Unit 4
(Other) Fast Sorting and Selection Algorithms

T.H. Cormen et al., "Introduction to Algorithms", 3rd ed., Chapters 7-9.

R. Sedgewick, "Algorithms in C++", Chapters 9,10.

Quicksort

☛ **類似 mergesort 用 divide-and-conquer 的技巧**

a[ ]

從 a[ ] 選一數 *v*

a[ ] ≤ *v*	*v*	a[ ] ≥ *v*

Quicksort 程式 (p.171)

```
quicksort(a[ ], ℓ, r) {
    if (ℓ < r) {
        i  =  partition(a, ℓ, r);
       quicksort(a, ℓ, i-1);   /* a[ℓ..i-1] ≤ a[i] */
       quicksort(a, i+1, r);  /* a[i+1..r] ≥ a[i] */
    }
}
```

ℓ r

a[ ]

Partition: 從 a[ ] 選一數 *v*

a[ ] ≤ *v*	*v*	a[ ] ≥ *v*

ℓ i r

Partition Method 1

```
v = a[r] ;  /* pivot (partition) element 為  a[r]  */
i =  ℓ – 1;    j = r;
for (  ;  ;  ) {
       while ( a[ ++i ] < v )   ;
       while ( a[ ––j ] > v )   ;
       if( i  >=  j ) break;
       swap(a, i, j);   }
swap(a, i, r);
return i;
```

 # comparisons ≤ *n* + 1

ℓ i j r

a[ ]	a[ ] ≤ v				a[ ] ≥ v	v

需在 a[0] 放一 sentinel key –∞

Partition Method 2 (p.171)

```
v = a[r] ;  /* pivot (partition) element 為  a[r]  */
i =  ℓ – 1;
for (j = ℓ; j < r; j++)
        if ( a[ j ] <= v ) swap(a, ++i, j);
swap(a, ++i, r);
return i;
```

 # comparisons = *n* – 1

ℓ i j r

a[ ]	a[ ] ≤ v	a[ ] > v			v

Analysis of Quicksort

☛ **Best case :**

$T(n) = 2T(n/2) + n - 1 \approx n\lg n$

☛ **Worst case :**

$T(n) = T(n-1) + n - 1 \approx n^2/2$

☛ **Average case :**

Assume that all the possible inputs of n elements are ***uniformly distributed***.

Then, we have:

Average Case Analysis of Quicksort

☛ v : pivot element

a[ ]

v 為 a[ ]中第 k 小元素的機率為 $1/n$, $1 \le k \le n$

a[ ] $\le v$	v	a[ ] $\ge v$
[k–1 個元素]		[n–k 個元素]

$$T(n) = n-1+\frac{1}{n}\sum_{k=1}^{n}(T(k-1)+T(n-k)) \approx 1.38n\lg n$$

Quicksort 的改進（增快約25至30%）

1. Removing recursion

- 在 worst case 時所需 stack size 可能為 $O(n)$.
- 分成兩個 subfiles 後, 先 sort 小的再把大的資料放在 stack. 可證得所需 stack size 為 $O(\lg n)$.

Quicksort 的改進（續1）

2. Small subfiles.

- 當 subfile 夠小時, 用 elementary sort.
- 也可在 subfile 夠小時, 不做任何處理, 最後再做一次 insertion sort.

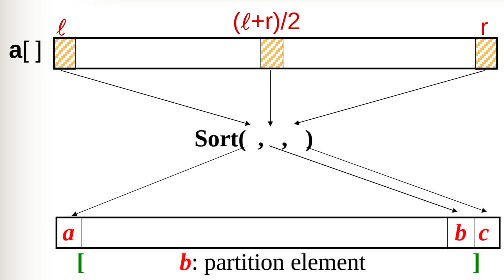
Quicksort 的改進（續2）

3. Median-of-three partitioning (M-o-3).

☛選 pivot element 不用固定，可用random 法（仍有缺點）或 M-o-3:

- Sort a[ℓ],a[(ℓ +r)/2],a[r],
- swap(a,(ℓ +r)/2,r-1),
- 在 a[ℓ–1...r–2] 對 a[r–1] 做 partition.

Median-of-three partitioning (M-o-3)



好處：worst case 不容易發生，且不用在 a[0] 放 sentinel (指 partition method 1)

A Randomized Quicksort (p.179)

```
R_quicksort(a[ ], ℓ, r) {
  if (ℓ < r)  {
    q  =  random(ℓ, r);
    swap(a, q, r);
     i  =  partition(a, ℓ, r);
    R_quicksort(a, ℓ, i-1);   /* a[ℓ..i-1] ≤ a[i] */
    R_quicksort(a, i+1, r);  /* a[i+1..r] ≥ a[i] */
  }
}
```

Analysis of Randomized Quicksort (1)

- Each execution of randomized quicksort can be viewed as a ***random experiment***.
- Let X be a ***random variable*** that denotes the number of comparisons performed by the algorithm. We want to compute $E(X)$.
- For ease of analysis, we assume the elements of array a[ ] are distinct and rename them as: $z_1, z_2, \dots, z_n$ with $z_1 < z_2 < \dots < z_n$.

Analysis of Randomized Quicksort (2)

- We define a group of random variables as:
  X_{ij} = 1 if z_i is compared to z_j
  = 0 otherwise.
- Then $X = \Sigma_{1\le i<j\le n} X_{ij}$.
- Hence $E(X) = E[\,\Sigma_{1\le i<j\le n} X_{ij}\,] = \Sigma_{1\le i<j\le n} E(X_{ij})$.
- Note that $E(X_{ij})$ = Pr{z_i is compared to z_j} = $p_{i,j}$
- For example, $p_{i,i+1}$ = ? $p_{1,n}$ = ?

Analysis of Randomized Quicksort (3)

- Let $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$
- Pr{z_i is compared to z_j}
  = Pr{z_i or z_j is first pivot chosen from Z_{ij}}
  = Pr{z_i is first pivot chosen from Z_{ij}}
  + Pr{z_j is first pivot chosen from Z_{ij}}
  $= 1/(j-i+1) + 1/(j-i+1) = 2/(j-i+1)$
- Hence $E(X) = \Sigma_{1\le i<j\le n} 2/(j-i+1)$
  $< 2n(1+ 1/2 + 1/3 + \dots + 1/n) < 2n \ln n$
  $\approx 1.38 n \lg n$.

Sorting Algorithms

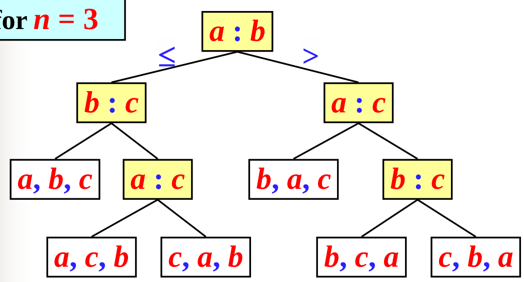
Algorithms	Time colplexity
Selection Sort	$\Theta(n^2)$
Insertion Sort	$\Theta(n^2)$
Bubble Sort	$\Theta(n^2)$
Batcher's Sort	$\Theta(n \lg^2 n)$
Quicksort	$\Theta(n \lg n)$ (average case)
Mergesort	$\Theta(n \lg n)$ Optimal ?
Heapsort	$\Theta(n \lg n)$

Comparison Sorts

- ***Comparison sorts*** : sorting algorithms based only on comparisons between the input elements.
- Sorting algorithms studied thus far are comparison sorts.
- We will show that $\Omega(n \lg n)$ is a lower bound for all comparison sorts.
- Hence, heapsort and mergesort are asymptotically optimal comparison sorts (for the worst case).

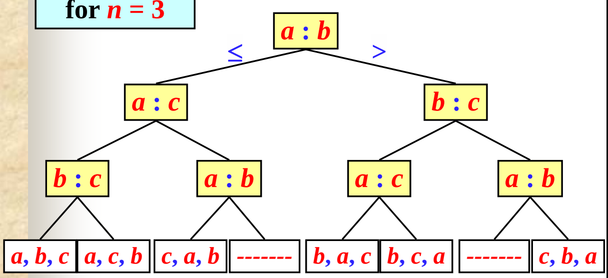
Decision Trees for Insertion Sorts

An example for $n = 3$



Decision Trees for Selection Sorts

An example for n = 3



Decision Trees for Comparison Sorts

- Every comparison sort that sorts an instance with n elements corresponds to a decision tree with the number of leaves ≥ __.
- Any such a tree has height ≥ ______.
- $\lg n! = \Theta(n \lg n)$ (see Unit 2 or 3)
- So $\Omega(n \lg n)$ is a lower bound for all comparison sorts.

A Lower Bound for the Average Case

- Using the decision tree model, we can show that $\Omega(n\lg n)$ is also a lower bound for all comparison sorts in the average case. (Computing the average height of a decision tree with each leaf having the same probability.)
- Hence, heapsort, mergesort, and quicksot are asymptotically optimal comparison sorts (for the average case).

Sorts Not Based on Comparisons

- How to break the $\Omega(n\lg n)$ barrier for sorting problems?
- Such a barrier-breaking sort must not be a comparison sort.
- Consider a special case of sorting n elements with key range **0..*n***.
- Examples: counting sort, radix sort, bucket sort. (All of them are ***special purpose*** and in general not in place.)

Counting Sort (例)

A	B	B	A	C	A	D	A	B	B	A	D	D	A

有 6 A 4 B 1 C 3 D

累計得 6 10 11 14

					6				10	11			14

				5	A_6				10	11			14

				5	A_6				10	11		13	D_3

				5	A_6				10	11	12	D_2	D_3

Counting Sort (程式)

```
/* input: a[ ],  output: b[ ],  working space: c[ ] */

for (j = 0 ; j <= k; j++)  c[j] = 0;
for (i = 1 ; i <= n; i++)  c[a[i]]++;
for (j = 1 ; j <= k; j++)  c[j] += c[j-1];
for (i = n ; i >= 1; i−−)  b[c[a[i]]−−] = a[i];
```

Counting Sort (討論)

- 假設 key range 為 **0 ... *k***, ***k* = *O*(*n*)**.
- 執行時間為 ***O*(*n*)**.
- 不為 in place.
- 為 *special purpose* sort.
- 為 stable sort.

Radix Sort

- 可看成是 counting sort 的推廣
- 類似counting sort, 適用 radix sort 的排序問題其 key range 不能太大 (即所謂 special purpose sort)
- Radix sort 將 key 看成 ***M*** 進位之整數：
  key = $d_1, d_2, \ldots, d_b$, $0 \leq d_i < M$
  分 ***b*** 個 passes 進行, 每個 pass 只處理一位數.
- 處理的方向可以是由左到右或是由右到左, 以下例子我們只考慮右到左.

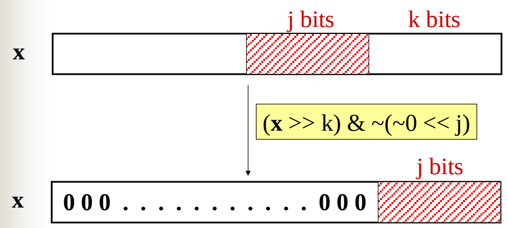
Radix Sort (右到左例子：*M* = 10)

428	790	412	126
494	412	624	227
258	963	724	258
412	494	126	412
963	624	227	428
790	724	428	446
624	775	938	494
775	446	446	624
938	126	258	689
227	227	963	724
446	428	775	775
126	258	689	790
724	938	790	938
689	689	494	963

但每一個 pass 所用的排序法需為 stable

Radix sort 中處理位數的技巧

- 由 word **x** 中抓某 j 個 bits.



Radix Sort (討論)

- 假設要排序 ***n*** 個 ***b***-bit 的數, 若 ***M* = 2****^{m}, radix sort 執行時間固定為 ***O*(*nb*/*m*)**, 經驗值為 ***b*/*m* = 4**. 可視為一種 linear-time sort, 但須大量額外 working space.
- 當 ***n* << 2****^{b}, 不適合用 radix sort.

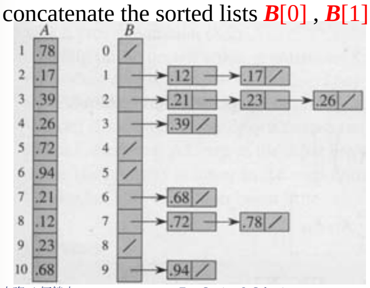
Bucket Sort

- Sort ***n*** elements with keys drawn uniformly from a known range.
- The range can be converted into the interval [0,1).
- Basic idea: Divide the interval into equal-sized subintervals, or ***buckets***, distribute the input into the buckets, sort elements in each bucket, and then concatenate the sorted lists in order.

Bucket Sort (pseudo-code & example)

1. **for** $i \leftarrow 1$ **to** n
2. **do** insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$
3. concatenate the sorted lists $B[0]$, $B[1]$,…, $B[n-1]$

Do insertion sort



Bucket Sort (Analysis 1)

- **Best case :** $T(n)$ = _____.
- **Worst case :** $T(n)$ = _____.
- **Average case :** $T(n) = \Theta(n) + \Sigma_{0 \le i < n} O(n_i^2)$

$$E[T(n)] = \Theta(n) + E[\ \Sigma_{0 \le i < n} O(n_i^2)\ ]$$
$$= \Theta(n) + \Sigma_{0 \le i < n} E[O(n_i^2)]$$
$$= \Theta(n) + \Sigma_{0 \le i < n} O(E[n_i^2])$$
$$= \Theta(n) + \Sigma_{0 \le i < n} O(1)$$
$$= \Theta(n) + \Theta(n) = \Theta(n)$$

Bucket Sort (Analysis 2)

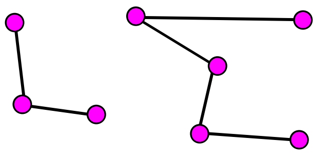
- Here n_i is the random variable denoting the number of elements placed in bucket $B[i]$.
- Actually, n_i is a __________ random variable.
- Hence, $E(n_i) = n \cdot 1/n = 1$, $V(n_i) = n \cdot 1/n \cdot (1-1/n)$.
- By $V(n_i) = E(n_i^2) - [E(n_i)]^2 = 1-1/n$,
- We have: $E(n_i^2) = 2-1/n = O(1)$.

The Selection Problem

- Given a set S of n numbers and an integer i, find the i-th smallest number in S.
- The problems of finding the ***minimum***, ***maximum***, and ***median*** of S are special cases of this problem.
- The selection problem can be solved by first sorting the numbers in S. However, there are faster algorithms.

Minimum and Maximum

- Finding the minimum (and maximum as well) can be accomplished with $n-1$ comparisons.
- Is this the best we can do?
- Yes (for comparison-based algorithms).



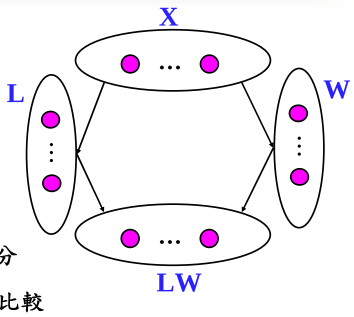
Simultaneous Minimum and Maximum

- A naïve algorithm uses $n-1 + n-2 = 2n-3$ comparisons of keys.
- A divide-and-conquer algorithm (or the algorithm described in the text book) can solve this problem using $\lceil 3n/2 \rceil - 2$ comparisons.
- Is this an ***optimal*** algorithm?
- Yes.

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2$,
$T(1) = 0$, $T(2) = 1$.

Adversary Arguments

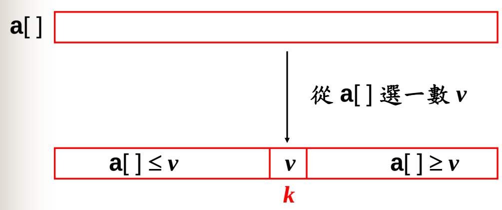
- X ↘ L, W 得 **0.5** 分
- L, W ↘ LW 得 **1** 分
- 共 ________分可拿
- 比較一次最多拿__分
- ∴需 __________次比較



Finding the Second Smallest Key

- The smallest key must be found first.
- A naïve algorithm uses $n-1 + n-2 = 2n-3$ comparisons of keys.
- Applying the tournament method, only ____________ comparisons of keys are used.

Find the ith smallest in $\Theta(n)$ expected time



If $i = k$, o.k.

If $i < k$, search left part;

If $i > k$, search right part, and $i \leftarrow i - k$

Analysis of Quicksort Based Selection

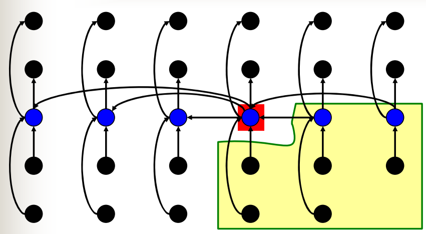
- Best case : $T(n)$ = _____.
- Worst case : $T(n)$ = _____.
- Expected #comparisons $T(n)$: $T(1) = 0$

  $T(n) \le \left[\sum_{1\le k\le n} T(\max(k-1, n-k))\right] / n + n-1$

  $T(n) \le 2\sum_{\lfloor n/2\rfloor \le k<n} T(k)/n + n-1$

  $T(n) \approx 4n$ (can be proved by induction on n)

Selection in Worst-Case Linear Time



- 時間複雜度：
  $T(n)=T(\lceil n/5\rceil)+T(7n/10+6)+an+O(1)$, for $n > 70$
  $\Rightarrow T(n) \approx 20an$ (can be proved by induction on n)