Data Structure

Sorting and Searching

Instructors:
Md Nazrul Islam Mondal &
Rizoan Toufiq
Department of Computer Science & Engineering
Rajshahi University of Engineering &
Technology Rajshahi-6204

Outline

- Sorting
- Insertion Sort
- Selection Sort
- Merging
- Merge-Sort
- Radix Sort
- Searching and Data Modification
- Hashing

Sorting

Sorting

- Let A be a list of n elements $A_{1,}$ $A_{2,}$ $A_{3,}$ … , A_n in memory
- Sorting A refers to the operation of rearrange the contents of A so that they are increasing in order(numerically or lexicographically), that is

$$A_1 \leq A_2 \leq A_3 \leq \ldots \leq A_n$$

- There are n! ways that the contents can appear in A.
- There are n! possibilities.

Sorting (Complexity)

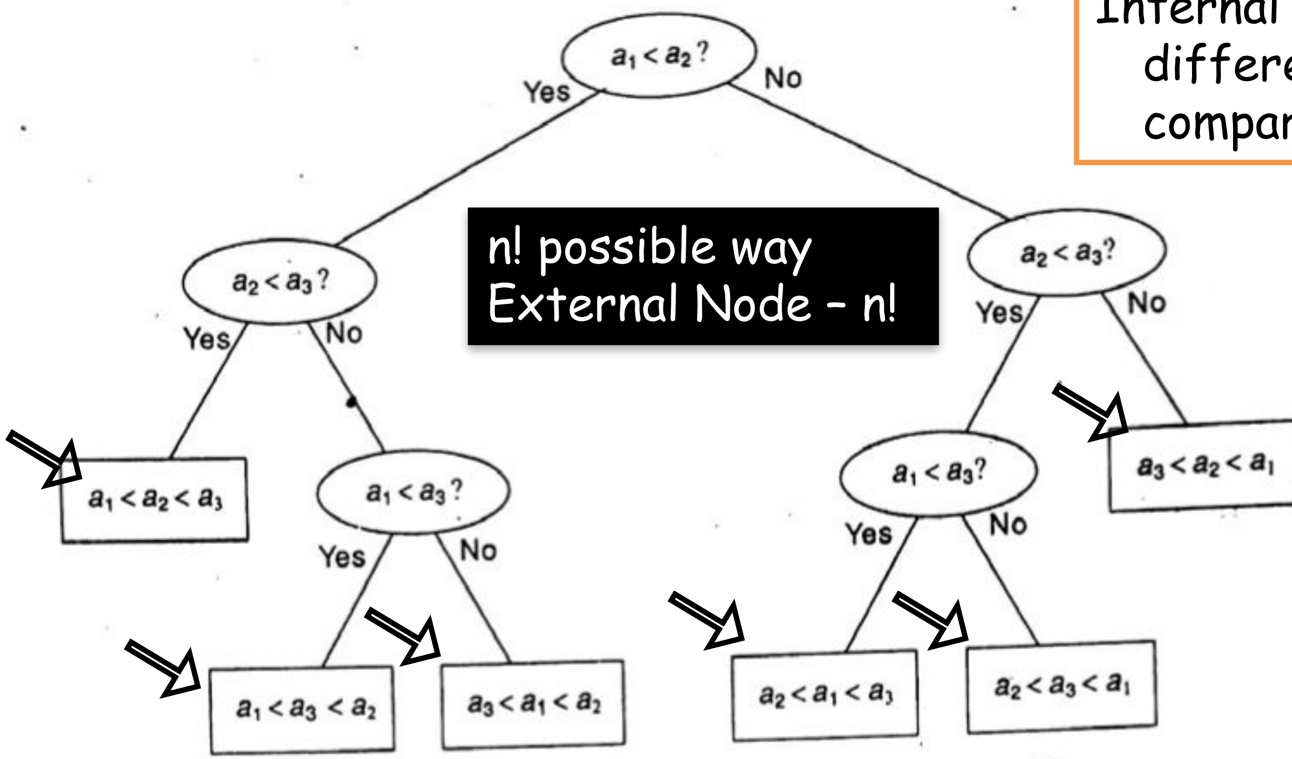
- Normally, the complexity function measures only the number of comparisons.
- We consider two case: i) Average Case and ii) Worst Case

Algorithm	*Worst Case*	*Average Case*
Bubble Sort	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$
Quicksort	$\frac{n(n+3)}{2} = O(n^2)$	$1.4n \log n = O(n \log n)$
Heapsort	$3n \log n = O(n \log n)$	$3n \log n = O(n \log n)$

Sorting (Complexity: Lower Bound)

- Complexity Less than nlogn? Answer: No

Internal node – different comparison



n! possible way
External Node – n!

Fig. 9.1 *Decision Tree T for Sorting n = 3 items*

Sorting
(Complexity : Lower Bound)

- **Worst Case** – the length of the longest path in the decision tree T or, in other words, **the depth D of the tree T**
- **Average Case** – the **average external path length** $\bar{E}$ of the tree T.

- D = 3
- $\bar{E}$ = 1/6(2+3+3+3+3+2) = 2.667

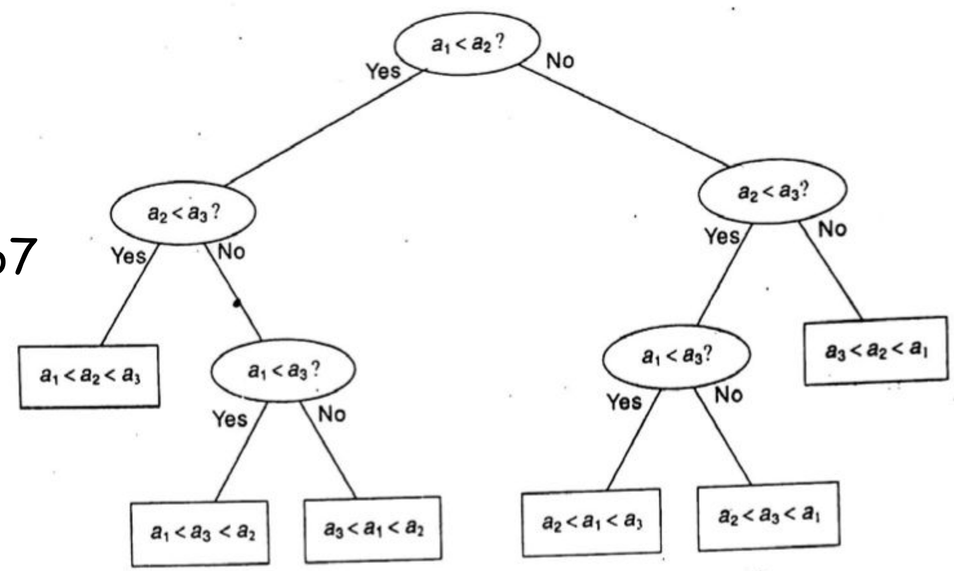


Fig. 9.1 *Decision Tree T for Sorting n = 3 items*

Sorting
(Complexity : Lower Bound)

- Suppose T is an external binary tree with N external nodes, depth D and external path length E(T).
- Any such tree cannot have more than 2^D external nodes and so

$$2^D \geq N \;\Rightarrow D \geq \log N$$

- When T is a complete tree, T will have a **minimum external path length E(L)** among all such tree with N nodes
  - There are N paths with length logN or logN +1
  - So **E(L) = logN + logN+1 + logN+ … + 1 + … logN**
    = NlogN+ O(N)≥NlogN

- *Average External Path Length* $\bar{E} = E(L)/N \geq N\log N/N = \log N$

$$\bar{E} \geq \log N$$

Sorting (Complexity : Lower Bound)

- Now suppose T is the decision tree corresponding to a sorting algorithm S which sort n items.
- Then T has n! external node.
- N = n!
- We know, D≥logN or D≥ logn! ≈ nlogn (Worst Case Complexity)
- We know, $\bar{E}$≥logN or $\bar{E}$≥logn! ≈ nlogn (Average Case Complexity)

- The condition logn! ≈ nlogn comes from Striling's formula, that

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\frac{1}{12n}+\cdots\right)$$

- Thus **nlogn** is a **lower bound** for both the worst case and the average case. In other words, **O(nlogn)** is the best possible for any sorting algorithm which sorts n items

Sorting
(Sorting Files; Sorting Pointers)

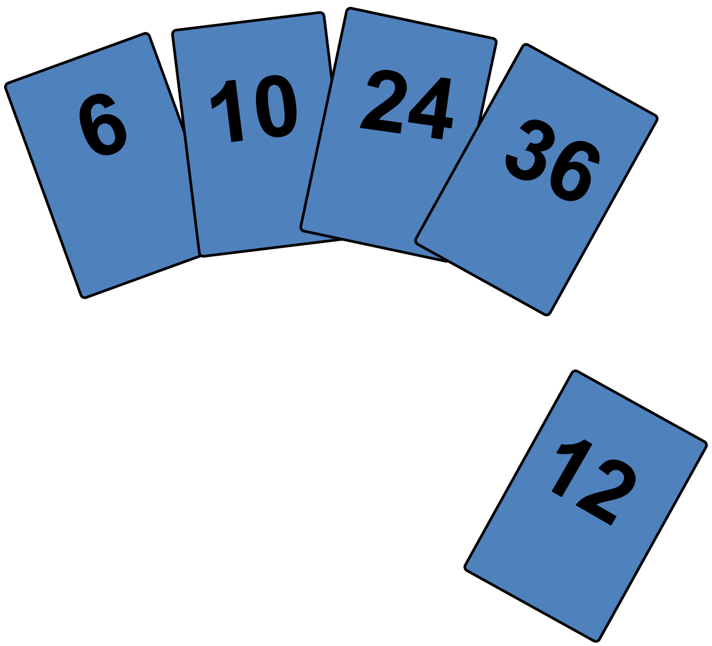
- Suppose a file F of records $R_1, R_2, \ldots, R_n$ is stored in memory.
- "Sorting F" **refers** to sorting F with respect to some field K with corresponding values $k_1, k_2, \ldots, k_n$. That is, the records are ordered so tha

$$k_1 \leq k_2 \leq \cdots \leq k_n$$

- The filed K is called the **sort key** (primary key)

Insertion Sort

Insertion Sort



To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort

- Suppose an Array A contain 8 elements as follows:
  77 33 44 11 88 22 66 55

Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1:	−∞	77	33	44	11	88	22	66	55
K = 2:	−∞	77	33	44	11	88	22	66	55
K = 3:	−∞	33	77	44	11	88	22	66	55
K = 4:	−∞	33	44	77	11	88	22	66	55
K = 5:	−∞	11	33	44	77	88	22	66	55
K = 6:	−∞	11	33	44	77	88	22	66	55
K = 7:	−∞	11	22	33	44	77	88	66	55
K = 8:	−∞	11	22	33	44	66	77	88	55
Sorted:	−∞	11	22	33	44	55	66	77	88

Fig. 9.3 *Insertion Sort for n = 8 Items*

Insertion Sort

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	77	33	44	11	88	22	66	55

- Take value
- Shift values
- Assign Value

ITEM

77

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	77	33	44	11	88	22	66	55

A[1]←ITEM or A[1]←77

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	77	33	44	11	88	22	66	55

Insertion Sort

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	77	33	44	11	88	22	66	55

- Take value
- Shift values
- Assign Value

ITEM

33

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	77	77	44	11	88	22	66	55

A[1]←ITEM or A[1]←33

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	33	77	44	11	88	22	66	55

Insertion Sort

- Take value
- Shift values
- Assign Value

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	33	77	44	11	88	22	66	55

ITEM

44

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	33	77	77	11	88	22	66	55

A[2]←ITEM or A[2]←44

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	33	44	77	11	88	22	66	55

Insertion Sort

- Take value
- Shift values
- Assign Value

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	33	44	77	11	88	22	66	55

ITEM

11

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	33	33	44	77	88	22	66	55

A[1]←ITEM or A[1]←11

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	44	77	88	22	66	55

Insertion Sort

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	44	77	88	22	66	55

- Take value
- Shift values
- Assign Value

ITEM

88

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	44	77	88	22	66	55

A[5]←ITEM or A[5]←88

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	44	77	88	22	66	55

Insertion Sort

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	44	77	88	22	66	55

- Take value
- Shift values
- Assign Value

ITEM
22

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	33	44	77	88	66	55

A[2]←ITEM or A[2]←22

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	22	33	44	77	88	66	55

Insertion Sort

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	22	33	44	77	88	66	55

- Take value
- Shift values
- Assign Value

ITEM
66

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	33	44	77	77	88	55

A[5]←ITEM or A[5]←66

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	22	33	44	66	77	88	55

Insertion Sort

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	22	33	44	66	77	88	55

- Take value
- Shift values
- Assign Value

ITEM

55

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	33	33	44	66	66	77	88

A[5]←ITEM or A[5]←55

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
-∞	11	22	33	44	55	66	77	88

Insertion Sort

Algorithm 9.1: (Insertion Sort) INSERTION(A, N).
This algorithm sorts the array A with N elements.

1. Set $A[0] := -\infty$. [Initializes sentinel element.]
2. Repeat Steps 3 to 5 for K = 2, 3, ..., N:
3. Set TEMP := A[K] and PTR := K − 1.
4. Repeat while TEMP < A[PTR]:
   - (a) Set A[PTR + 1] := A[PTR]. [Moves element forward.]
   - (b) Set PTR := PTR − 1.

   [End of loop.]
5. Set A[PTR + 1] := TEMP. [Inserts element in proper place.]

   [End of Step 2 loop.]
6. Return.

Insertion Sort (Complexity)

- Worst Case: Data are sorted in reverse order.

$$f(n) = 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

- Average Case: Random order

$$f(n) = \frac{1}{2} + \frac{2}{2} + \cdots + \frac{n-1}{2} = \frac{n(n-1)}{4} = O(n^2)$$

Algorithm	*Worst Case*	*Average Case*
Insertion Sort	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{4} = O(n^2)$

Selection Sort

Selection Sort

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted

Example

8	4	6	9	2	3	1

1	4	6	9	2	3	8

1	2	6	9	4	3	8

1	2	3	9	4	6	8

1	2	3	4	9	6	8

1	2	3	4	6	9	8

1	2	3	4	6	8	9

1	2	3	4	6	8	9

Selection Sort

- Suppose an Array A contain 8 elements as follows:
  77 33 44 11 88 22 66 55

Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1, LOC = 4	77	33	44	11	88	22	66	55
K = 2, LOC = 6	11	33	44	77	88	22	66	55
K = 3, LOC = 6	11	22	44	77	88	33	66	55
K = 4, LOC = 6	11	22	33	77	88	44	66	55
K = 5, LOC = 8	11	22	33	44	88	77	66	55
K = 6, LOC = 7	11	22	33	44	55	77	66	88
K = 7, LOC = 7	11	22	33	44	55	66	77	88
Sorted:	11	22	33	44	55	66	77	88

Fig. 9.4 *Selection Sort for n = 8 Items*

Selection Sort

Procedure 9.2: MIN(A, K, N, LOC)

An array A is in memory. This procedure finds the location LOC of the smallest element among A[K], A[K + 1], ..., A[N].

1. Set MIN := A[K] and LOC := K. [Initializes pointers.]
2. Repeat for J = K + 1, K + 2, ..., N:

   If MIN > A[J], then: Set MIN := A[J] and LOC := A[J] and LOC := J.

   [End of loop.]
3. Return.

Algorithm 9.3: (Selection Sort) SELECTION(A, N)

This algorithm sorts the array A with N elements.

1. Repeat Steps 2 and 3 for K = 1, 2, ..., N – 1:
2. Call MIN(A, K, N, LOC).
3. [Interchange A[K] and A[LOC].]

   Set TEMP := A[K], A[K] := A[LOC] and A[LOC] := TEMP.

   [End of Step 1 loop.]
4. Exit.

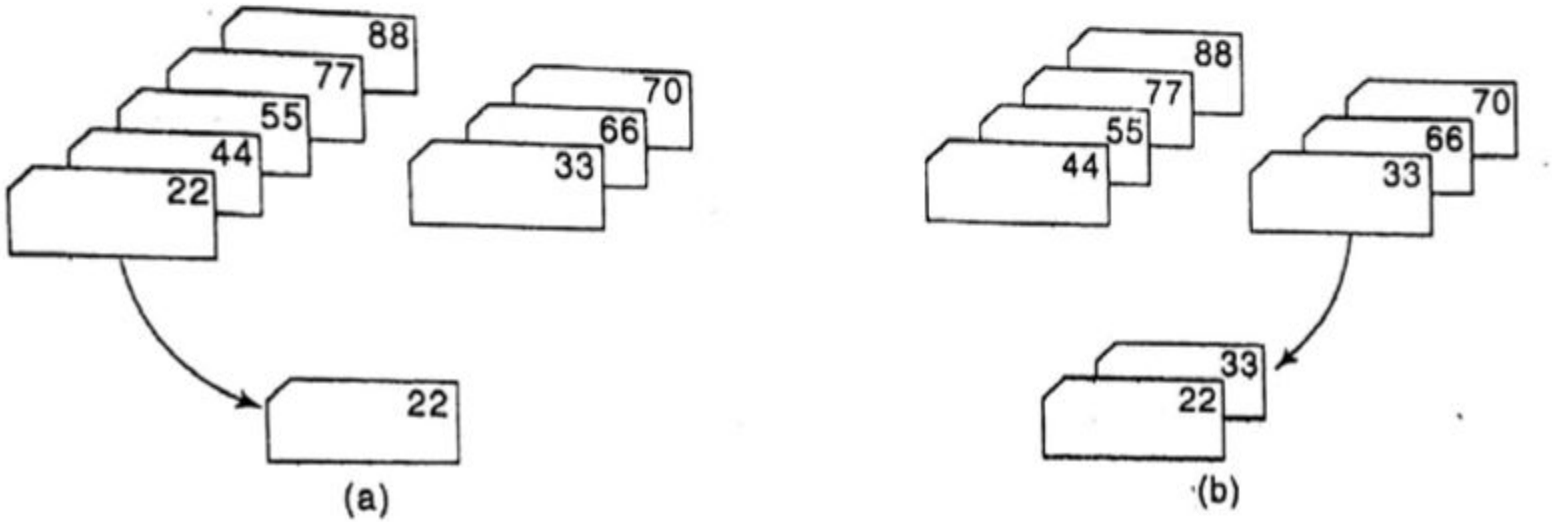
Selection Sort

$$f(n) = (n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2} = O(n^2)$$

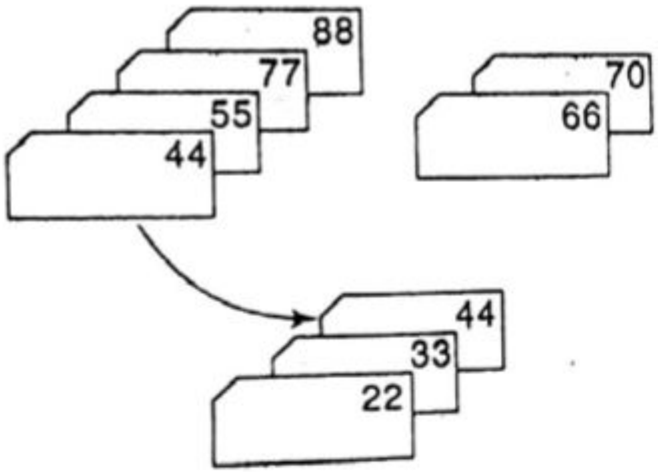
Algorithm	*Worst Case*	*Average Case*
Selection Sort	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$

Merging

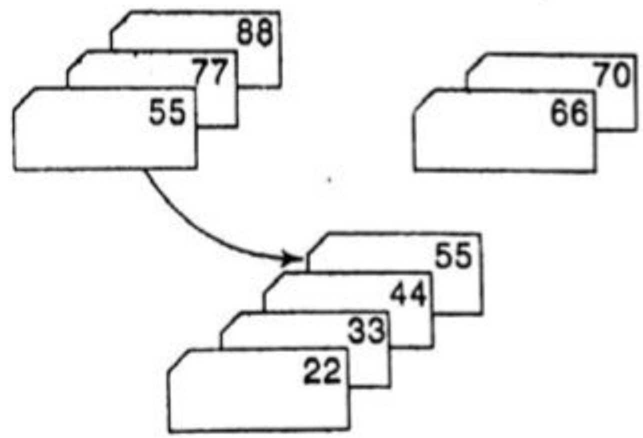
Merging



Merging



(c)



(d)

Merging

I=1

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

J=1

33	66	70
B[1]	B[2]	B[3]

- Marge A and B into C, the size of C is 5+3 = 8

K=1

C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

~~I=1~~ I=2

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

J=1

33	66	70
B[1]	B[2]	B[3]

K=1

22							
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

I=2

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

~~J=1~~ J=2

33	66	70
B[1]	B[2]	B[3]

K=2

22	33						
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

~~I=2~~ I=3

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

J=2

33	66	70
B[1]	B[2]	B[3]

K=3

22	33	44					
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

~~I=3~~ I=4

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

J=2

33	66	70
B[1]	B[2]	B[3]

K=4

22	33	44	55				
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

I=4

22	44	55	$\boxed{77}$	88
A[1]	A[2]	A[3]	A[4]	A[5]

~~J=2~~ J=3

33	$\boxed{66}$	70
B[1]	B[2]	B[3]

K=5

22	33	44	55	$\boxed{66}$			
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

I=4

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

J=4,
Break

~~J=3~~

33	66	70
B[1]	B[2]	B[3]

K=6

22	33	44	55	66	70		
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

I=4

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

- Condition of Loop: I≤sizeof(A) and J≤sizeof(B)

- **Copy the remaining element of A if J> sizeof(B)**
- **Copy the remaining element of B if I> sizeof(A)**

K=7

22	33	44	55	66	70		
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

~~I=4~~				I=5
22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

- Copy the remaining element of A if J> sizeof(B)

						K=7	
22	33	44	55	66	70	77	
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

~~I=5~~

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

I=6
Break

- Copy the remaining element of B if J> sizeof(A)

K=8

22	33	44	55	66	70	77	88
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

22	33	44	55	66	70	77	88
C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]

Merging

Algorithm 9.4: MERGING(A, R, B, S, C)
Let A and B be sorted arrays with R and S elements, respectively. This algorithm merges A and B into an array C with N = R + S elements.

1. [Initialize.] Set NA := 1, NB := 1 and PTR := 1.
2. [Compare.] Repeat while NA ≤ R and NB ≤ S:
   If A[NA] < B[NB], then:
      (a) [Assign element from A to C.] Set C[PTR] := A[NA].
      (b) [Update pointers.] Set PTR := PTR + 1 and NA := NA + 1.
   Else:
      (a) [Assign element from B to C.] Set C[PTR] := B[NB].
      (b) [Update pointers.] Set PTR := PTR + 1 and NB := NB + 1.
   [End of If structure.]
   [End of loop.]
3. [Assign remaining elements to C.]
   If NA > R, then:
      Repeat for K = 0, 1, 2, ..., S – NB:
         Set C[PTR + K] := B[NB + K].
      [End of loop.]
   Else:
      Repeat for K = 0, 1, 2, ..., R – NA:
         Set C[PTR + K] := A[NA + K].
      [End of loop.]
   [End of If structure.]
4. Exit.

Merging

- Let, N ← sizeof(A) + sizeof(B)
- Complexity: O(N)

Merging

Nonregular Matrices

- Given,
  - The lower bound of *A* is LBA, and number of element□r
  - The lower bound of B is LBB and number of element□s
  - The lower bound of *C* is LBC
- UBA□ LBA+r-1, UBB□ LBB+s-1,

Procedure 9.5: MERGE(A, R, LBA, S, LBB, C, LBC)
This procedure merges the sorted arrays A and B into the array C.

1. Set NA := LBA, NB := LBB, PTR := LBC, UBA := LBA + R − 1, UBB := LBB + S − 1.
2. Same as Algorithm 9.4 except R is replaced by UBA and S by UBB.
3. Same as Algorithm 9.4 except R is replaced by UBA and S by UBB.
4. Return.

Merging

Binary Search and Insertion ALgorithm

22	44	55	77	88
A[1]	A[2]	A[3]	A[4]	A[5]

33	66	70
B[1]	B[2]	B[3]

- Take 33 from B
- Search 33 in A and find the position logn (Binary Search)
- Insert 33 into C

- **Note: Best for the only case,**
  - **sizeof(A)>>sizeof(B) or sizeof(A)<<sizeof(B)**

Merging

Binary Search and Insertion ALgorithm

- Let, sizeof(A) = 5 and sizeof(B)=100,
  - Merge algorithm: 100 comparisons required
  - **Binary search and insertion Algorithm: 5xlog(100) = 5x7 = 35**
- Let size(A) = 90 and sizeof(B) = 100
  - Merge algorithm: 100 comparisons required
  - **Binary search and insertion algorithm: 90xlog100 = 630**

- **Note: Best for the only case,**
  - **sizeof(A)>>sizeof(B) or sizeof(A)<<sizeof(B)**

Merging

Binary Search and Insertion ALgorithm

- Two improved algorithm:
  1) **Reducing the target set:**
     - Let First binary Search: A[1] is to be inserted after B[16],
     - Next Binary search must be performed between B[17] to B[100] and so on.

  2) **Tabbing**:
     - We have, {B[20], B[40], B[60], B[80], B[100]}
     - Use Linear search to find A[1] ≤ B[K], K = 20, 40, 60, 80, or 100
     - Then use binary search on B[K-20], B[K-19],…,B[K]

Merge-Sort

Merge-Sort

- Input Data Set:

66, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30

Pass 1. Merge each pair of elements to obtain the following list of sorted pairs:

33, 66   22, 40   55, 88   11, 60   20, 80   44, 50   30, 70

Pass 2. Merge each pair of pairs to obtain the following list of sorted quadruplets:

22, 33, 40, 66   11, 55, 60, 88   20, 44, 50, 80   30, 77

Pass 3. Merge each pair of sorted quadruplets to obtain the following two sorted subarrays:

11, 22, 33, 40, 55, 60, 66, 88   20, 30, 44, 50, 77, 80

Pass 4. Merge the two sorted subarrays to obtain the single sorted array

11, 20, 22, 30, 33, 40, 44, 50, 55, 60, 66, 77, 80, 88

Merge-Sort

66	33	40	22	55	88	60	11	80	20	50	44	77	33
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]	A[13]	A[14]

Explain the following Function (**Review**):
MERGE(A,R,LBA,B,S,LBB,C,LBC)

Merge-Sort

22	33	40	66	11	55	60	88
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

Explain the following Function (**Review**):
MERGE(A,4,1,A,4,4+1,B,1)

- **A □Array**
  LBA□ 1, and Number of Element 4, UBA□4
- **A□Array**
- **LBA□ 5, and Number of Element 4, UBA□8**
- **Merge this Two array and save into B**

11	22	33	40	55	60	66	88
B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]

Merge-Sort

Algorithm 9.7: MERGESORT(A, N)
This algorithm sorts the N-element array A using an auxiliary array B.

1. Set L := 1. [Initializes the number of elements in the subarrays.]
2. Repeat Steps 3 to 6 while L < N:
3. Call MERGEPASS(A, N, L, B).
4. Call MERGEPASS(B, N, 2 * L, A).
5. Set L := 4 * L.

   [End of Step 2 loop.]
6. Exit.

Merge-Sort

MERGESORT(A,N)
[A Array, NTotal Element]

MERGEPASS(A,N,L,B)

[
A Array,
NTotal Element
L each subarray has L element
(Except Last one)
B Auxiliary Array
]

MERGEPASS(B,N,2L,A)

[
A Array,
NTotal Element
L each subarray has L element
(Except Last one)
B Auxiliary Array
]

Merge-Sort

Procedure 9.6: MERGEPASS(A, N, L, B)

The N-element array A is composed of sorted subarrays where each subarray has L elements except possibly the last subarray, which may have fewer than L elements. The procedure merges the pairs of subarrays of A and assigns them to the array B.

1. Set Q := INT(N/(2*L)), S := 2*L*Q and R := N – S.
2. [Use Procedure 9.5 to merge the Q pairs of subarrays.]
   Repeat for J = 1, 2, …, Q:
   (a) Set LB := 1 + (2*J – 2)*L. [Finds lower bound of first array.]
   (b) Call MERGE(A, L, LB, A, L, LB + L, B, LB).
   [End of loop.]
3. [Only one subarray left?]
   If R ≤ L, then:
      Repeat for J = 1, 2, …, R:
         Set B(S + J) := A(S + J).
      [End of loop.]
   Else:
      Call MERGE(A, L, S + 1, A, R, L + S + 1, B, S + 1).
   [End of If structure.]
4. Return.

Merge-Sort

MERGESORT(A,N)
[A ←Array, N←Total Element]

L = 1

MERGEPASS(A,14,1,B)

MERGEPASS(B,10,2,A)

B← {MERGE(A[1],A[2]), MERGE(A[3],A[4]),… MERGE(A[13],A[14]}

B←{ **33,66**, 22,40, 55,88, 11,60, 20,80, 44,50, 30,70}

Merge-Sort

MERGESORT(A,N)
[A ←Array, N←Total Element]

L = 1

MERGEPASS(A,14,1,B)

B←{ **33,66**, 22,40, 55,88, 11,60, 20,80, 44,50, 30,70}

MERGEPASS(B,10,2,A)

A← {MERGE(B[1-2],B[3-4]),…
MERGE(B[9-10],B[11-12]),
COPY(B[13-14]}

A←{ **22,33,40,66**, **11,55,60,88**,
20,44,50,80, 30,70}

Merge-Sort

MERGESORT(A,N)
[A ←Array, N←Total Element]

L = 4*L = 4,
Condition: N>L

MERGEPASS(A,14,4,B)

MERGEPASS(B,10,8,A)

B← {MERGE(A[1-4],A[5-8]), MERGE(A[9-12],A[13-14])}

B←{ **11,22,33,40,55,60,66,88**, **20,30,44,50,70,80**}

Previous Slide:A←{ **22,33,40,66**, **11,55,60,88**, **20,44,50,80**, 30,70}

Merge-Sort

MERGESORT(A,N)
[A Array, NTotal Element]

L = 4*L = 4,
Condition: N>L

MERGEPASS(A,14,4,B)

MERGEPASS(B,10,8,A)

B{ 11,22,33,40,55,60,66,88, 20,30,44,50,70,80}

A {MERGE(B[1-8],B[9-14])}

A{
11,20,22,30,33,40,44,50,55,60,66,70,80,88}

Merge-Sort

MERGESORT(A,N)
[A Array, NTotal Element]

L = 4*L = 16, N=12
Condition: N>L false Break

A{ 11,20,22,30,33,40,44,50,55,60,66,70,80,88}

Algorithm	*Worst Case*	*Average Case*	*Extra Memory*
Merge-Sort	$n \log n = O(n \log n)$	$n \log n = O(n \log n)$	$O(n)$

Radix Sort

Radix Sort

- Three digit, Three Steps

Input	0	1	2	3	4	5	6	7	8	9
348										
143										
361										
423										
538										
128										
321										
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348										
143										
361										
423										
538										
128										
321										
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143										
361										
423										
538										
128										
321										
543										
366										

Data Structure

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361										
423										
538										
128										
321										
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361	361									
423										
538										
128										
321										
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361	361									
423				423						
538										
128										
321										
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361	361									
423				423						
538									538	
128										
321										
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361	361									
423				423						
538									538	
128									128	
321										
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361	361									
423				423						
538									538	
128									128	
321	321									
543										
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361	361									
423				423						
538									538	
128									128	
321	321									
543				543						
366										

Radix Sort

- STEP 1: Last Digit

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361	361									
423				423						
538									538	
128									128	
321	321									
543				543						
366							366			

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
								348	
			143						
361									
			423						
								538	
								128	
321									
			543						
						366			

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361		348	
			143						
			423						
								538	
								128	
321									
			543						
						366			

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361		348	
		321	143						
			423						
								538	
								128	
			543						
						366			

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361		348	
		321							
				143					
			423						
								538	
								128	
			543						
						366			

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361		348	
		321							
				143					
		423							
								538	
								128	
			$\boxed{543}$						
						366			

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361		348	
		321							
				143					
		423							
				543				538	
								128	
						$\boxed{366}$			

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361		348	
		321							
				143					
		423							
				543				538	
						366		128	

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361			
		321							
				143					
		423							
				543				538	
						366		128	
				348					

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361			
		321							
				143					
		423							
				543					
						366		$\boxed{128}$	
				348					
			538						

Radix Sort

- STEP 2: Second Last Digit

0	1	2	3	4	5	6	7	8	9
						361			
		321							
				143					
		423							
				543					
						366			
				348					
			538						
		128							

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
						361			
		321							
				143					
		423							
				543					
						366			
				348					
			538						
		128							

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321			361			
				143					
		423							
				543					
						366			
				348					
			538						
		128							

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321			361			
				423					
				143					
				543					
						366			
				348					
			538						
		128							

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321			361			
				423					
	128			143					
				543					
						366			
				348					
			538						

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321			361			
				423					
	128			143					
					538				
				543					
						366			
				348					

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321			361			
				423					
	128								
					538				
	143			543					
						366			
				348					

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321			361			
				423					
	128								
					538				
	143								
					543	366			
				348					

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321			361			
				423					
	128								
					538				
	143								
					543	366			
			348						

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321						
				423					
	128								
					538				
	143								
					543	366			
			348						
			361						

Radix Sort

- STEP 3: First Digit

0	1	2	3	4	5	6	7	8	9
			321						
				423					
	128								
					538				
	143								
					543				
			348						
			361						
			366						

Radix Sort

0	1	2	3	4	5	6	7	8	9
			321						
				423					
	128								
					538				
	143								
					543				
			348						
			361						
			366						

128, 143, 321, 348, 361, 366, 423, 538, 543

Radix Sort

- n□total items
- d□the radix (i.e. d=10 for decimal digit, d=26 for)
- s□number of digits/letter

$$A_i = d_{i1}\ d_{i2}\ d_{i3}\ \dots\ d_{is}$$

- Total Pass = s
- In K^{th} pass, compare d_{iK} with each of the d digits
- Then

$$C(n) \le d*s*n$$

- d does not depends on n, so worst case s=n, $C(n) = O(n^2)$
- Best Case, $s = \log_d n$, $C(n) = O(n\log n)$

Radix Sort

- For array implementation, Memory Cost = d*n
- Linked list representation, Memory Coat = 2*n

Any Query?