Sorting algorithms:

Let A be a list of n elements $A_1, A_2, \ldots, A_n$ in memory. *Sorting* A refers

$$A_1 \le A_2 \le A_3 \le \cdots \le A_n$$

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 algorithms:

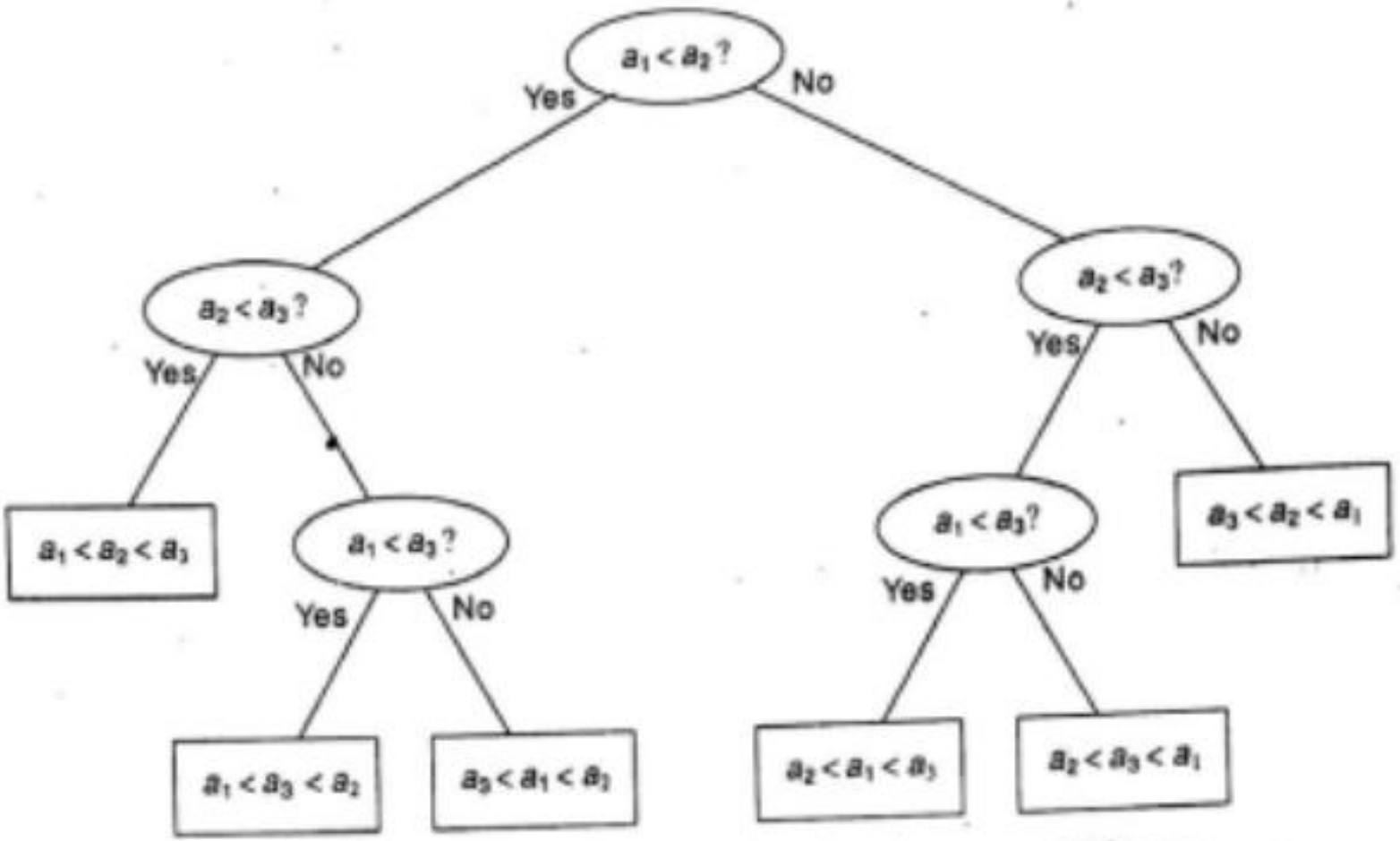


Fig. 9.1 *Decision Tree T for Sorting n = 3 items*

Sorting algorithms:

INSERTION SORT

Suppose an array A with n elements A[1], A[2], …, A[N] is in memory.

Pass 1. A[1] by itself is trivially sorted.
Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.
Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3] so that:
A[1], A[2], A[3], A[4] is sorted.

.....

Pass N. A[N] is inserted into its proper place in A[1], A[2], …, A[N – 1] so that:
A[1], A[2], …, A[N] is sorted.

Sorted after Pass 1

5	3	7	10	0	11	8

Next Item

Sorting algorithms:

INSERTION SORT

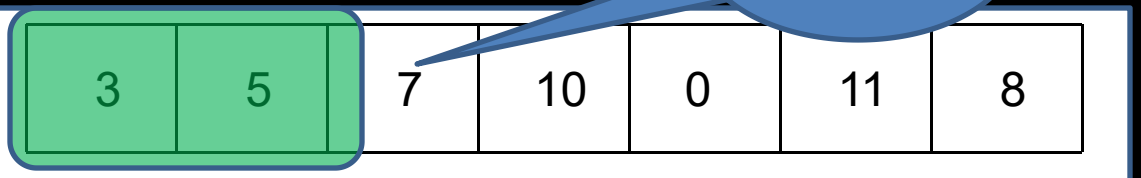
Suppose an array A with n elements A[1], A[2], …, A[N] is in memory.

Pass 1. A[1] by itself is trivially sorted.
Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.
Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3] so that:
A[1], A[2], A[3], A[4] is sorted.

.....

Pass N. A[N] is inserted into its proper place in A[1], A[2], …, A[N – 1] so that:
A[1], A[2], …, A[N] is sorted.

Sorted after Pass 2



Sorting algorithms:

INSERTION SORT

Suppose an array A with n elements A[1], A[2], …, A[N] is in memory.

Pass 1. A[1] by itself is trivially sorted.
Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.
Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3] so that:
A[1], A[2], A[3], A[4] is sorted.

.....

Pass N. A[N] is inserted into its proper place in A[1], A[2], …, A[N – 1] so that:
A[1], A[2], …, A[N] is sorted.

Sorted after Pass 3

3	5	7	10	0	11	8

Next Item

Sorting algorithms:

INSERTION SORT

Suppose an array A with n elements A[1], A[2], …, A[N] is in memory.

Pass 1. A[1] by itself is trivially sorted.
Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.
Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3] so that:
A[1], A[2], A[3], A[4] is sorted.

.....

Pass N. A[N] is inserted into its proper place in A[1], A[2], …, A[N – 1] so that:
A[1], A[2], …, A[N] is sorted.

Next Item

Sorted after Pass 4

3	5	7	10	0	11	8

Sorting algorithms:

INSERTION SORT

Suppose an array A with n elements A[1], A[2], …, A[N] is in memory.

Pass 1. A[1] by itself is trivially sorted.
Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.
Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3] so that:
A[1], A[2], A[3], A[4] is sorted.

.....

Pass N. A[N] is inserted into its proper place in A[1], A[2], …, A[N – 1] so t…t:
A[1], A[2], …, A[N] is sorted.

Next Item

Sorted after Pass 5

0	3	5	7	10	11	8

Sorting algorithms:

INSERTION SORT

Suppose an array A with n elements A[1], A[2], …, A[N] is in memory.

Pass 1. A[1] by itself is trivially sorted.
Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.
Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3] so that: A[1], A[2], A[3], A[4] is sorted.

.....

Pass N. A[N] is inserted into its proper place in A[1], A[2], …, A[N – 1] so that A[1], A[2], …, A[N] is sorted.

Next Item

Sorted after Pass 6

0	3	5	7	10	11	8

Sorting algorithms:

INSERTION SORT

Suppose an array A with n elements A[1], A[2], …, A[N] is in memory.

Pass 1. A[1] by itself is trivially sorted.
Pass 2. A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3. A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that: A[1], A[2], A[3] is sorted.
Pass 4. A[4] is inserted into its proper place in A[1], A[2], A[3] so that:
A[1], A[2], A[3], A[4] is sorted.

.....

Pass N. A[N] is inserted into its proper place in A[1], A[2], …, A[N – 1] so that:
A[1], A[2], …, A[N] is sorted.

Sorted after Pass 7

0	3	5	7	8	10	11

CSE 203: Data Structure

Data Structure

Sorting algorithms:

INSERTION SORT

List of variables:
A: Integer
TEMP: Integer
PTR: Integer
K: Integer
N: Integer

List of loops:
For loop in Step 2
While loop in step 4

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

1. Set A[0] := −∞. [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.

CSE 203: Data Structure

Data Structure

Sorting algorithms:

INSERTION SORT

List of variables:

A: Integer
TEMP: Integer
PTR: Integer
K: Integer

List of loops:

For loop in Step 2
While loop in step 4

```
main(){
int A[10], ITEM, PTR, K, N=9;
```

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

1. Set A[0] := −∞. [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]. [
   (b) Set PTR := PTR − 1.
   [End of loop.]
5. Set A[PTR + 1] := TEMP. [Inserts ele
   [End of Step 2 loop.]
6. Return.

CSE 203: Data Structure

Data Structure

Sorting algorithms:

INSERTION SORT

List of variables:

A: Integer
TEMP: Integer
PTR: Integer
K: Integer

List of loops:

For loop in Step 2
While loop in step 4

```
main(){
int A[10], ITEM, PTR, K, N=9;
A[0]=-99999;
for(K=2;K<=N;K++)                    //Step 2
{
```

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

1. Set A[0] := −∞. [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]. [
   (b) Set PTR := PTR − 1.
   [End of loop.]
5. Set A[PTR + 1] := TEMP. [Inserts ele
   [End of Step 2 loop.]
6. Return.

CSE 203: Data Structure

Data Structure

Sorting algorithms:

INSERTION SORT

List of variables:

A: Integer
TEMP: Integer
PTR: Integer
K: Integer

List of loops:

For loop in Step 2
While loop in step 4

```
main(){
int A[10], ITEM, PTR, K, N=9;
A[0]=-99999;
for(K=2;K<=N;K++)              //Step 2
{
TEMP=A[K];                     //Step 3
PTR=K-1                        //Step3
```

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

1. Set A[0] := −∞. [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]. [
   (b) Set PTR := PTR − 1.
   [End of loop.]
5. Set A[PTR + 1] := TEMP. [Inserts ele
   [End of Step 2 loop.]
6. Return.

CSE 203: Data Structure

Data Structure

Sorting algorithms:

INSERTION SORT

List of variables:

A: Integer
TEMP: Integer
PTR: Integer
K: Integer

List of loops:

For loop in Step 2
While loop in step 4

```
main(){
int A[10], ITEM, PTR, K, N=9;
A[0]=-99999;
for(K=2;K<=N;K++)
{
TEMP=A[K];
PTR=K-1
while(TEMP<A[PTR])                    //Step 4
{
A[PTR+1]=A[PTR];
PTR=PTR-1;
}
```

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

1. Set A[0] := −∞. [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]. [
   (b) Set PTR := PTR − 1.
   [End of loop.]
5. Set A[PTR + 1] := TEMP. [Inserts ele
   [End of Step 2 loop.]
6. Return.

CSE 203: Data Structure

Data Structure

Sorting algorithms:

INSERTION SORT

List of variables:

A: Integer
TEMP: Integer
PTR: Integer
K: Integer

List of loops:

For loop in Step 2
While loop in step 4

```
main(){
int A[10], ITEM, PTR, K, N=9;
A[0]=-99999;
for(K=2;K<=N;K++)
{
TEMP=A[K];
PTR=K-1
while(TEMP<A[PTR])
{
A[PTR+1]=A[PTR];
PTR=PTR-1;
}
A[PTR+1]=TEMP;                    //Step 5
}
}                                 //Step 6
```

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

1. Set A[0] := −∞. [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]. [
   (b) Set PTR := PTR − 1.
   [End of loop.]
5. Set A[PTR + 1] := TEMP. [Inserts ele
   [End of Step 2 loop.]
6. Return.

Sorting algorithms: SELECTION SORT

Pass 1. Find the location LOC of the smallest in the list of N elements A[1], A[2], ..., A[N], and then interchange A[LOC] and A[1]. Then: A[1] is sorted.

Pass 2. Find the location LOC of the smallest in the sublist of N – 1 elements A[2], A[3], ..., A[N], and then interchange A[LOC] and A[2]. Then: A[1], A[2] is sorted, since A[1] ≤ A[2].

pass 3. Find the location LOC of the smallest in the sublist of N – 2 elements A[3], A[4], ..., A[N], and then interchange A[LOC] and A[3]. Then: A[1], A[2], ..., A[3] is sorted, since A[2] ≤ A[3].

... .

... .

Pass N – 1. Find the location LOC of the smaller of the elements A[N – 1], A[N], and then interchange A[LOC] and A[N – 1]. Then: A[1], A[2], ..., A[N] is sorted, since A[N – 1] ≤ A[N].

Thus A is sorted after N – 1 passes.

Sorting algorithms: SELECTION SORT

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*

Sorting algorithms: MERGING

Suppose **A** is a sorted list with **r** elements and **B** is another sorted list with **s** elements
Now make a combine and sorted list of these **r+s** elements.

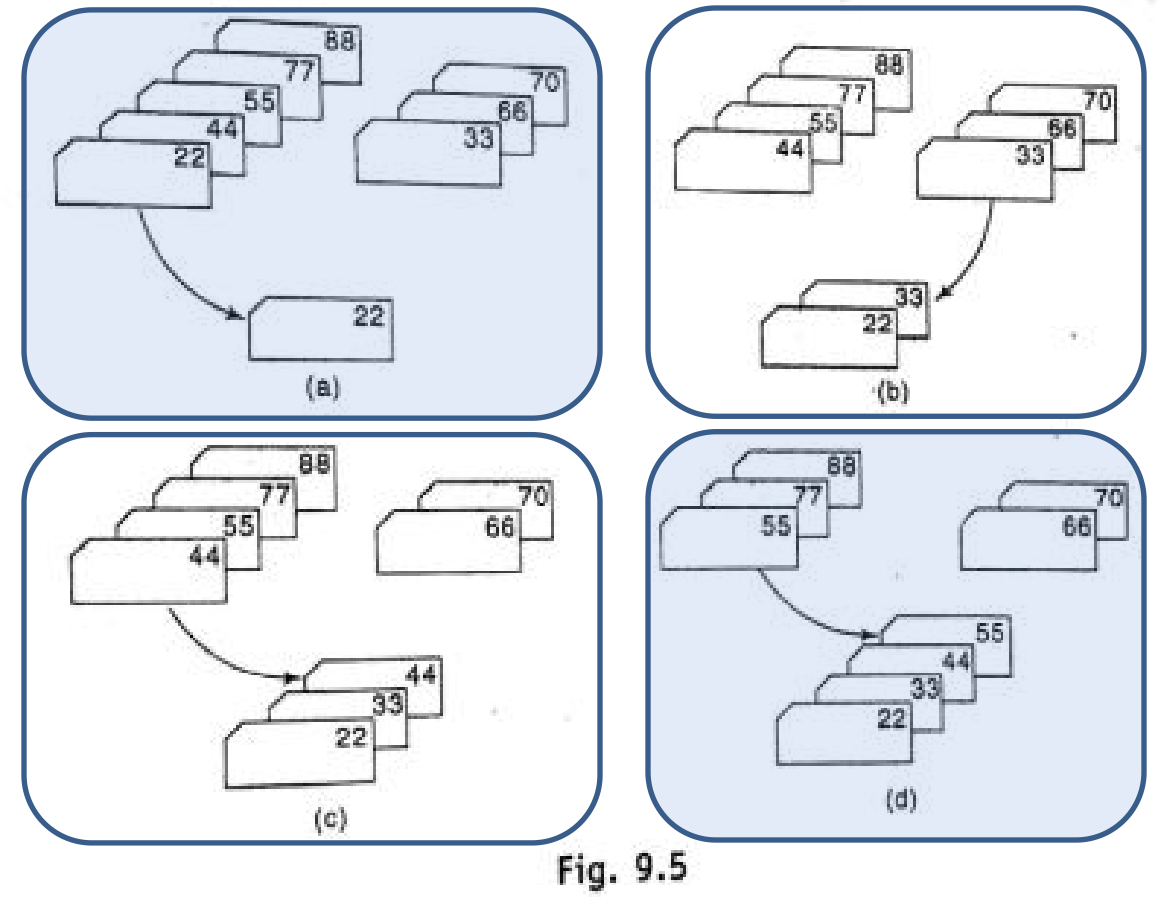


Fig. 9.5

Sorting algorithms: MERGE-SORT

Example 9.7

Suppose the array A contains 14 elements as follows:

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

Each pass of the merge-sort algorithm will start at the beginning of the array A and merge pairs of sorted subarrays as follows:

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

The original array A is now sorted.

Sorting algorithms: RADIX SORT

... phases, as pictured in

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

(a) First pass

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

(b) Second pass

Sorting algorithms: RADIX SORT

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

(b) Second pass

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

(c) Third pass

Sorting algorithms: HASHING

function H from the set K of keys into the set L of memory addresses. Such a function.

$$H: K \rightarrow L$$

is called a *hash function* or *hashing function*. Unfortunately, such a function H may not yield distinct values: it is possible that two different keys k_1 and k_2 will yield the same hash address. This situation is called *collision*, and some method must be used to resolve it. Accordingly, the topic of hashing is divided into two parts: (1) hash functions and (2) collision resolutions. We discuss these two parts separately.

Sorting algorithms: HASHING

computer.

(a) *Division method.* Choose a number m larger than the number n of keys in K. (The number m is usually chosen to be a prime number or a number without small divisors, since this frequently minimizes the number of collisions.) The hash function H is defined by

$$H(k) = k \pmod{m} \qquad \text{or} \qquad H(k) = k \pmod{m} + 1$$

Here $k \pmod{m}$ denotes the remainder when k is divided by m. The second formula is used when we want the hash addresses to range from 1 to m rather than from 0 to $m - 1$.

(b) *Midsquare method.* The key k is squared. Then the hash function H is defined by

$$H(k) = l$$

where l is obtained by deleting digits from both ends of k^2. We emphasize that the same positions of k^2 must be used for all of the keys.

(c) *Folding method.* The key k is partitioned into a number of parts, $k_1, \ldots, k_r$, where each part, except possibly the last, has the same number of digits as the required address. Then the parts are added together, ignoring the last carry. That is,

$$H(k) = k_1 + k_2 + \cdots + k_r$$

where the leading-digit carries, if any, are ignored. Sometimes, for extra "milling," the even-numbered parts, $k_2, k_4, \ldots$, are each reversed before the addition.

End of the Course

CSE 203: Data Structure

Data Structure

CSE 203: Data Structure

Data Structure