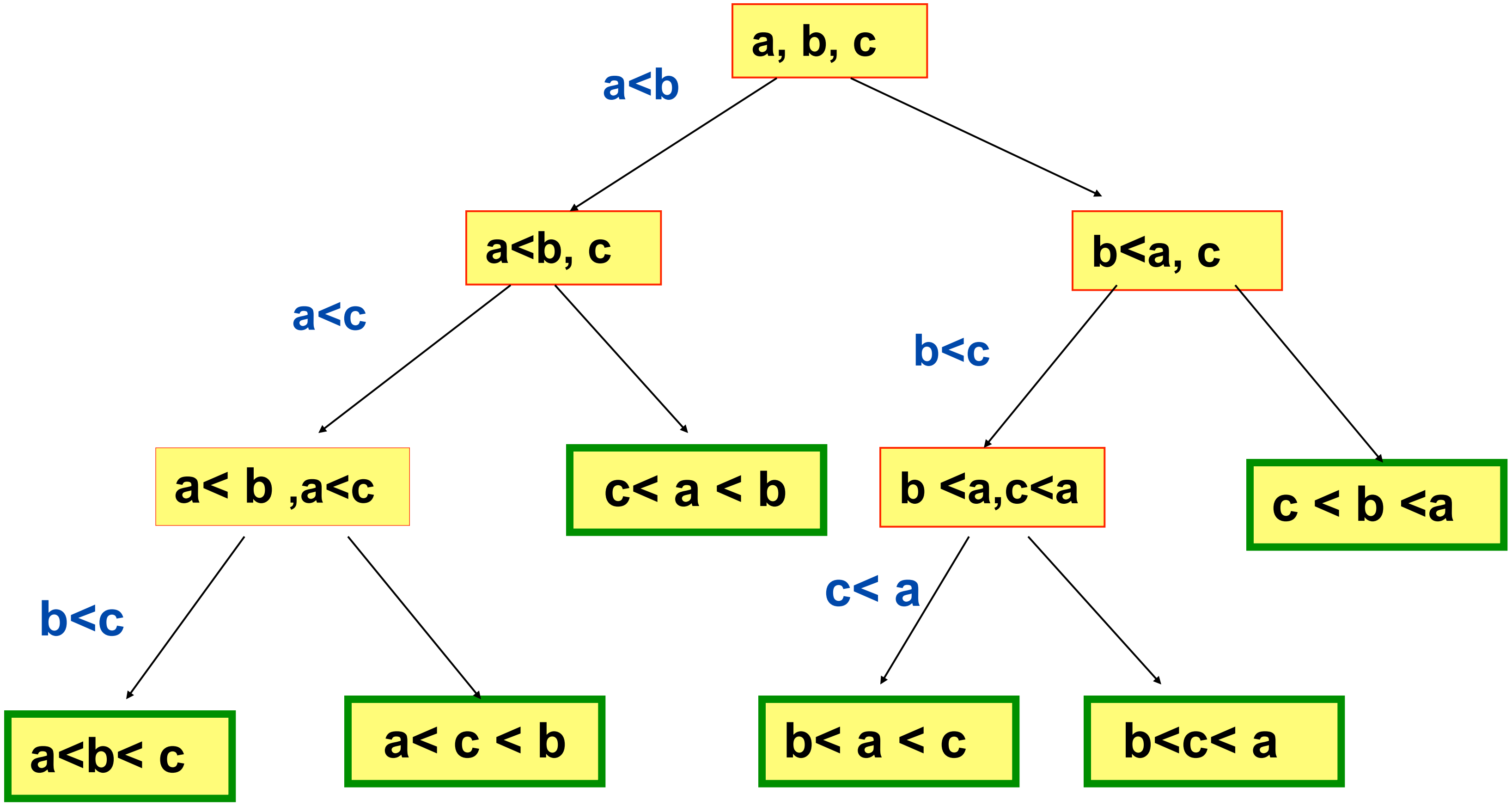
Searching and Sorting

- Searching
  - Linear O(N)
  - bisection O(log(N))
  - dictionary O(log(log(N
- Sorting
  - Insertion, bubble, selection O(N^2) CRLS: 2.1
  - Merge, Quick, Heap O(N log (N)) CRLS: 2.2,
    Proof: $\Omega(N^2)$ near neighbor exchange
- Proof: Ω(N log(N)) Comparison search
- Median (or k quick selection) Problem CLRS: 9
  - Bin (Count), Radix, Bucket O(N) CLRS: 8
-

Decision Tree

Proof of $\Omega(N\log(N))$



Binary decisions: 3! = 6 possible outcomes. Longest path: log(3!)

Lower Bound Theorem for Camparision Sort

Proof: Compute the maximum depth D of decision tree?

- Need N! leaves to get all possible outcomes of a sorting routine.
- Each level at most doubles: $1 * 2 * 4 * \cdots * 2^D$
- Consequently for D levels: $N! \leq 2^D \Rightarrow D \geq log_2(N!)$

$$\Rightarrow T(N) = \Omega(D) = \Omega(\log_2(N!)) = \Omega(N \log_2(N))$$

$$Information = \log_2(N!) \cong N \log_2(N)$$

Number of bits to encode any (initial) state is information (- Entropy)

Searching: *"Why Sort at All?"*

- int a[0], a[1],a[2],a[3],.... a[m],.... a[2],a[N-1]

Three Algorithms:

- *Linear Search* → O(N)

(after Sorting)

- *Bisection Search* → O(log(N)).
- *Dictionary Search* → O(log[log[N]])

Bisection Search of Sorted List

- int a[0], a[1],a[2],a[3],.... a[m],.... a[N-2],a[N-1]

i j

```
i= 0; j= N-1; m = N/2
 while(b!=a[m] && i!=j ){
            if(b>a[m])    i = m+1;
           if(b<a[m])    j = m-1;
            m =   (j-i)/2 + i;}
if(b==a[m])) “found it” else “not found”
```

Choose mid point

$$T(N) = T(N/2) + c_0 \quad \rightarrow \quad T(N) \approx \text{Log}(N)$$

Dictionary: Sorting a nearly Uniform Sequence

- int a[0], a[1],a[2],a[3],.... a[m],.... a[2],a[N-1]

i

j

Dictionary: Same code EXCEPT

estimate location of b

x = fractional distance (0<x<1)

x = (b-a[i])/(a[j] – a[i]) ;

m = x (j-i) + i ;

m

$$N \to N^{\frac{1}{2}} \to N^{\frac{1}{4}} \to N^{\frac{1}{8}} \cdots \to N^{\frac{1}{2^n}} = 1 \quad \text{or} \quad n = log_2(log_2(N))$$

$$T(N) \simeq T(N^{1/2}) + c_0 \text{ or with } N = 2^n \ , \ T(n) = T(n/2) + c_0$$
$$\text{Master Equ.} \implies T(n) = O(\log(n)) = O(log(log_2(N))$$

- *Extra Knowledge Helps: % Error » 1/N$^{1/2}$*

Insertion Sort *--- Deck of Cards*

- Insertion Sort(a[0:N-1]):

```
for (i=1;  i < n;  i ++)
      for (j = i;  (j>0) && (a[j]<a[j-1]);  j--)
            swap a[j] and a[j-1] ;
```

Worst case $\Theta(N^2)$ number of "swaps" (i.e. time)

Outer loop trace for Insertion Sort

- a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]
  (Swaps)
- 6 | 5 2 8 3 4 7 1 (1)
  5← → 6
- 5 6 | 2 8 3 4 7 1
  (2)
  2← → 6
  2← → 5
- 2 5 6 | 8 3 4 7 1 (0)
- 2 5 6 8 | 3 4 7 1 (3)
- 2 3 5 6 8 | 4 7 1 (3)
- 2 3 4 5 6 8 | 7 1 (1)
- 2 3 4 5 6 7 8 | 1 (7)

Bubble Sort --- Sweep R to L

- *<u>Bubble Sort(a[0:N-1])</u>:*

```
for i=0 to n-1
      for j = n-1 to i + 1
            if a[j]<a[j-1] then
                  swap a[i] and a[j]
```

Worst case $\Theta(N^2)$ swaps (time)

Outer loop trace for Bubble Sort

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	(Swaps)
6	5	2	8	3	4	7	1	(7)
1 |	6	5	2	8	3	4	7	(3)
1	2 |	6	5	3	8	4	7	(3)
1	2	3 |	6	5	4	8	7	(3)
1	2	3	4 |	6	5	7	8	(1)
1	2	3	4	5 |	6	7	8	(0)
1	2	3	4	5	6 |	7	8	(0)
1	2	3	4	5	6	7	8	(17)

- NOTE SAME # OF SWAPS? WHY?

Average Number of N(N-1)/4 swaps

- Best Case: sorted order → 0 swaps
- Worst Case: reverse order→N(N-1)/2 swaps
  since 1 + 2 + ...+N-1 = N(N-1)/2
- Average Case: Pair up each of the N! permutations with its reverse order →Every pair must swap in one or the other: Thus average is half of all swaps → (1/2) N(N-1)/2 q.e.d.

Selection Sort --- (Bubble only the index)

- Selection Sort(a[0:N-1]):

```
for i=1 to n-2
   { min = i
    for j = n-1 to i + 1
          if a[j]<a[min] then
                  min = j;
        swap a[i] and a[min];
    }
```

worst case $\Theta(N)$ *swaps* + $\Theta(N^2)$ *comparisons*

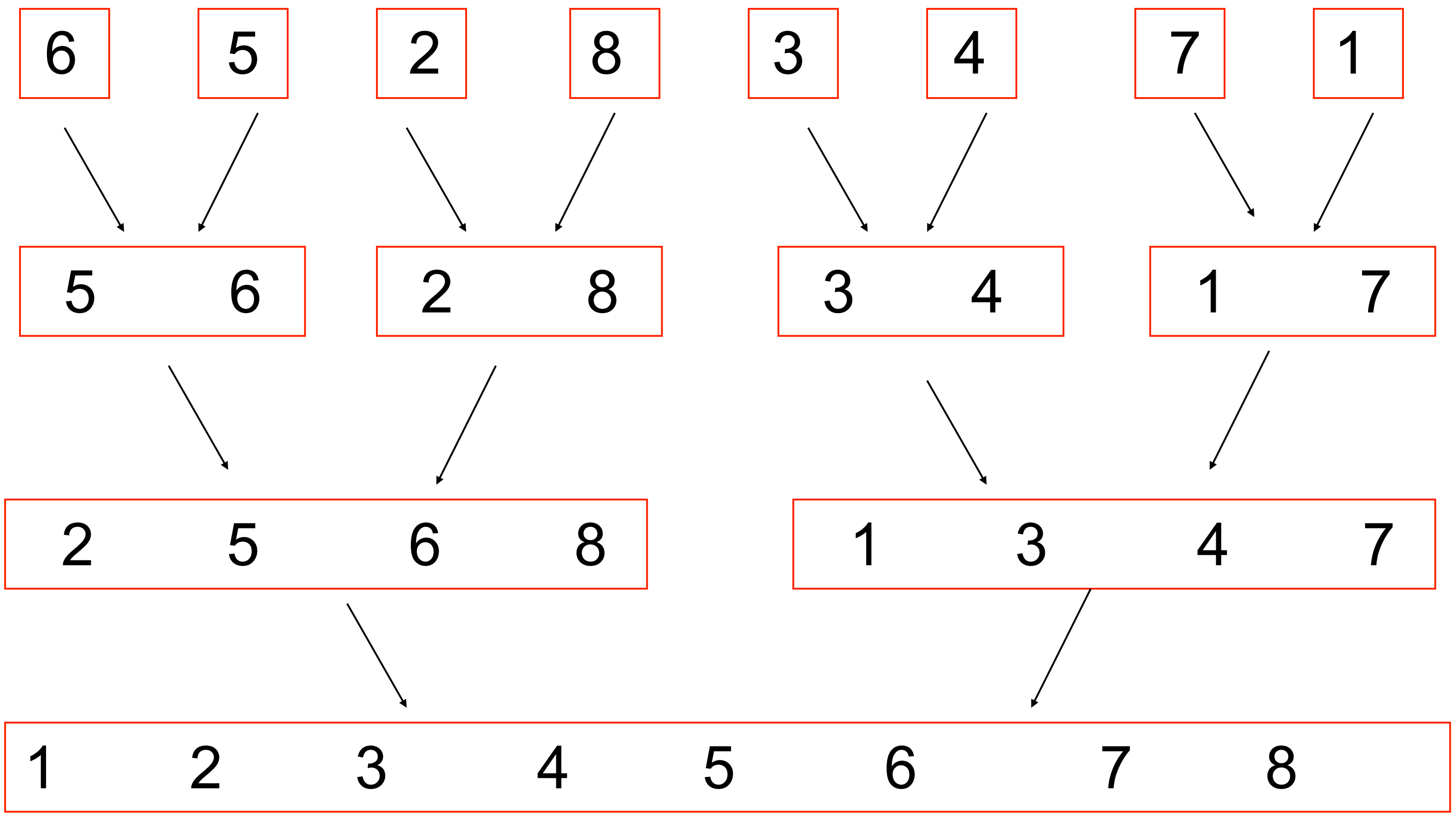
Outer loop trace for Selection Sort

(Swaps)	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
	6	5	2	8	3	4	7	1
(1)	1 ←						→	6
	1	5	2	8	3	4	7	6
(1)		2 ↔	5					
	1	2 |	5	8	3	4	7	6
(1)								
	1	2	3 |	8	5	4	7	6
(1)								
	1	2	3	4 |	5	8	7	6
(0)								
	1	2	3	4	5 |	6	7	8

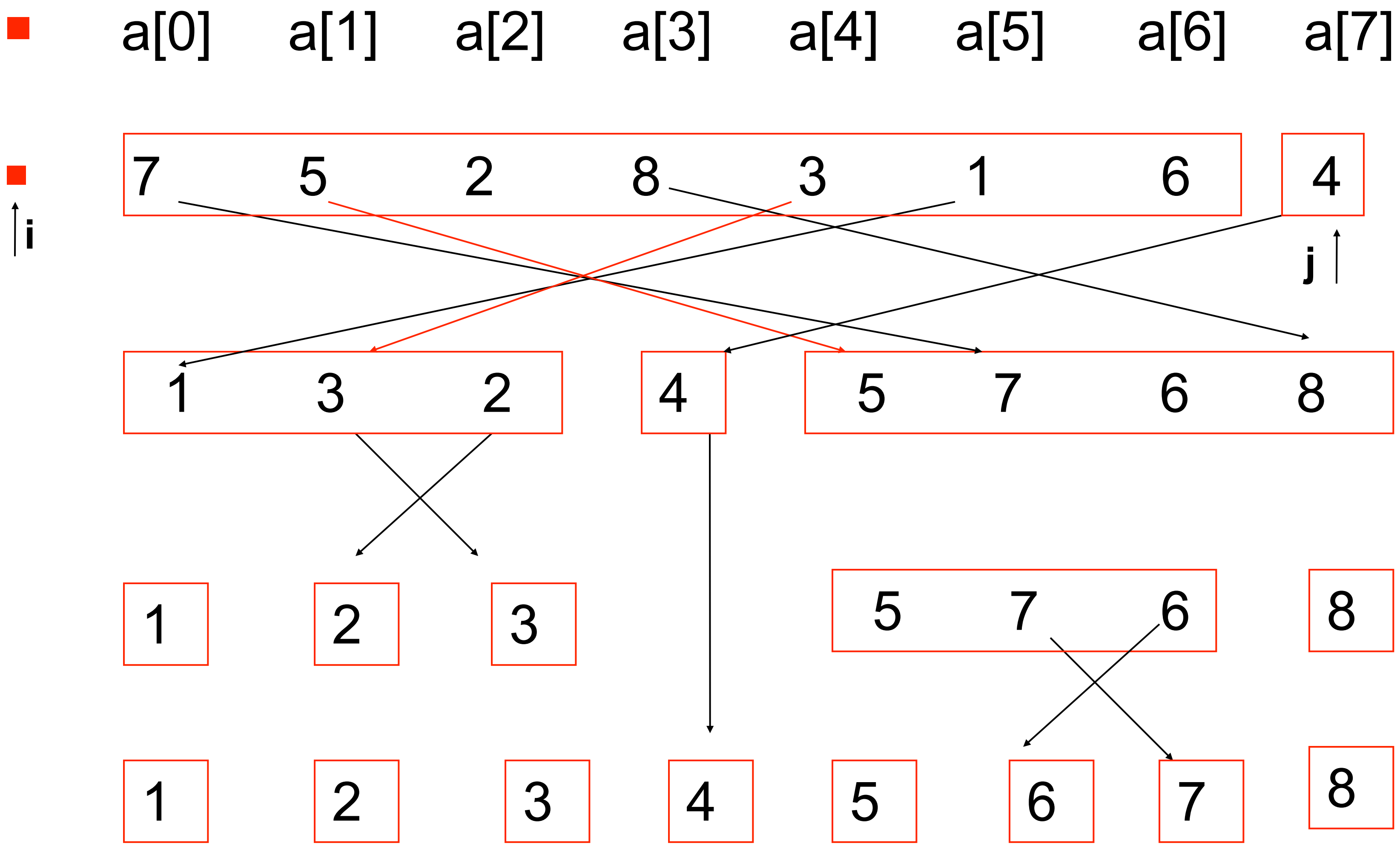
Merge Sort: Worst Case $\Theta(Nlog(N))$

```
void mergesort(int a[ ], int l, int r)
   if (r > l)  {
      m = (r+l)/2;
      mergesort(a, l, m);
      mergesort(a, m+1, r);
      for (i = l; i < m+1; I++)  b[i] = a[i];
      for (j = m; j < r; j++)    b[r+m-j] = a[j+1];   // reverse
      for (k = l; k <= r; k++)
              a[k] = (b[i] < b[j]) ? b[i++] : b[j--]; }
```

Outer loop trace for Merge Sort

- a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]
- 

Outer loop trace for Quick Sort (i moves before j)



Quick Sort: original Hoare partition

```
void quicksort(int a[ ], int l, int r)
  if (r > l){
      v = a[r]; i = l-1; j = r;
      for (;;){  while (a[++i] < v);        // move first i to right
                 while (a[--j] > v) ;        // then mover j left
                   if (i >= j) break  ;
                  swap(&a[i],  &a[j]); }
        swap(&a[i], &a[r]);                 // move pivot in to center
        quicksort(a, l, i-1);
        quicksort(a, i+1, r);
    }
```

Quick Sort: My Implementation

```
void quicksort(int a[], int l, int r)
 {  int i, j, vl
   if (r > l)
    {   v = a[r]; i = l-1; j = r;
      for (;;)
       {   while (a[++i] < v && i >= j ) ;  // Move i first from left
           while (a[--j] >= v && i >= j) ;  // Move j second from right
           if (i >= j) break;
           swap(&a[i], &a[j]);   } //  swap root to divide left and right sublist
      swap(&a[i], &a[r]);
      quicksort(a, l, i-1);     quicksort(a, i+1, r);
    }  }
```

Quick Sort: CLRS:7 Implementation

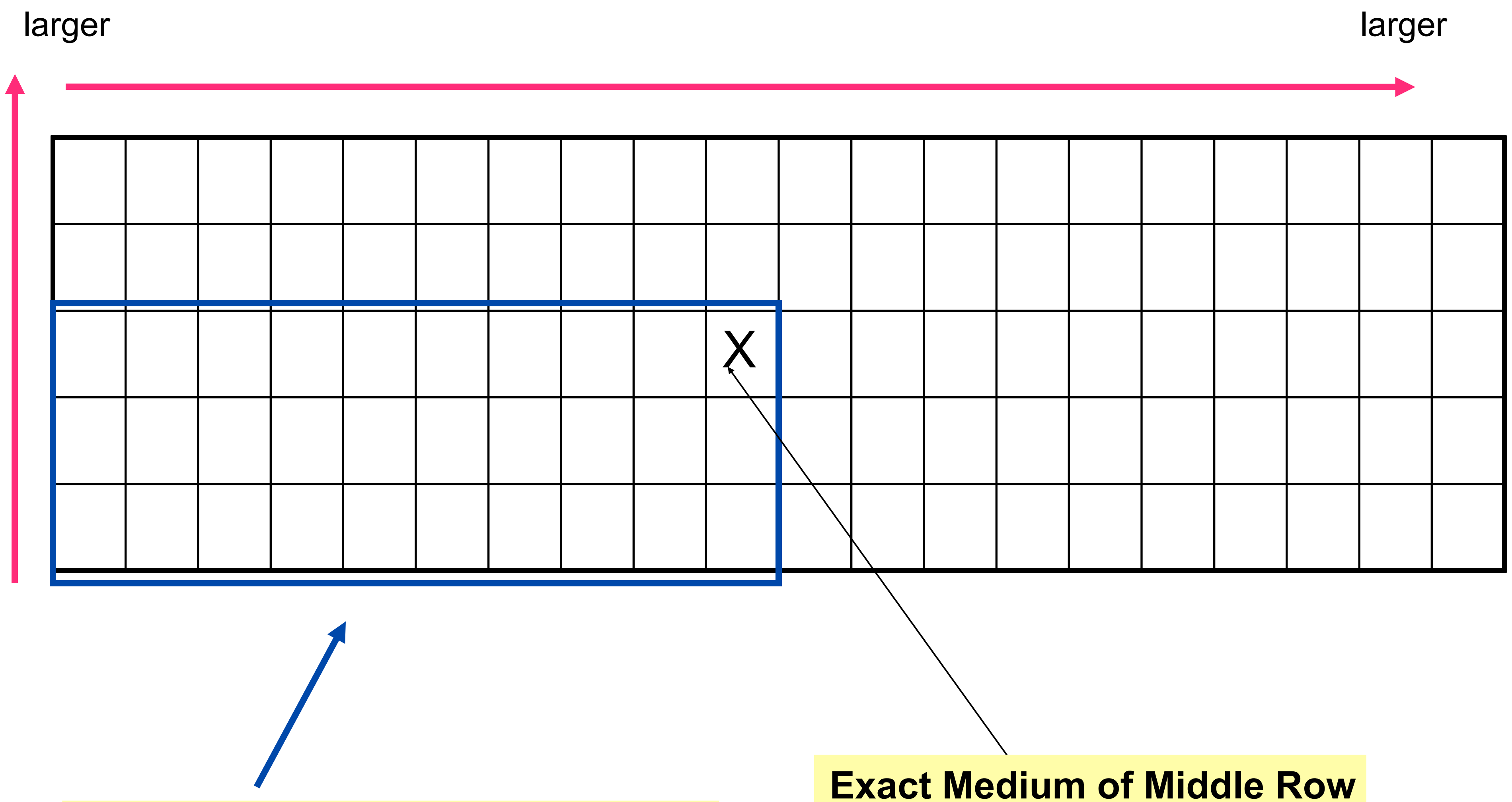
```
void quicksort(int a[ ], int l, int r)
  if (r > l){
      v = a[r]; i = l-1;
      for (int j= l ; j<r ; j++){
         if (a[j] <= v) {
               i +=1 ;                          // move first i to right
              swap(&a[i],  &a[j])  }
       swap(&a[i], &a[r]);                      // move pivot in to center
       quicksort(a, l, i-1);
       quicksort(a, i+1, r);
    }
```

Very cute: See CLRS page 172 Figure 7.1

- Worst Case (choose smallest as pivot!):
  - T(N) = T(N-1) + c N ➔ T(N) =O(N^2)
- Best Case:(choose median as pivot)
  - T(N) = 2 T(N/2) + cN ➔ T(N) = O(N log(N))
- Average Case:
  - T(N) = 2[T(0) + T(1) + …+ T(N-1)]/N + c N
    - ➔ T(N) = O(N log (N))
    - ➔ Using Calculus if you are lazy! (x = N)

$$xT(x) \simeq 2\int_0^x dy T(y) + cx^2 \quad x\frac{dT(x)}{dx} = T(x) + 2cx \Rightarrow T(x) = 2cx\log(x)$$

5 row of N/5 Columns



Exact Medium of Middle Row

3/5 * 1/2 N Boxes smaller than X

Median Finding: Quick Select

- Median is the element a[m] so that half is less/equal

- Generalize to finding k-th smallest in set S

- Quick(S,k): |S| = size of S
  - If |S| = 1, the k = 1 in S
  - Pick pivot v 2 S & Partition S – {v} into S_L & S_H
    - If k< $|S_L|$ + 1 then k-th 2 S_L: Quick(S_L,k)
    - If k = $|S_L|$ + 1 k-th is v : exit
    - If k > $|S_L|$ + 1 then k-th 2 S_R: Quick(S_R,k- $|S_L|$-1)

Now: T(N) = O(N) is average performance

T(N) = [T(0) + T(1) + … T(N-1)]/N + c N

$$xT(x) \simeq \int_0^x dy T(y) + cx^2 \quad x\frac{dT(x)}{dx} = 2cx \Rightarrow T(x) = 2cx$$

Can do better-- Worst Case O(N Lg(N)) !

- *Approximate Media Selector for pivot v:*
  - *Partition in 5 rows of N/5*
  - *Sort each column*
  - *Find (Exact) Medium of Middle list!*

- *Result there are (3/5)(1/2)N elements smaller than pivot v*

- *K-th find is O(N) --- Double recursion!*
  - *Sort of N/5 col   O(N)*
  - *Find media of T(N/5)*
  - *Find k-th in T(7N/10) at worst*

- *T(N) < C0 * (N/5) + T(N/5) + T(7N/10)*
  - *Try solution: T = C N*
  - *C(N – N/5 – 7N/10) = C N/10 < C0 N/5*
  - *C < 2 C0*

O(N): Bin, Radix &Bucket

- BIN Sort – make histogram (Counting sort CLRS 8.2):
  - N integers 0 < a[i] < M in the range v= 0,...,M-1.
  - Count number of occurrences in a[i]

```
    for(v=0; v<M; v++)  bin[ v ] =0;
    for(i=0;i<N; i++)    bin[a[i]] ++;
            j=0;
            for(v=0; v< M; v++) {
                 for(i=0; i<bin[v]; i++)
                      a[ j ] = v; j++;    }
```

→ O(M + N) so if M » N it is O(N)

Beating N log N: Radix Sort (IBM Card Sorter!)

- Represent integers in a[i] in base B: $n_0 + n_1 B + n_2 B^2 + ... + n_p B^P$
- Sort into buckets by low digits first: n_0, then n_1, etc.

Queues: B= 10 Example: 64, 8, 216, 512, 27, 729, 0, 1, 343, 125

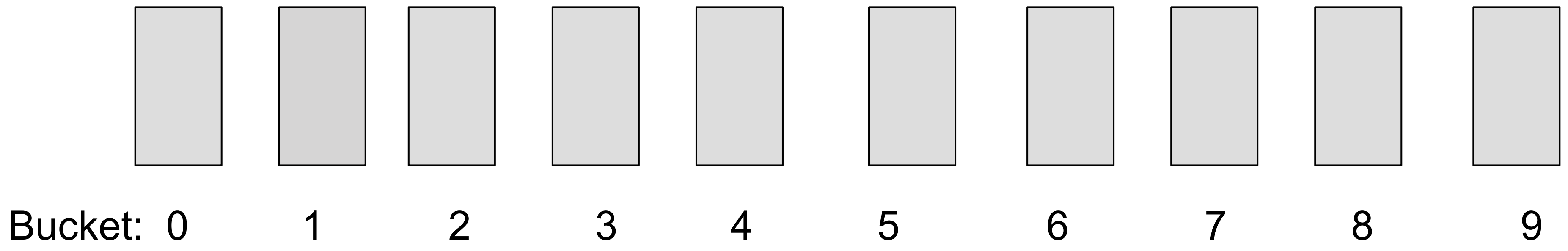
Bucket:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
#1	0	1	512	343	64	125	216	27	8	729
	8 1 0	216 512	729 27 125		343		64			
#3	64 27 8 1 0	125	216	343		512		729		

O(N P) where $B^P = N$ or P= log(N) / log(B) = O(1)

Bucket Sort

- Choose B Buckets as bins for high digits of a[i]
  - place N numbers in a[i] in buckets
  - Sort average of N/B elements in each bucket.
  - CRLS uses insertion sort

Linked list:



$$\Longrightarrow O(N + B(N/B)\log(N/B)) = O(N + N\log(N/B))$$

$$B = O(N)$$

Shell Sort: $O(N^\gamma)\ 1 < \gamma < 2$

Use insertion sort skip lists a[i] <a[i+h] in descending order

$$1 = h_1 < h_2 < \cdots < h_k < \cdots < N$$

```
void shellsort(int a[], int N)                                  //Kunth 1969
        int i, j, h , v;
        for (h = 1; h <= N/9; h = 3*h+1) ;                      // Find Largest h
        for ( ; h > 0; h = h/3)                                  // Descending skip distance
            for (i = h; i < N; i++)
                {   v = a[i];                                    // Insertion sort
                    for ( j = i; (j>=h) && (a[j-h] > v)
                        {   j   -=  h;
                            a[j] = a[j-h];
                        }
                    a[j] = v;
                }
```

Properties of Shell Sort

- *Shell's sequence:*
  - *h = 1, 2, 4, 8, ... 2^N* ➔ *Worst Case: $O(N^2)$*

- *Hibbards seqence:*
  - *1, 3, 7,15, ... $2^k - 1$* ➔ *Average Case: $O(N^{5/4})$,*

    ➔ *Worst: $O(N^{3/2})$*

- *Theorem:*

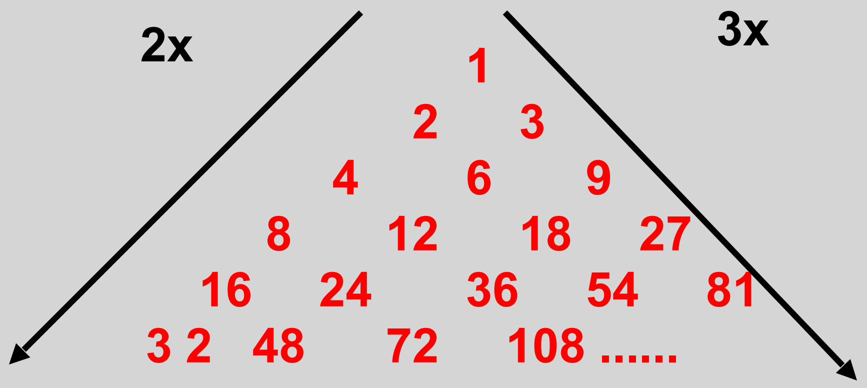
  A "h = p" sorted list remains p sorted after a "h =q" sort!
  (Proof is hard -- for me anyway)

Cute increment : $T(N) = \Omega(N \log^2(N))$ for Shell Sort!

- *Each sort finds at most one adjacent elment, a[(i-1) h], out order!*
  - *Each pass O(N)*

  .... a[(i-5)h], a[(i-4)h], a[(i-3)h], a[(i-2)h], a[(i-1)h] ,a[i h]
  All other are (2 n + 3 m) h away! (e.g 5 = 2 + 3).
- *The number of increments h's smaller than O(N) is the area: O(log(N) log(N)). q.e.d.*

h-triangle: 2x/3x for left/right child ➔



$x = \log_2(N)$ and $y = \log_3(N)$ ➔ Area = $x * y/2 = O(\log(N)*\log(N))$