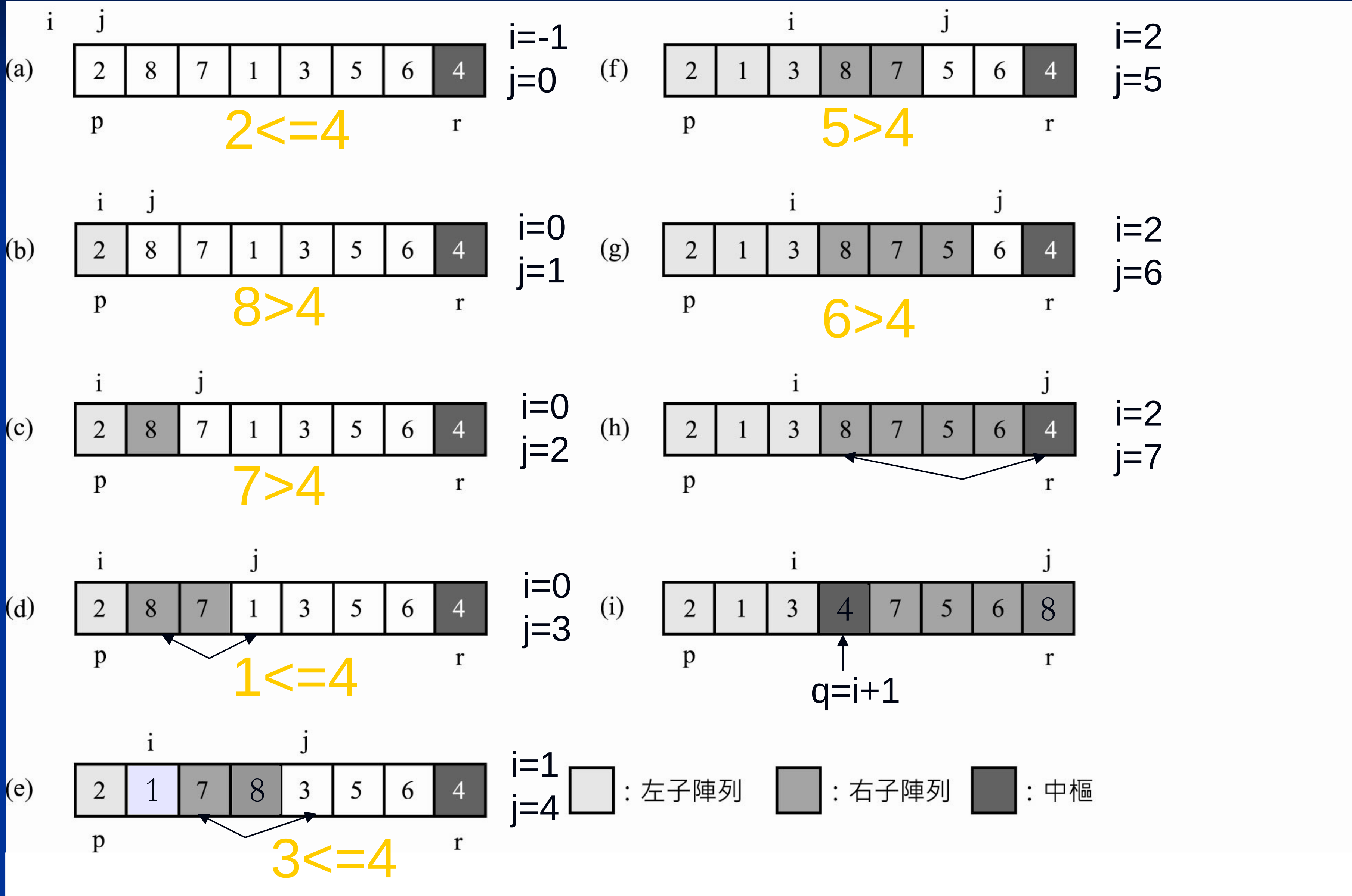
```
void QuickSort (int A[], int p, int r) {
  int q;
  if (p<r){
      q=Partition(A, p, r);
      QuickSort(A, p, q-1);
      QuickSort(A, q+1, r);
  }
}
```

分割程序 Partition (A, p, r)

- 假設陣列的最後一個元素值 A[r] 為中樞
- 依序將 A[p...r-1] 跟中樞A[r]做比較
  小於或等於A[r] 的放在左段 (左段最後為 i)
  否則放在右段 (右段最後為 j)

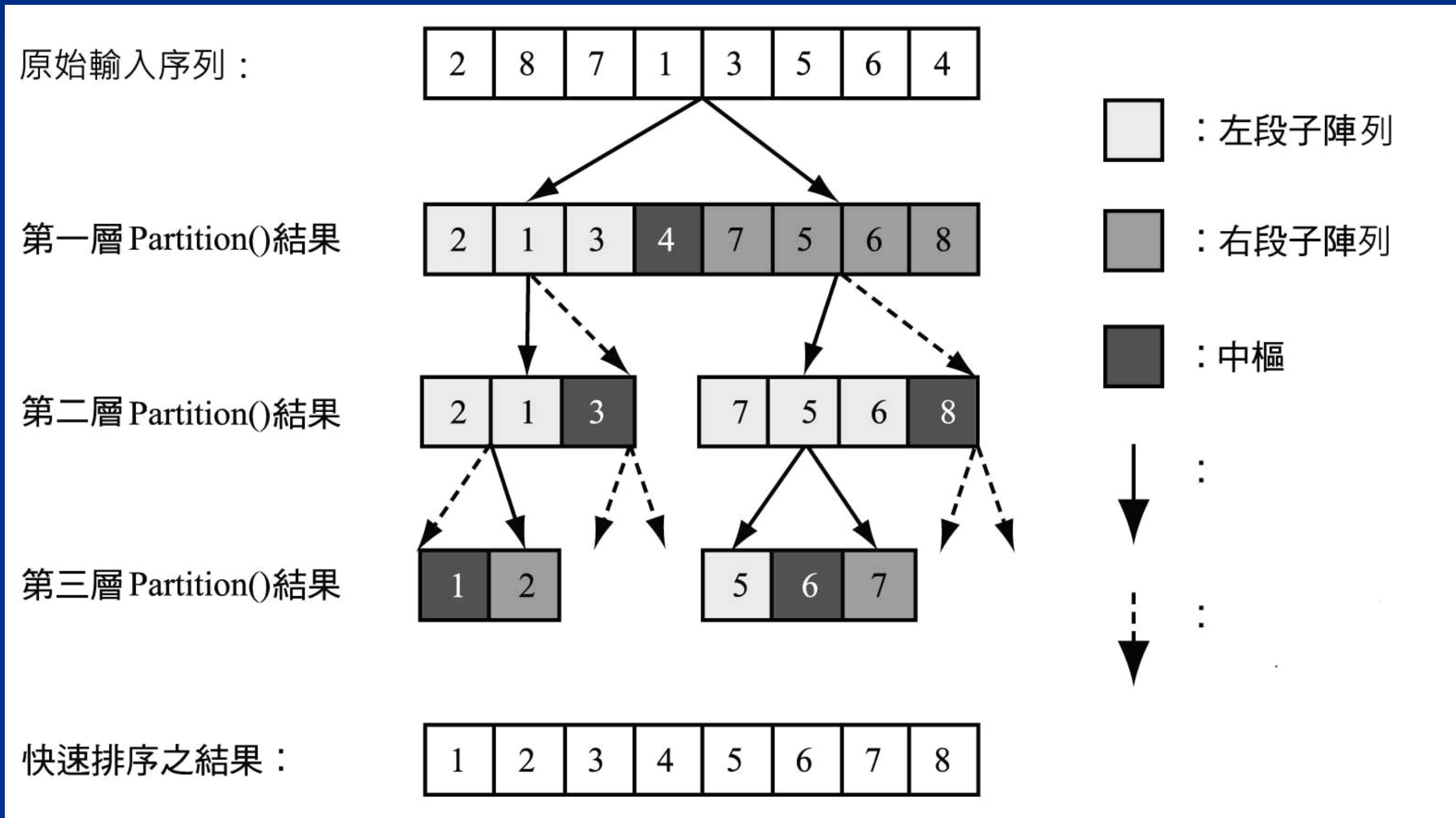
- 最後再將 A[r] 的值放到左右兩段的中間
  並回傳該索引值 q

分割程序 Partition ()



```
int Partition(int A[], int p, int r){
   int j, x=A[r], i=p-1;
   for(j=p; j<r;j++) {
        if (A[j]<=x){
                i++;
                if (i!=j) swap(A, i, j);
        }
   }
   swap(A, i+1, j);
   return i+1;
}
```

快速排序


原始輸入序列：
2 8 7 1 3 5 6 4
第一層Partition()結果
2 1 3 4 7 5 6 8
第二層Partition()結果
2 1 3
7 5 6 8
第三層Partition()結果
1 2
5 6 7
快速排序之結果：
1 2 3 4 5 6 7 8
：左段子陣列
：右段子陣列
：中樞

範例 5-4 快速排序

問題規模　　資料結構: 陣列

輸入:隨機產生數值

```
main(){
  int n=10;   int A[n];      int k;
  srand(time(0));
  for(k=0; k<n; k++) { A[k]=rand()%n;
      printf("%d ", A[k]);
  } printf("\n");
  QuickSort(A, 0, n-1);
  for(k=0; k<n; k++) printf("%d ", A[k]);
}
```

演算法

輸出:排序之結果

測試結果

參考範例 5-4

- n=5 1 3 3 4 0
  0 1 3 3 4
- n=10 4 9 1 8 7 9 0 9 4 7
  0 1 4 4 7 7 8 9 9 9
- n=15 7 7 12 8 10 13 2 8 5 0 8 9 10 9 3
  0 2 3 5 7 7 8 8 8 9 9 10 10 12 13

效能分析

- Partition() 時間複雜度: $\Theta(n)$
  需要 n-1 次跟中樞的比較，最多需要 n-1 次互換位置

- 最糟情況
  - 輸入序列剛好是由小到大 (已排序好了)
  - 每遞迴呼叫一次 Partition () 函式只能決定中樞一個元素的位置

快速排序 最糟情況

1	2	3	4	5

n-1 比較

1	2	3	4

n-2 比較

1	2	3

n-3 比較

1	2

n-4 比較

最糟有
n-1個遞迴

最糟需要 (n-1)+(n-2)+...+1=$(n^2-n)/2$ 個比較

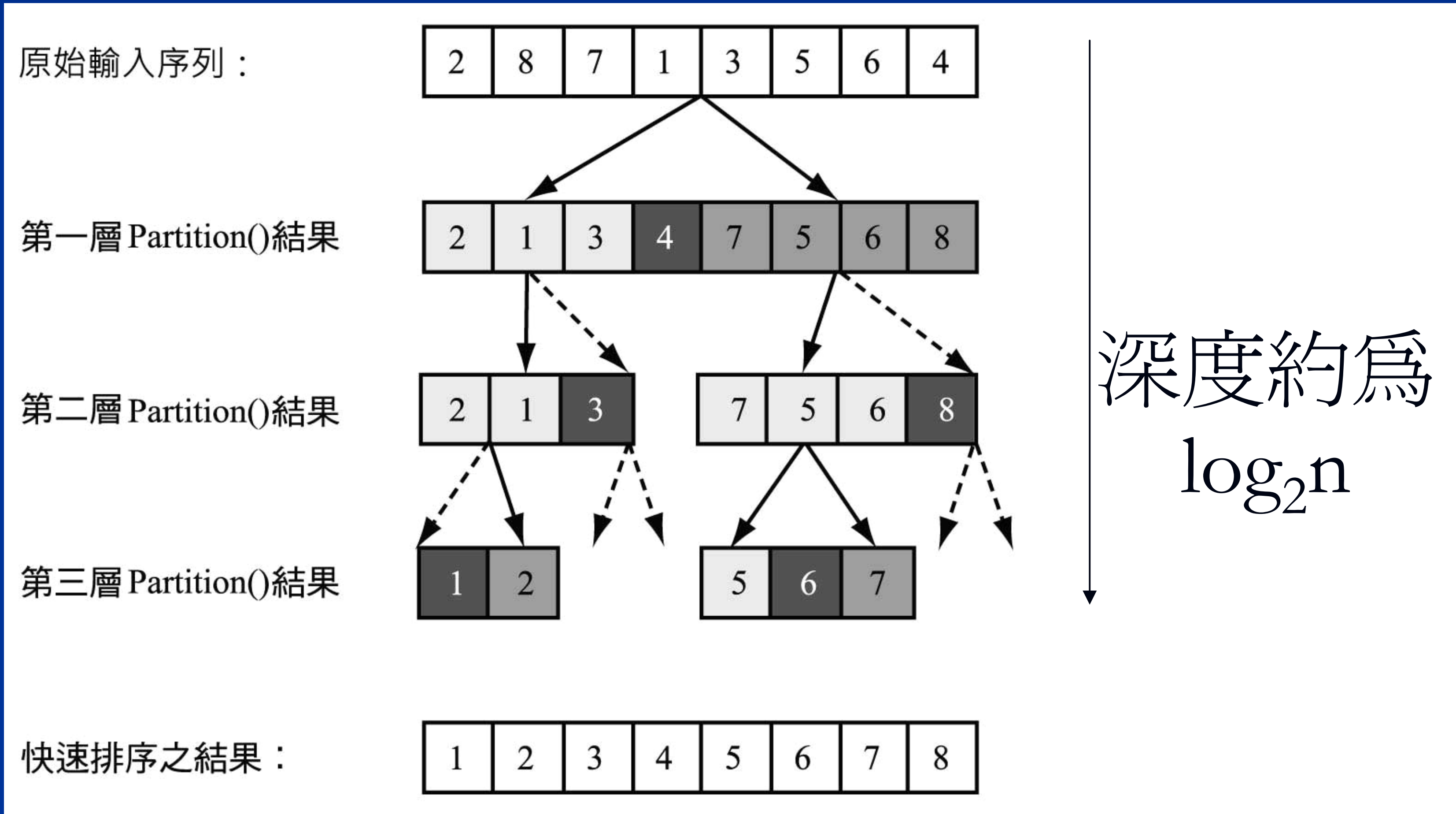
$T(n) \in \Theta(n^2)$ $S(n) \in \Theta(n)$

效能分析

- 最佳情況
  - 每次切割結果剛好左、右兩段子陣列的長度都大致相同
  - 相對應的遞迴二元樹爲平衡的
    深度爲 $\log_2 n$ （也就是 $\Theta(\mathit{log}n)$ ）
  - 執行時間 $T(n) \leq 2T(n/2) + \Theta(n)$
    由支配理論可知 $T(n) \in \Theta(n \log n)$

快速排序 最佳狀況

左段與右段長度差不多



效能分析

- 一般情況 (平均情況)
  假設中樞會發生在 0~n-1 之間的任何位置
  發生的機率為 1/n
  平均計算時間

$$T(0) = T(1) = C$$

$$T(n) = \frac{1}{n}\sum_{s=0}^{n-1}\left[T(s) + T(n-1-s) + \Theta(n)\right] \quad n > 1$$

$$T(n) \in \Theta(n \log n)$$

參考範例 5-4

效能測量

- 設定較大的 n 值來測試演算法

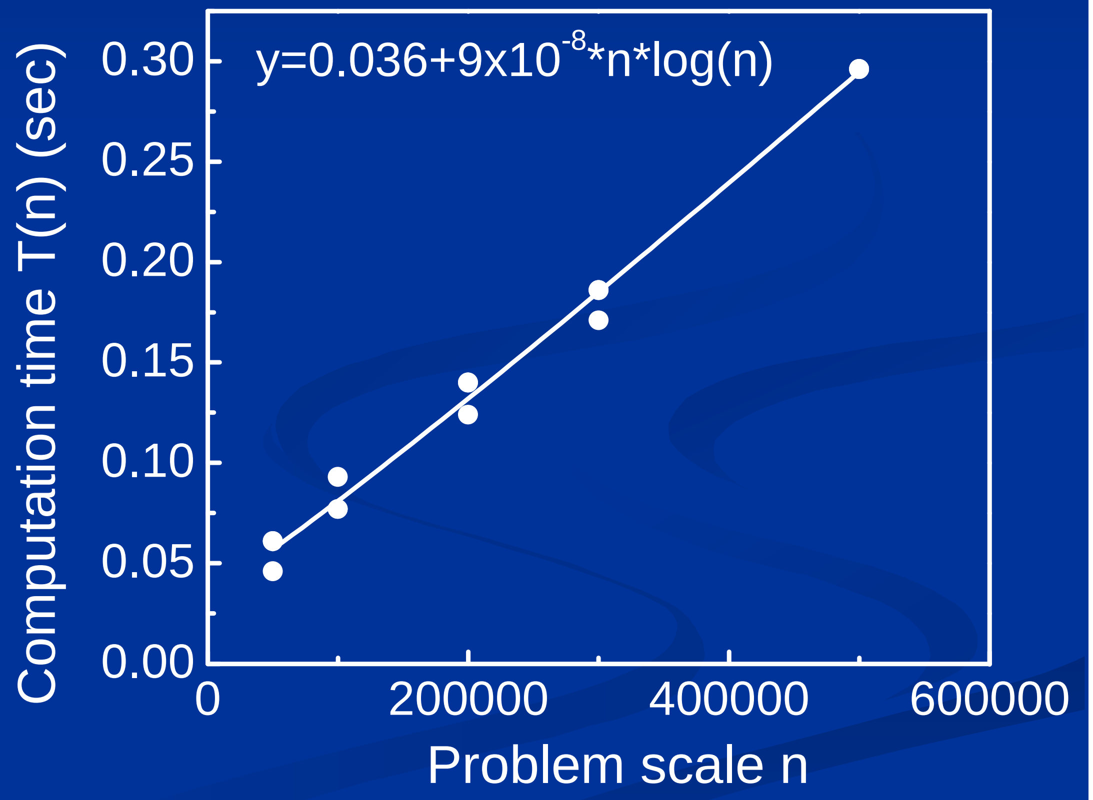
n= 50000 0.056 s

n=100000 0.082 s

n=200000 0.135 s

n=300000 0.181 s

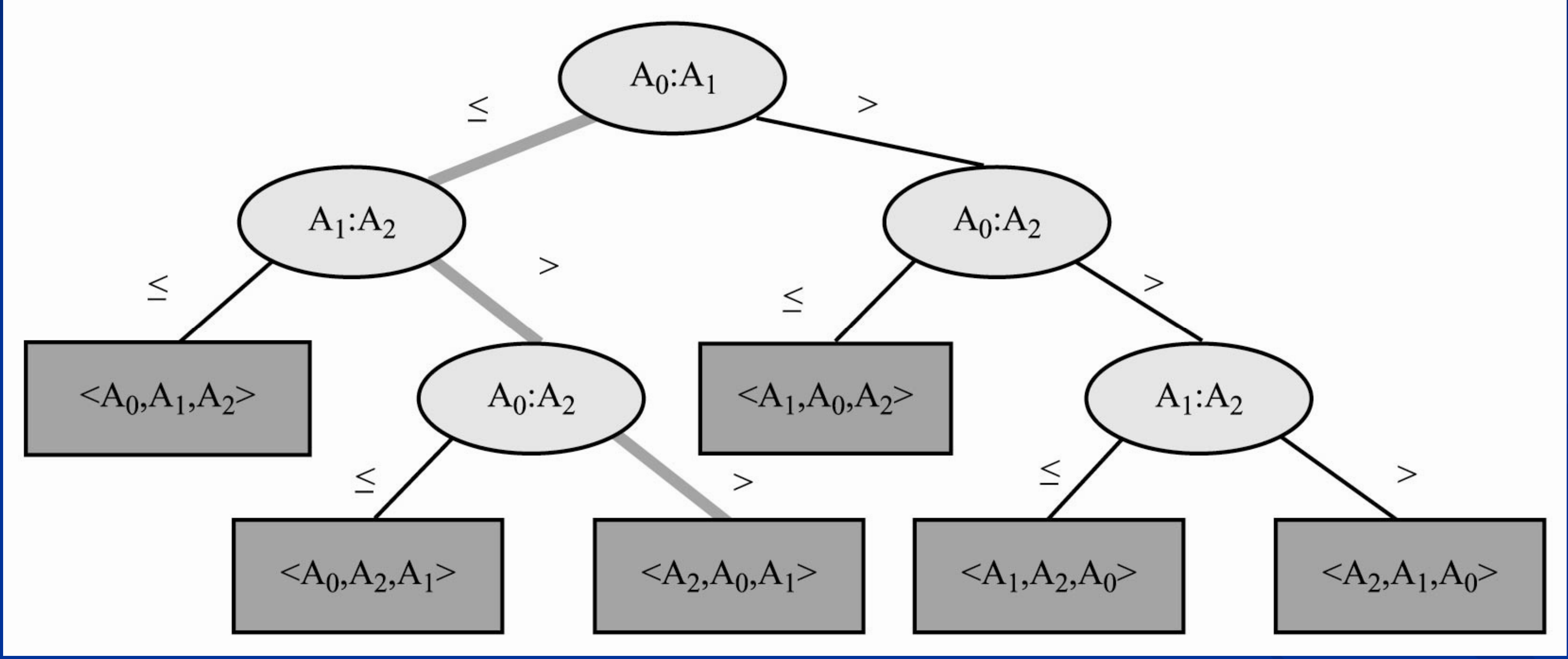
n=500000 0.296 s



比較排序演算法的時間下限

- Comparison Sort
  - 排序演算法所決定的排列順序，只基於輸入元素間的比較
  - 選擇排序、氣泡排序、插入排序、合併排序、快速排序、…
- 時間複雜度的下限 $\Omega(n \log n)$

決定樹模型(decision tree)
比較 $<A_0, A_1, A_2>$



決定樹必須能產生所有排序元素的排列組合
總共 n! 種

比較排序演算法的時間下限

- 決定樹接近完整的二元樹

  (高度 h，共 $2^{h+1}-1$ 個節點)

  $n! \le 2^{h+1}-1 \qquad h \ge \log_2(n!+1)-1$

  $\qquad\qquad\qquad\qquad \ge \log_2(n!)-1$

- 比較排序所需要的時間 T(n) 正比於高度 h

$$T(n)=ch \ge c\,(\log_2 n!-1) \in \Omega(n \log n)$$

$\ln(n!) \sim n\ln(n)-n$

計數排序

- 不屬於比較排序、以空間換取時間

A[0..n-1]

1	5	4	2

數值範圍 0~5 不重複
比較得最大值 k=5

C[0..k]

0	1	2	3	4	5
	1	2		4	5

B[0..n-1]

1	2	4	5

計算時間
T(n)=An+Bk+Cn

$$\in \Theta(n+k)$$

計數排序

- 不屬於比較排序、以空間換取時間

A[0..n-1]

13	11	12	13	12	12

數值範圍 11~13
重複

C[0..k]

0	1	2
1	3	2

計算時間
T(n)=An+Bk+Cn

$$\in \Theta(n+k)$$

B[0..n-1]

11	12	12	12	13	13