Big Picture

asymptotic analysis

時間複雜度

divide and conquer

merge sort

insertion sort

worst case

average case

compared based

quick sort

best case

sorting

linear

index sort

counting sort

partition

Select(A,p,r,i)

order statistic

best case

average case

search, delet, insert

collision

hashing

hash function

probe

演算期中

Insertion Sort

```
INSERTION-SORT(A, n)
for i = 2 to n
    key = A[i]
    // Insert A[i] into the sorted subarray A[1 : i – 1].
    j = i – 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]
        j = j – 1
    A[j + 1] = key
```

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i - 1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .$$

$$\sum_{i=2}^{n} i = \left(\sum_{i=1}^{n} i\right) - 1$$

$$= \frac{n(n+1)}{2} - 1 \quad \text{(by equation (A.2) on page 1141)}$$

worst case = average case = Θ

Divide & Conquer

- Divide
- Conquer
- Combine

Merge sort

```
MERGE(A, p, q, r)
n_L = q - p + 1                  // length of A[p:q]
n_R = r - q                      // length of A[q+1:r]
let L[0:n_L - 1] and R[0:n_R - 1] be new arrays
for i = 0 to n_L - 1             // copy A[p:q] into L[0:n_L - 1]
    L[i] = A[p + i]
for j = 0 to n_R - 1             // copy A[q+1:r] into R[0:n_R - 1]
    R[j] = A[q + j + 1]
i = 0                            // i indexes the smallest remaining element in L
j = 0                            // j indexes the smallest remaining element in R
k = p                            // k indexes the location in A to fill
// As long as each of the arrays L and R contains an unmerged element,
//        copy the smallest unmerged element back into A[p:r].
while i < n_L and j < n_R
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
    k = k + 1
// Having gone through one of L and R entirely, copy the
//        remainder of the other to the end of A[p:r].
while i < n_L
    A[k] = L[i]
    i = i + 1
    k = k + 1
while j < n_R
    A[k] = R[j]
    j = j + 1
    k = k + 1
```

1個 = sorted

```
MERGE-SORT(A, p, r)
if p ≥ r                         // zero or one element?
    return
q = ⌊(p + r)/2⌋                  // midpoint of A[p:r]
MERGE-SORT(A, p, q)              // recursively sort A[p:q]
MERGE-SORT(A, q + 1, r)          // recursively sort A[q+1:r]
// Merge A[p:q] and A[q+1:r] into A[p:r].
MERGE(A, p, q, r)
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n < n_0 , \\ D(n) + aT(n/b) + C(n) & \text{otherwise} . \end{cases}$$

Divide: The divide step just computes the middle of the subarray, which takes constant time. Thus, $D(n) = \Theta(1)$.

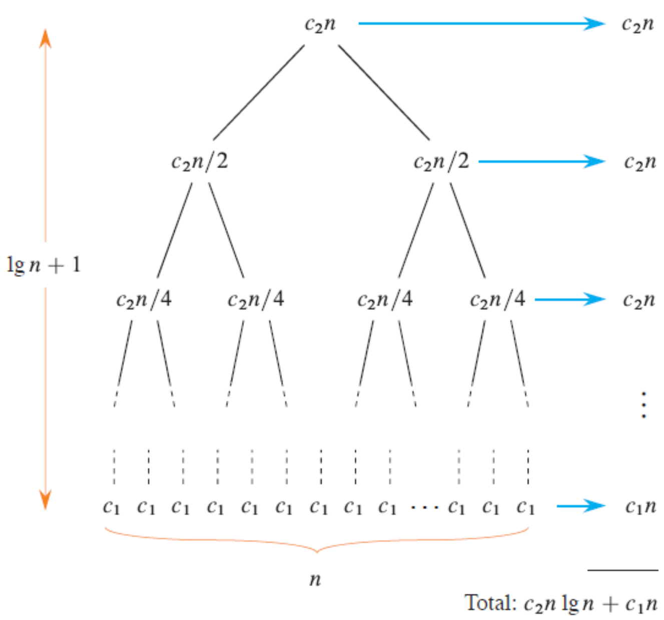
Conquer: Recursively solving two subproblems, each of size $n/2$, contributes $2T(n/2)$ to the running time (ignoring the floors and ceilings, as we discussed).

Combine: Since the MERGE procedure on an n-element subarray takes $\Theta(n)$ time, we have $C(n) = \Theta(n)$.

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

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

Recursion tree



(d)

CH3

Asymptotic

$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.[1]

$\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.

$\Theta(g(n)) = \{f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 . \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty ,$$

floor

modular

polynomial — *polynomially bounded* if $f(n) = O(n^k)$ — $p(n) = \sum_{i=0}^{d} a_i n^i$

exponential — $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$, — $\lim_{n\to\infty} \frac{n^b}{a^n} = 0$, $\rightarrow n^b = o(a^n)$.

logarithm — $f(n) = O(\lg^k n)$ *polylogarithmically bounded* $\rightarrow \lg^b n = o(n^a)$.

factorial — $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$, *Stirling's approximation*,

$$n! = o(n^n) ,$$
$$n! = \omega(2^n) ,$$
$$\lg(n!) = \Theta(n \lg n) ,$$

fibonacci numbers

golden ratio ϕ

$$\phi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots ,$$

negate $\longrightarrow$

$$\hat{\phi} = \frac{1-\sqrt{5}}{2} = -.61803\ldots .$$

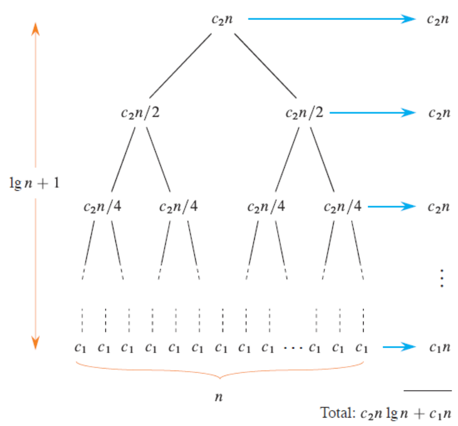
CH4

Divide & Conquer

substitution method

1. Guess the form of the solution using symbolic constants.
2. Use mathematical induction to show that the solution works, and find the constants.

recursion-tree method



(d)

iregular example

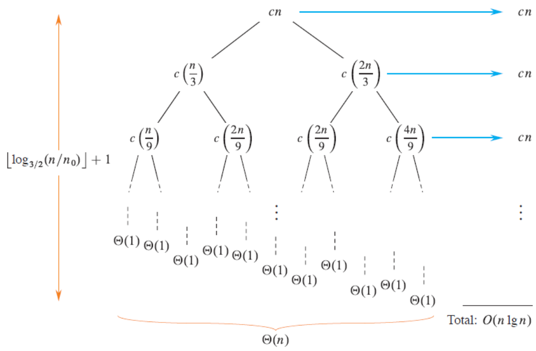


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

master method

$$T(n) = aT(n/b) + f(n),$$

1. If there exists a constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.
2. If there exists a constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.
3. If there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and if $f(n)$ additionally satisfies the ***regularity condition*** $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ∎

watershed function : $n^{\log_b a}$

$$\begin{cases} \Theta(n^{\log_b a}), & n^{\log_b a} > f(n) \\ \Theta(n^{\log_b a} \lg^{k+1} n), & n^{\log_b a} = f(n) \\ \Theta(f(n)), & n^{\log_b a} < f(n) \end{cases}$$

matrix multiplication

recurssion vision

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dh \\ u &= cf + dg \end{aligned}\right\}$$

recursive

8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices

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

\# submatrices

submatrix size

work adding submatrices

$$o(n^3)$$

strassen

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 \quad (= A_{11} \cdot B_{12} - A_{11} \cdot B_{22})\,, \\
P_2 &= S_2 \cdot B_{22} \quad (= A_{11} \cdot B_{22} + A_{12} \cdot B_{22})\,, \\
P_3 &= S_3 \cdot B_{11} \quad (= A_{21} \cdot B_{11} + A_{22} \cdot B_{11})\,, \\
P_4 &= A_{22} \cdot S_4 \quad (= A_{22} \cdot B_{21} - A_{22} \cdot B_{11})\,, \\
P_5 &= S_5 \cdot S_6 \quad (= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22})\,, \\
P_6 &= S_7 \cdot S_8 \quad (= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22})\,, \\
P_7 &= S_9 \cdot S_{10} \quad (= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12})\,.
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

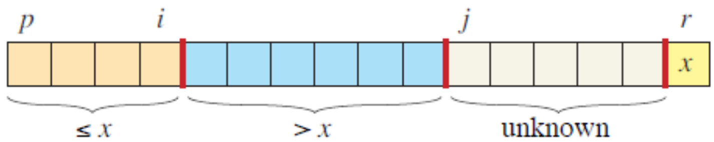
$$\Theta(n^{\lg 7}) \approx O(n^{2.81})$$

Quick Sort

```
QUICKSORT(A, p, r)
if p < r
    // Partition the subarray around the pivot, which ends up in A[q].
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)   // recursively sort the low side
    QUICKSORT(A, q + 1, r)   // recursively sort the high side
```

```
PARTITION(A, p, r)
1  x = A[r]                        // the pivot
2  i = p − 1                       // highest index into the low side
3  for j = p to r − 1              // process each element other than the pivot
4      if A[j] ≤ x                 // does this element belong on the low side?
5          i = i + 1               // index of a new slot in the low side
6          exchange A[i] with A[j] // put this element there
7  exchange A[i + 1] with A[r]     // pivot goes just to the right of the low side
8  return i + 1                    // new index of the pivot
```

loop invariant



Initialization: Prior to the first iteration of the loop, we have $i = p - 1$ and $j = p$. Because no values lie between p and i and no values lie between $i + 1$ and $j - 1$, the first two conditions of the loop invariant are trivially satisfied. The assignment in line 1 satisfies the third condition.

Maintenance: As Figure 7.3 shows, we consider two cases, depending on the outcome of the test in line 4. Figure 7.3(a) shows what happens when $A[j] > x$: the only action in the loop is to increment j. After j has been incremented, the second condition holds for $A[j - 1]$ and all other entries remain unchanged. Figure 7.3(b) shows what happens when $A[j] \leq x$: the loop increments i, swaps $A[i]$ and $A[j]$, and then increments j. Because of the swap, we now have that $A[i] \leq x$, and condition 1 is satisfied. Similarly, we also have that $A[j - 1] > x$, since the item that was swapped into $A[j - 1]$ is, by the loop invariant, greater than x.

Termination: Since the loop makes exactly $r - p$ iterations, it terminates, whereupon $j = r$. At that point, the unexamined subarray $A[j : r - 1]$ is empty, and every entry in the array belongs to one of the other three sets described by the invariant. Thus, the values in the array have been partitioned into three sets: those less than or equal to x (the low side), those greater than x (the high side), and a singleton set containing x (the pivot).

Worst-case partitioning

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) . \end{aligned}$$

$$\Theta(n^2)$$

Best-case partitioning

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

$$O(n \lg n)$$

Balanced partitioning

$$T(n) = T(9n/10) + T(n/10) + \Theta(n)$$

$$\log_{10/9} n = \Theta(\lg n)$$

$$O(n \lg n)$$

randonmized partition

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Worst-case analysis

$$T(n) = \max\{T(q) + T(n-1-q) : 0 \le q \le n-1\} + \Theta(n) ,$$

$$\begin{aligned} T(n) &\le \max\{cq^2 + c(n-1-q)^2 : 0 \le q \le n-1\} + \Theta(n) \\ &= c \cdot \max\{q^2 + (n-1-q)^2 : 0 \le q \le n-1\} + \Theta(n) . \end{aligned}$$

$$\begin{aligned} q^2 + (n-1-q)^2 &= q^2 + (n-1)^2 - 2q(n-1) + q^2 \\ &= (n-1)^2 + 2q(q-(n-1)) \\ &\le (n-1)^2 \end{aligned}$$

$$\begin{aligned} T(n) &\le c(n-1)^2 + \Theta(n) \\ &\le cn^2 - c(2n-1) + \Theta(n) \\ &\le cn^2 , \end{aligned}$$

Expected running time $O(n \lg n)$

Theorem 7.4
The expected running time of RANDOMIZED-QUICKSORT on an input of n distinct elements is $O(n \lg n)$.

Proof The analysis uses indicator random variables (see Section 5.2). Let the n distinct elements be $z_1 < z_2 < \cdots < z_n$, and for $1 \le i < j \le n$, define the indicator random variable $X_{ij} = \mathrm{I}\{z_i \text{ is compared with } z_j\}$. From Lemma 7.2, each pair is compared at most once, and so we can express X as follows:

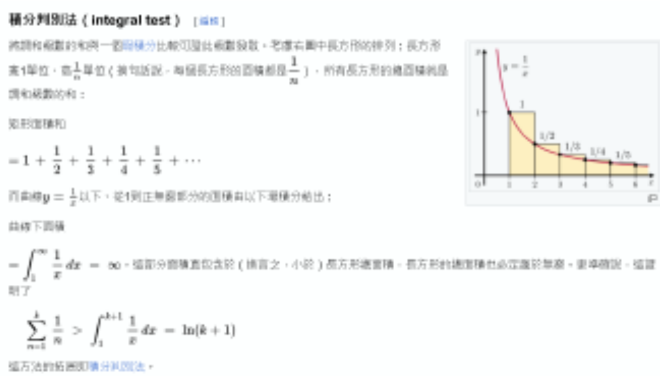
$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} .$$

By taking expectations of both sides and using linearity of expectation (equation (C.24) on page 1192) and Lemma 5.1 on page 130, we obtain

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] && \text{(by linearity of expectation)} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared with } z_j\} && \text{(by Lemma 5.1)} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} && \text{(by Lemma 7.3)} .
\end{aligned}$$

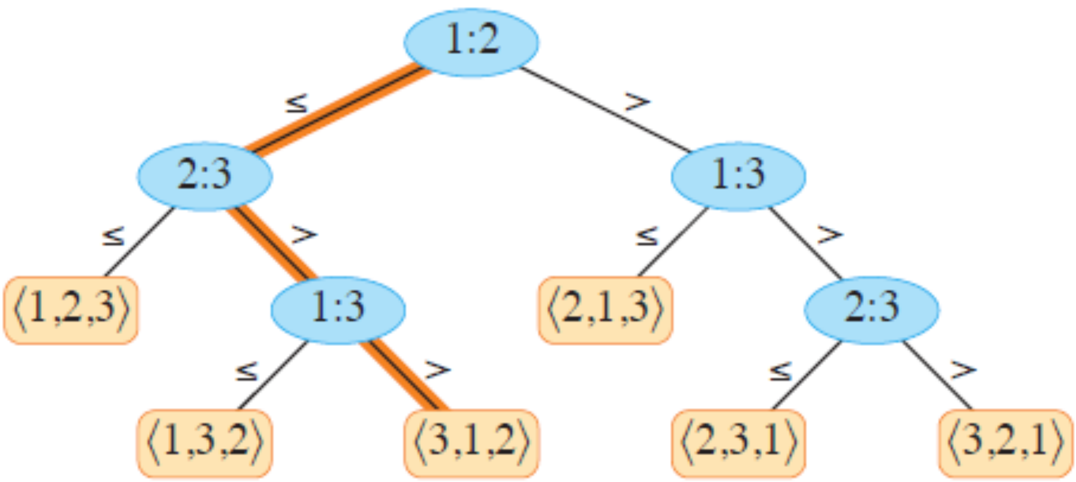
We can evaluate this sum using a change of variables ($k = j - i$) and the bound on the harmonic series in equation (A.9) on page 1142:

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n) .
\end{aligned}$$



Sorting in linear time

decision-tree



Theorem 8.1
Any comparison sort algorithm requires $\Omega(n \lg n)$ comparisons in the worst case.

Proof From the preceding discussion, it suffices to determine the height of a decision tree in which each permutation appears as a reachable leaf. Consider a decision tree of height h with l reachable leaves corresponding to a comparison sort on n elements. Because each of the $n!$ permutations of the input appears as one or more leaves, we have $n! \le l$. Since a binary tree of height h has no more than 2^h leaves, we have

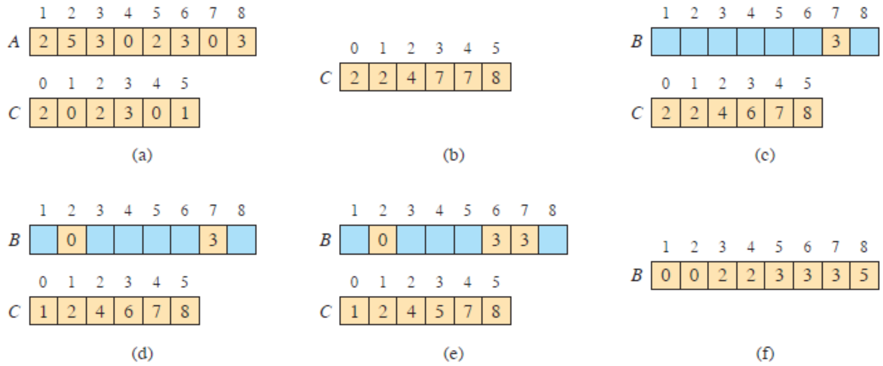
$$n! \le l \le 2^h ,$$

which, by taking logarithms, implies

$$\begin{aligned} h &\ge \lg(n!) && \text{(since the lg function is monotonically increasing)} \\ &= \Omega(n \lg n) && \text{(by equation (3.28) on page 67)} . \end{aligned}$$

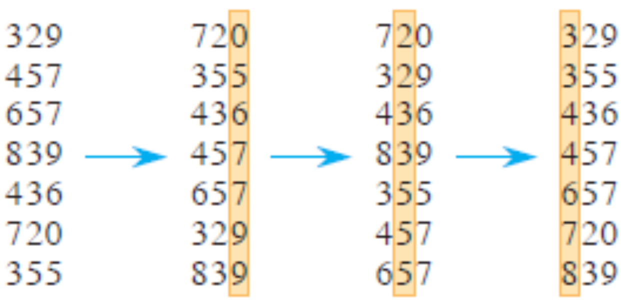
■

counting sort




```
COUNTING-SORT(A, n, k)
let B[1 : n] and C[0 : k] be new arrays
for i = 0 to k
    C[i] = 0
for j = 1 to n
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i.
// Copy A to B, starting from the end of A.
for j = n downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] − 1    // to handle duplicate values
return B
```

radix sort




```
RADIX-SORT(A, n, d)
for i = 1 to d
    use a stable sort to sort array A[1 : n] on digit i
```

CH9

Median & Order Statistic

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]        // 1 ≤ i ≤ r − p + 1 when p == r means that i = 1
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1          → k 為 subarray 的長度
if i == k
    return A[q]        // the pivot value is the answer
elseif i < k
    return RANDOMIZED-SELECT(A, p, q − 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

p k q-1 q+1 r

worst-case $\Theta(n^2)$

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

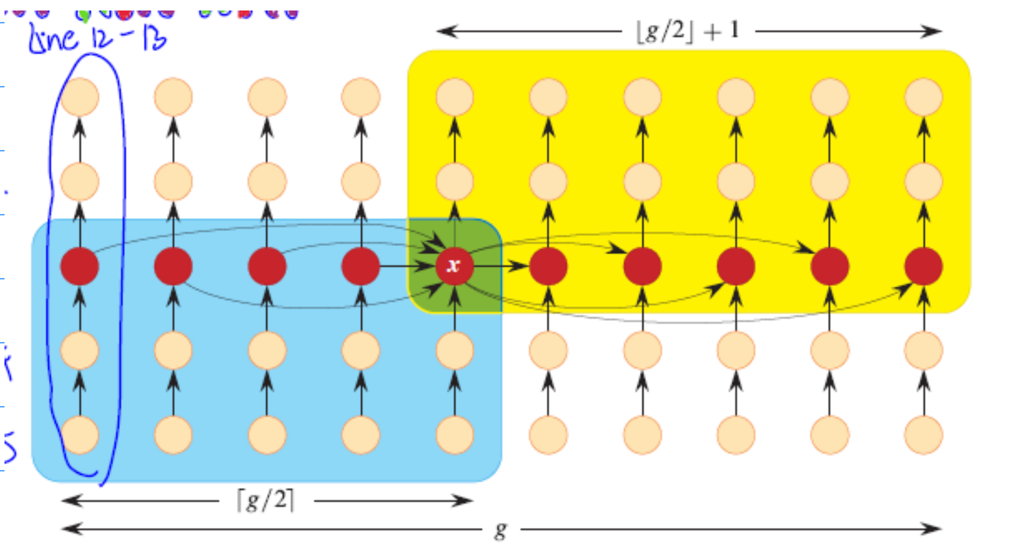
Select

SELECT guarantees a good
split by choosing a provably good pivot

```
SELECT(A, p, r, i)                                           0~4
1   while (r − p + 1) mod 5 ≠ 0
2       for j = p + 1 to r               // put the minimum into A[p]
3           if A[p] > A[j]
4               exchange A[p] with A[j]
5       // If we want the minimum of A[p : r], we're done.
6       if i == 1
7           return A[p]
8       // Otherwise, we want the (i − 1)st element of A[p + 1 : r].
9       p = p + 1
10      i = i − 1
11  g = (r − p + 1)/5                    // number of 5-element groups
12  for j = p to p + g − 1               // sort each group
13      sort ⟨A[j], A[j + g], A[j + 2g], A[j + 3g], A[j + 4g]⟩ in place
14  // All group medians now lie in the middle fifth of A[p : r].
15  // Find the pivot x recursively as the median of the group medians.
16  x = SELECT(A, p + 2g, p + 3g − 1, ⌈g/2⌉)     median of median
17  q = PARTITION-AROUND(A, p, r, x)     // partition around the pivot
18  // The rest is just like lines 3–9 of RANDOMIZED-SELECT.
19  k = q − p + 1
20  if i == k
21      return A[q]                      // the pivot value is the answer
22  elseif i < k
23      return SELECT(A, p, q − 1, i)
24  else return SELECT(A, q + 1, r, i − k)
```

從 (p+1) 為 base，找 (i-1) 大的

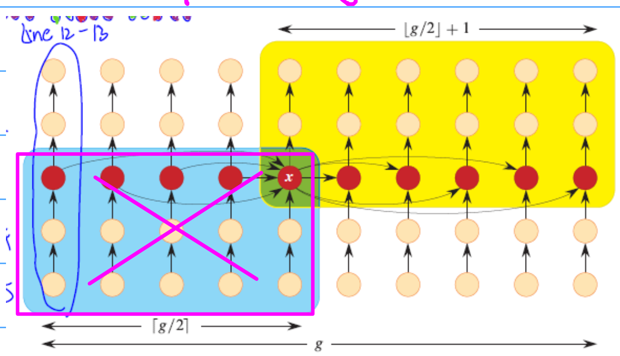
maybe use in 9.1-1 ?



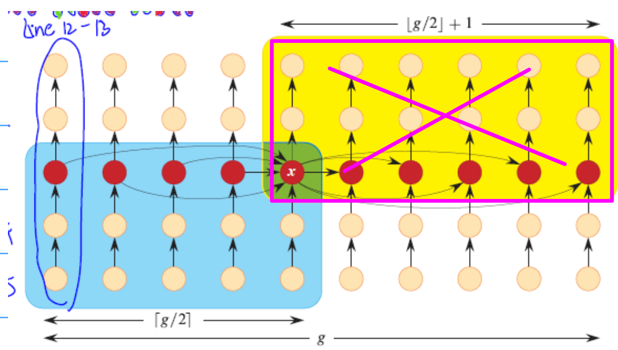
low & high side

$$5g - 3g/2 = 7g/2 \le 7n/10$$

pick high



pick low



可以捨棄至少 low region ($\ge 3g/2$)

可以捨棄至少 high region ($\ge 3g/2$)

∵ white region 都可能落在 high & low

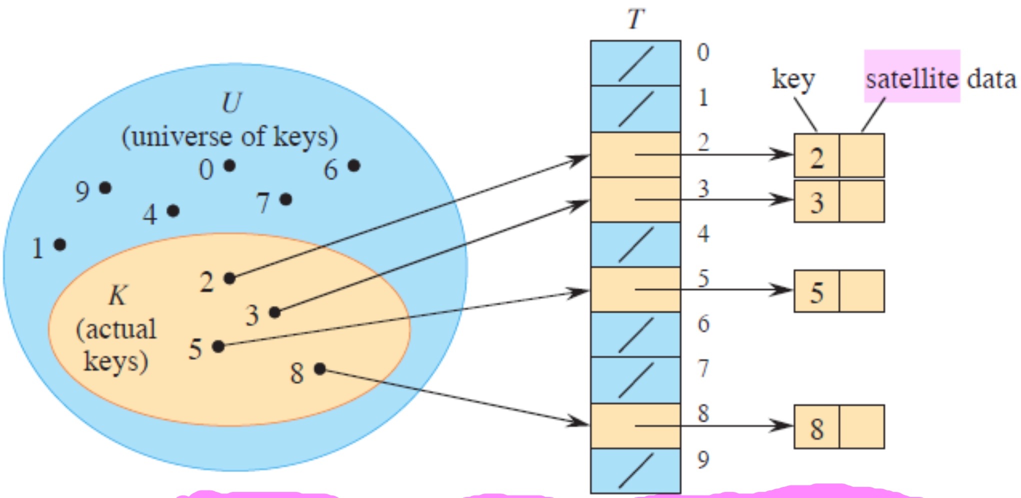
∴ 不能捨棄

$$T(n) \le T(n/5) + T(7n/10) + \Theta(n) .$$

$$\begin{aligned} T(n) &\le c(n/5) + c(7n/10) + \Theta(n) \\ &\le 9cn/10 + \Theta(n) \\ &= cn - cn/10 + \Theta(n) \\ &\le cn \end{aligned}$$

Hash

Direct Address



avoiding collisions altogether is impossible !

DIRECT-ADDRESS-SEARCH(T, k)
1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)
1 $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)
1 $T[x.key] = \text{NIL}$

→ $O(1)$

independent uniform hash function

每個 elements 都被隨機抽中

→ 每個 slot 能平均分配到接近相同的個數

each key is equally likely to hash to any one of the m slots

Analysis

- load factor $\alpha = \frac{n}{m}$
- search
  - successful (thm 11.1)

    $\Theta(1+\alpha)$ on average
  - unsuccessful (thm 11.2)

    $\Theta(1+\alpha)$ on average
- O(1) 條件

  $n = O(m)$

  $\alpha = n/m = O(m)/m = O(1)$
- 當是 ***independent uniform*** 時

  two distinct keys collide $\longrightarrow$ probability $1/m$

Hash function

The division method

$$h(k) = k \bmod m\ .$$

The multiplication method

$$h(k) = \lfloor m\,(kA \bmod 1) \rfloor$$

The multiply-shift method

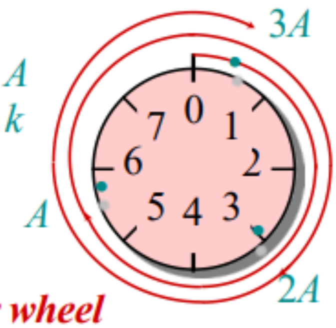
$$h_a(k) = (ka \bmod 2^w) \ggg (w - \ell)$$

Multiplication method example

$$h(k) = (A \cdot k \bmod 2^w) \text{ rsh } (w - r)$$

Suppose that $m = 8 = 2^3$ and that our computer has $w = 7$-bit words:

$$\begin{array}{rl} 1011001 & = A \\ \times\quad 1101011 & = k \\ \hline 1001010\underbrace{011}_{h(k)}0011 & \end{array}$$



Modular wheel

Double hashing

probe 的次數

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m$$

Linear probing

$$h(k,i) = (h_1(k) + i) \bmod m$$

$h_2(k) = 1$

Advantage :

可以更平均的 hash 到每個 slot !

Open address

with no satellite information

$$h : U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}$$

```
HASH-INSERT(T, k)
i = 0
repeat
    q = h(k, i)
    if T[q] == NIL
        T[q] = k
        return q
    else i = i + 1
until i == m
error "hash table overflow"
```

```
HASH-SEARCH(T, k)
i = 0
repeat
    q = h(k, i)
    if T[q] == k
        return q
    i = i + 1
until T[q] == NIL or i == m
return NIL
```

$m!$ permutations of $\langle 0, 1, \ldots, m-1 \rangle$

Analysis

Search

- successful (thm 11.6)

$$\le \frac{1}{1-\alpha}$$

- unsuccessful (thm 11.7)

$$\le \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

Why ?

youtube:hash介紹

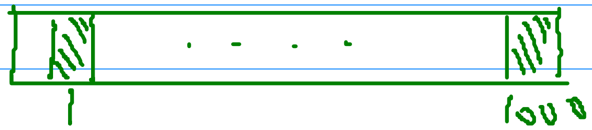
https://www.youtube.com/watch?v=fT3uaSleptI&list=PLyyZcZrmGuHFLOltniDCj5Jbgm_dyfh6T&index=7&ab_channel=%E6%86%B6%E7%B4%94%E6%99%83%E6%99%83

Hash

(1) 為何要有 hash ?

ex. 只有2個 key : $k_1=1$, $k_2=1000$

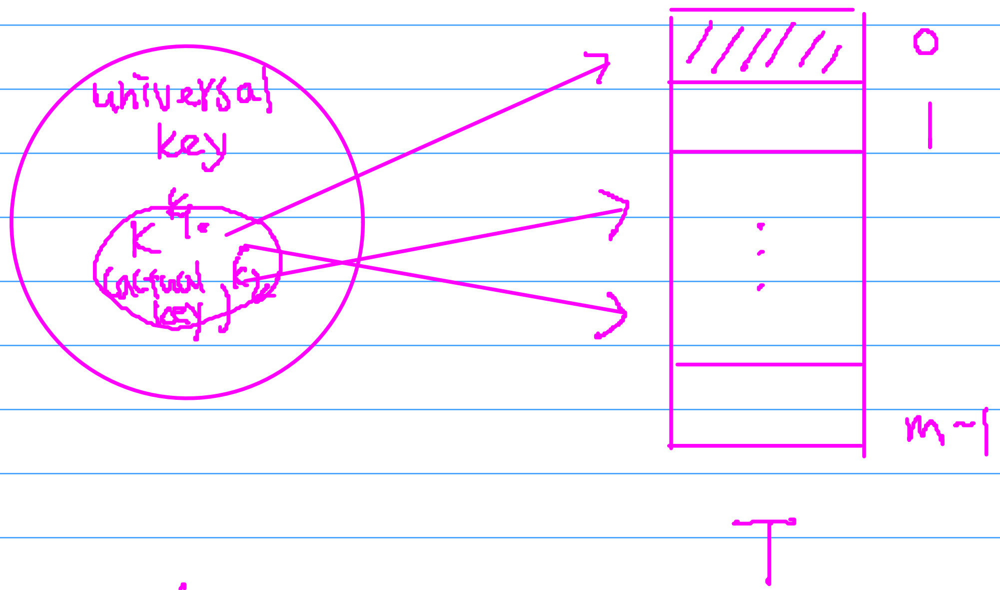
。without hash table (using array)



只有 key=1, 跟 key=1000 是有用的, 其他 998 個都沒用到

⇒ 浪費

。with hash table

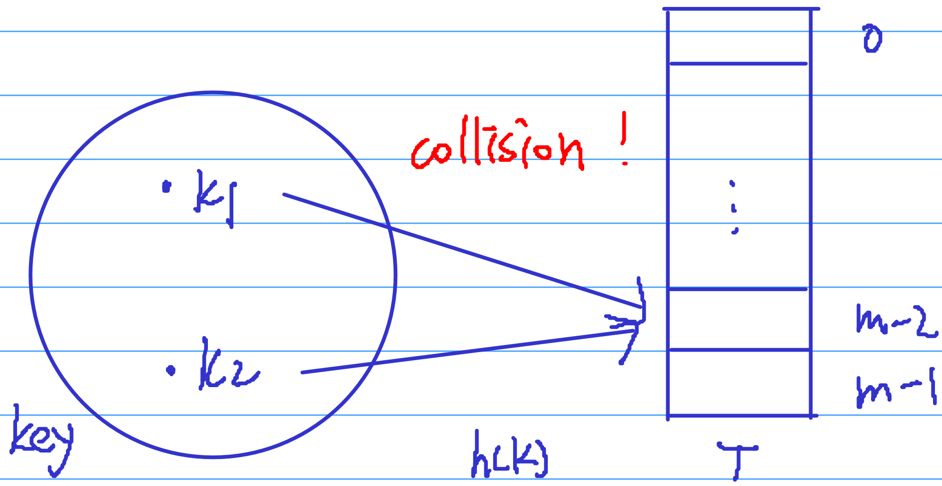


這樣一來, 只要開一個大小為 m 的 hash table 就好了!

跟 actual key 一樣多?

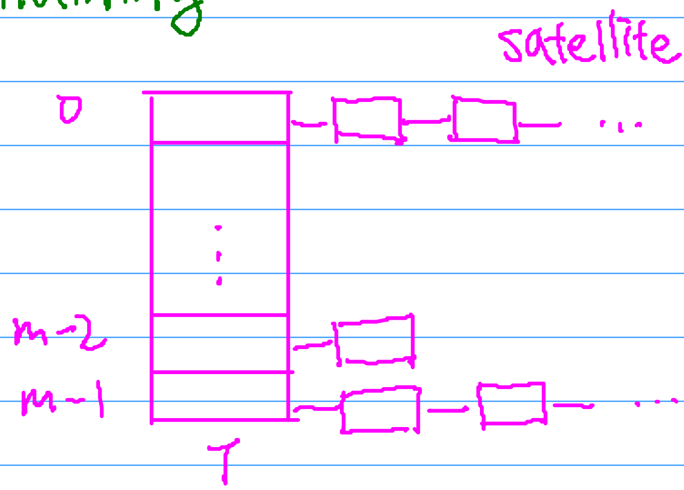
(2) hash 產生的 problem

collision (different keys point to the same slot)



Solution:

(1) chaining



- Operations

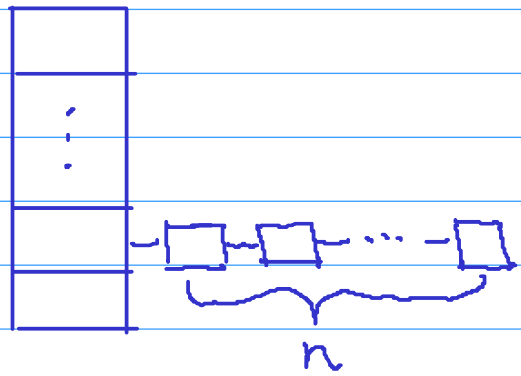
  Insert : $O(1)$

  Delete : $O(1)$

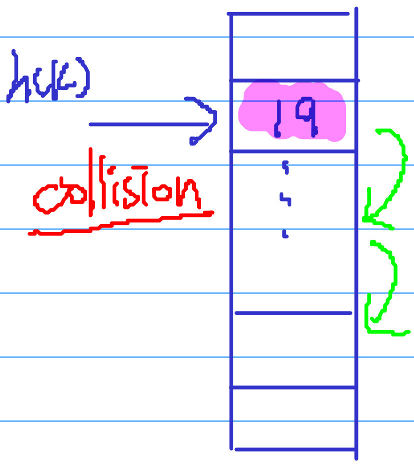
  Search : $O(1)$

- problem – search time complexity
  - worst case : $\Theta(n)$
  - average case : $\Theta(1+\alpha)$, $\alpha = \frac{n}{m}$ load factor

bad :



(2) open addressing



probe : 往下找可以

linear probing
quadratic probing
double hashing