Unit - IV

Searching & Sorting

Searching is of 2 types:- (Based on type of files)
- Internal
- External → (when data resides in sec. memory & very vast.)
  → data is less, so main memory me hi honge operation.

Types:-
- Linear search / Sequential search.
- Binary search.

Conditions:-
- Successful search
- Unsuccessful search

Algo for linear search:- (P.T.O)

(1) ~~If~~ [sea == a(i)]
{
(2) Set POS = -1 & i = 0
Repeat step (3) while i <= N.
if A[i] = val
Set POS = i
Print POS.
Go to step (5).

Algo for Binary search:-

Binary (A, LB, UB, Item, Loc, Mid, BEG, END)

Initialise variable.

Set BEG = LB, END = UB.

MID = INT (BEG + END)/2.

(2) → Repeat step (3) & (4) while BEG <= END & A[Mid] ≠ item.

(3) if item < A[Mid]:
then set End = Mid - 1;
else Set BEG = MID + 1. (end of if)

(4) Set Mid = int (BEG + END)/2.
(end of step 2 & loop)

(5) if A[Mid] = Item then set Loc = Mid.
else Set Loc = Null.
(end of if)

(6) Exit.

Pass-1.

35
30
50
65
25
15
55
85

Pass-2

~~35~~	30	30	30	30
~~30~~	35	35	35	35
50	50	50	50	50
65	65	65	65	25
25	25	25	25	65
15	15	15	15	15
55	55	55	55	55
85	85	85	85	85

30 35 50 25 15 65 55 85.

30 35 50 25 15 55 65 85

Pass-3:

30 35 50 25 15 55 65 85

30 35 50 25 15 55 65 85

30 35 25 50 15 55 65 85

30 35 25 15 50 55 65 85.

30 35 25 15 50 55 65 85

30 35 25 15 50 55 65 85

30 35 25 15 50 55 65 85.

Pass-4:

30 35 25 15 50 55 65 85

30 25 35 15 50 55 65 85.

30 25 15 35 50 55 65 85.

Pass-5:

25 30 15 35 50 55 65 85

25 15 30 35 50 55 65 85

25 15 30 35 50 55 65 85

25 15 30 35 50 55 65 85.

25 15 30 35 50 55 65 85

25 15 30 35 50 55 65 85.

Pass-6:

15 25 30 35 50 55 65 85.

15 25 30 35 50 55 65 85

15 25 30 35 50 55 65 85.

15 25 30 35 50 55 65 85.

15 25 30 35 50 55 65 85

15 25 30 35 50 55 65 85

15 25 30 35 50 55 65 85

Pass 7:-
15 25 30 35 50 55 65 85
15 25 30 35 50 55 65 85
15 25 30 35 50 55 65 85

15 25 30 35 50 55 65 85
15 25 30 35 50 55 65 85

15 25 30 35 50 55 65 85
15 25 30 35 50 55 65 85

32 57 27 85 66 23 13 57

Pass 1:- 32 57 27 85 66 23 13 57
32 27 57 85 66 23 13 57

32 27 57 85 66 23 13 57
32 27 57 66 85 23 13 57
32 27 57 66 23 85 13 57

32 27 57 66 23 13 85 57

32 27 57 66 23 13 57 85

Pass 2:- 27 32 57 66 23 13 57 85
27 32 57 66 23 13 57 85
27 32 57 66 23 13 57 85

27 32 57 23 66 13 57 85

27 32 57 23 13 66 57 85

27 32 57 23 13 57 66 85

27 32 57 23 13 57 66 85

Pass 3:- 27 32 57 23 13 57 66 85
27 32 57 23 13 57 66 85

27 32 23 57 13 57 66 85
27 32 23 13 57 57 66 85

27 32 23 13 57 57 66 85
27 32 23 13 57 57 66 85
27 32 23 13 57 57 66 85

Pass 4:- 27 32 23 13 57 57 66 85

27 23 32 13 57 57 66 85

27 23 13 32 57 57 66 85

27 23 13 32 57 57 66 85

27 23 13 32 57 57 66 85

27 23 13 32 57 57 66 85

27 23 13 32 57 57 66 85

Pass 5:- 27 23 13 32 57 57 66 85.

23 27 13 32 57 57 66 85

23 13 27 32 57 57 66 85

23 13 27 32 57 57 66 85

23 13 27 32 57 57 66 85

23 13 27 32 57 57 66 85

Pass 6:- 13 23 27 32 57 57 66 85

13 23 27 32 57 57 66 85

13 23 27 32 57 57 66 85

13 23 27 32 57 57 66 85

:- Algorith :-

Bubble Sort (n, list, ptr, k, temp)

(1.) Repeat step (2) & (3) for K=1 to N-1.

(2.) Set PTR=1.

(3.) Repeat while PTR ≤ N-K.

(a.) if DATA[PTR] > DATA[PTR+1].

{ temp = DATA[PTR]

DATA[PTR] = DATA[PTR+1].

DATA[PTR+1] = temp.

}

(end of if)

(b.) Set PTR = PTR+1.

(end of inner loop).

end of step (1) loop

(2.) exit.

Selection sort :-

K=1 77 33 44 11 88 22 66 55.

LOC=4 [1] [2] [3] [4] [5] [6] [7] [8].

K=2 11 33 44 77 88 22 66 55

LOC=6

K=3 11 22 44 77 88 33 66 55

LOC=6 #

K=4 11 22 33 77 88 44 66 55.

LOC=6

K=5 11 22 33 44 88 77 66 55

LOC=8

K=6 11 22 33 44 55 77 66 88.
LOC=7.

K=7 11 22 33 44 55 66 77 88
LOC=7

→ 11 22 33 44 55 66 77 88

II.

K=1 25 11 13 42 17 90 85 7 10 20
LOC=8

K=2 7 11 13 42 17 90 85 25 10 20
LOC=9

K=3 7 10 13 42 17 90 85 25 11 20
LOC=9

K=4 7 10 11 42 17 90 85 25 13 20
LOC=9

K=5 7 10 11 13 17 90 85 25 42 20.
LOC=25

K=6 7 10 11 13 17 90 85 25 42 20
LOC=10.

7 10 11 13 17 20 85 25 42 90.

K=7
LOC=8

K=8 7 10 11 13 17 20 25 85 42 90
LOC=9

K=9 7 10 11 13 17 20 25 42 85 90
LOC=9

Algorithm for selection sort :-

Min (A, K, N, LOC)

(1) Set Min = A[K] & LOC = K.

(2) Repeat for j = K+1, K+2, K+3, K+4, ---N.

if Min > A[j] & LOC = j

j = J+1

(end of loop).

(3) Return.

Selection (A, N, ---).

(1) Repeat step (2) & (3) for K=1, ---N-1

(2) Call Min (A, K, N, LOC).

(3) Interchange A[K] & A[LOC].

temp = A[K]
A[K] = A[LOC]
A[LOC] = temp.
(end of step 1 & loop)

(4) Exit.

Insertion:-

(1) Set A[0] = -1.

(2) Repeat step (3) to (5) for K = 1, 2, 3, ... (K: iterative variable)

(3) Set Temp = A[K] & 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.
(insert element at proper place).

[end of step 1 & loop]

(6) Return.

Dry-run:-

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

1 7 3 2 10 5 0 8.

Pass	A[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
1	-1	1	7	3	2	10	5	0	8
2		1	7	3	2	10	5	0	8
3		1	3	7	2	10	5	0	8
4		1	2	3	7	10	5	0	8
5		1	2	3	7	10	5	0	8
6		1	2	3	5	7	10	0	8
7		0	1	2	3	5	7	10	8
8		0	1	2	3	5	7	8	

→ Divide & conquer method

Quick Sort: (Appl^{n} of Stack)

Quick(A, N, BEG, END, LEFT, RIGHT, LOC).

→ BEG & END contains boundary value of sublist of A to which this algorithm will be applied.

→ LOC keeps track of position of 1st element A[BEG]. of the sub-list during iterations.

→ Left & right contains the boundary value of the list of the element that has not been scanned.

(1.) Set Left = BEG, Right = END & LOC = BEG.

(2.) Scan from Right to left.

(a.) Repeat while A[LOC] ≤ A[Right] and LOC ≠ Right

Set Right = Right - 1.

[end of loop].

(b.) If LOC = Right then Return

(c.) if A[LOC] > A[Right] then

(i.) Interchange A[LOC] & A[Right].

Temp = A[LOC]
A[LOC] = A[Right].
A[Right] = Temp.

(ii.) Set LOC = Right.

(iii.) Go to step (3).

[end of if]

(3.) Scan form left to Right.

(a.) Repeat while A[Left] ≤ A[LOC] and Left ≠ LOC.

Set Left = Left + 1.

[end of loop].

(b.) If LOC = Left then return.

(c.) if

2.) if A[left] > A[loc] then.

(ii) Interchange A[left] & A[loc].

temp = A[loc].
A[loc] = A[left]
A[left] = temp.

(iii) set LOC = left.

(iii.) Go to step (2)
(end of if structure)

Algo - Quicksort (this algo sort array with N element).

(1.) (Initialize) TOP = NULL.

(2) [Push boundary value A onto the stack when A has 2 or more element].

if N > 1 then, TOP = TOP + 1.

lower[1] = 1.

Upper[1] = N.

(3.) Repeat step 4 to 7 while TOP ≠ NULL (backside note)

RADIX SORT. (more no. of data)

Eg. Roll no list.

Google — check from digit place.

one's place — ten's place

Eg:- 10, 15, 1, 60, 5, 100, 25, 50.

Step I:- find the largest no. ie, & count no. of digit in that no. if no is of n digit then no. of pass = n.

II:- Convert all no's (input) into n digit no.

In this case n = 3 so,

010, 015, 001, 060, 005, 100, 025, 050.

hint: ↓ unit, ↓ ten's

III:- classify no. according to unit digit place.

digit contain no's form.
0 - 9.

assign label 0
i
9. ↑ smallest

These nos. contains 0 in unit place assign them into lable.

Queue for digit.

Pass I:- 0 - 010, 060, 100, 050.
1 - 001
2 - ×
3 - ×
4 - ×
5 - 015, 005, 025.
6 - ×
7 - ×
8 - ×
9 - ×

Output of Pass-I

Starting with Ist queue or table

010, 060, 100, 050, 001, 015, 005, 025

for pass II we've input, ie, output of Pass I.

Pass II :- 010, 060, 100, 050, 001, 015, 005, 025
Check ten's place.

0 - 100, 001, 005
1 - 010, 015
2 - 025
3 - ×
4 - ×
5 - 050
6 - 060
7 - ×
8 - ×
9 - ×

O/p of Pass II:-

100, 001, 005, 010, 015, 025, 050, 060

↓ This will be I/p for Pass III.

Pass III:- 100, 001, 005, 010, 015, 025, 050, 060

0 - 001, 005, 010, 015, 025, 050, 060
1 - 100
2 - ×
3 - ×
4 - ×
5 - ×
6 - ×
7 - ×
8 - ×
9 - ×

O/p - 001, 005, 010, 015, 025, 050, 060, 100.

Ques:- 348, 143, 361, 423, 538, 128, 321, 543, 366, 4281, 2163.

Largest no. = 4 digit (4281)

0348, 0143, 0361, 0423, 0538, 0128, 0321, 0543, 0366, 4281, 2163.

Pass I:- Check unit's digit:-

0 - ✓
1 - 0361, 0321, 4281
2 - ×
3 - 0143, 0423, 0543, 2163
4 - ×
5 - ✓

6 - 0366
7 - ×
8 - 0348, 0538, 0128
9 - ×

O/p of Pass I:- 0361, 0321, 4281, 0143, 0423, 0543, 2163, 0366, 0348, 0538, 0128

Input for Pass II:-

0 - ✓
1 - ×
2 - 0321, 0423, 0128
3 - 0538
4 - 0143, 0543, 0348
5 - ×
6 - 0361, 2163, 0366
7 - ×
8 - 4281
9 - ×

O/p:- 0321, 0423, 0128, 0538, 0143, 0543, 0348, 0361, 2163, 0366, 4281,

I/p for Pass III:- 0321, 0423, 0128, 0538, 0143, 0543, 0348, 0361, 2163, 0366, 4281.

Pass III :-

0 - >
1 - 0128, 0143, 2163,
2 - 4281
3 - 0321, 0348, 0361, 0366,
4 - 0423
5 - 0538, 0543
6 - >
7 - >
8 - >
9 - >

O/p :-

0128, 0143, 2163, 4281, 0321, 0348, 0361,
0366, 0423, 0538, 0543

Pass IV :-

0 - 0128, 0143, 0321, 0348, 0361, 0366, 0423, 0538, 0543.
1 - >
2 - 2163
3 - >
4 - 4281
5 - >
6 - >
7 - >
8 - >
9 - >

O/p :- 0128, 0143, 0321, 0348, 0361, 0366, 0423, 0538, 0543, 2163, 4281.

Merge Sort :-

1st make pair of 2 then apply sorting after that make group of 4 then apply sorting then group of 8 & so on with power of 2.

Eg :- 5, 2, 4, 6, 3, 1, 6, 2

2, 5, 4, 6, 1, 3, 2, 6

2, 4, 5, 6 1, 2, 3, 6

1, 2, 2, 3, 4, 5, 6, 6 - sorted list

Pairing

Eg :- of 2 :- 66, 33, 40, 20, 55, 88, 60, 11, 80, 20, 50, 44, 70

Pairing of 4 :- 33, 66, 20, 40, 55, 88, 11, 60, 20, 80, 44, 50, 70

Pairing of 8 :- 20, 33, 40, 66, 11, 55, 60, 88, 20, 44, 50, 80, 70

Pairing of 16:-

11, 20, 33, 40, 55, 60, 66, 88. 20, 44, 50, 70, 77

11, 20, 20, 33, 40, 44, 50, 55, 60, 66, 70, 77, 88

Explanation of Quick Sort :-

44 33 11 55 77 90 40 60 99 22 88 66

LB (BEG) Left Loc(fin)

UB END

44 ≤ 66 ✓

44 ≤ 88 ✓

44 ≤ 22 ✗ (swap).

22 33 11 55 77 90 40 60 99 44 88 66

L. R (LOC) fin

22 ≤ 44 ✓

33 ≤ 44 ✓

11 ≤ 44 ✓

55 ≤ 44 ✗ (swap).

22 33 11 44 77 90 40 60 99 55 88 66

L (fin) (LOC). R

44 ≤ 55 ✓

44 ≤ 99 ✓

44 ≤ 60 ✓

44 ≤ 40 ✗ (swap).

22 33 11 40 77 90 44 60 99 55 88 66

L R (fin) (LOC)

44 44 ≤ 40.

40 ≤ 44 ✓

77 ≤ 44 ✗ (swap).

22 33 11 40 44 90 77 60 99 55 88 66

L (fin) LOC R

44 ≤ 77. ✓

44 ≤ 90 ~~90 ≤ 77. ✗ (swap)~~ ✓

~~22 33 11 40 44 77 90 60 99 55 88 66~~

22 33 11 40 44 90 77 60 99 55 88 66

L1 L R L2

[i.e, LOC = Right so, loop terminates]

90 77 60 99 55 88 66.
↑ ↑
Beg Right
Left
LOC (fin)

90 ≤ 66 × (swap).

66 77 60 99 55 88 90.
↑ ↑
Left Right
(LOC).

66 ≤ 90 ✓
77 ≤ 90 ✓
60 ≤ 90 ✓
99 ≤ 90 × (swap).

66 77 60 90 55 88 99
↑ ↑
L R
(fin)
LOC

90 ≤ 99 ✓
90 ≤ 88 × (swap)

66 77 60 88 55 90 99.
↑ ↑
L R (fin)
LOC.

88 ≤ 90. ✓

55 ≤ 90. ✓

66 77 60 88 55 90 99.
↑ ↑ ↑
L 3. R. R.
LOC
(fin)

66 ≤ 55 × (swap).

55 77 60 88 66.
↑ ↑
L. R
(fin).

55 ≤ 66 ✓
77 ≤ 66 × (swap).

55 66 60 88 77.
↑ ↑
L R.
LOC (fin)

66 ≤ 77 ✓
66 ≤ 88. ✓
66 ≤ 60 × (swap).

55 60 66 88 77
↑ ↑ 5
4. L R.
(fin).
LOC.

60 ≤ 66.

88 77.
↑ ↑
L R
LOC (fin)

77 ≤ 88 ✓

88 ≤ 77 × (swap)

77 88 R.

88 - fix.

77 88

R
55 60
BEG LOC (left)
Right

55 ≤ 60 ✓

22 33 11 40
left (LOC)
Right

22 ≤ 40 ✓

~~88~~ 22 ≤ 11 ✗ (swap)

~~22~~

11 33 22 40
L
R (LOC)

11 ≤ 22 (✓)

33 ≤ 22 (✗) swap.

R
11 33 22 33 40
L
LOC (fix)
R

22 ≤ 33 (✓)

R
11 22 33 40
left (LOC)
Right

~~11 22~~ 33 ≤ 40 (✓)

11 22 33 40 →

Step (4):-

[Pop sublist from stack]

Set BEG = Lower[TOP], END = UPPER[TOP],

TOP = TOP - 1.

(5) Call quick[A, N, BEG, END, LOC].

(6) [Push left sublist on stack].

if BEG < LOC - 1 then,

TOP = TOP + 1, Lower[TOP] = BEG.

UPPER[TOP] = LOC - 1.

[end of if ....).

) Push Right sublist on stack

if LOC+1 < END then

TOP = TOP + 1 , LOWER[TOP] = LOC+1

UPPER[TOP] = END

(end of if)

(end of step (3) & loop).

) EXIT.

10, 5, 63, 27, 88, 3, 15, 43, 4, 1

↑ BEG (Left) (fin)

↑ END Right

10 < 1. ✗ (swap?

1 5 63 27 88 3 15 4 3 4 10

↑ fin LOC

~~1 < 10~~ ✓

1 < 4 ✓

1 < 43 ✓

1 < ~~15~~ ✓.

to develop efficiency - sorting logic.

~~1 < 3.~~

1 < 10 ✓

5 < 10 ✓

63 < 10 ✗ (swap)

1 5 10 27 88 3 15 43 4 63

↑ LOC ↑ Right

~~15~~ 10 < 63 ✓

10 < 4 ✗ (swap)

1 5 4 27 88 3 15 43 10 63

↑ Left ↑ fin LOC

4 < 10 ✓

27 < 10 ✗ (swap).

1 5 4 10 88 3 15 43 27 63

↑ LOC ↑ Right

10 < 27 ✓

10 < 43 ✓

10 < 15 ✓

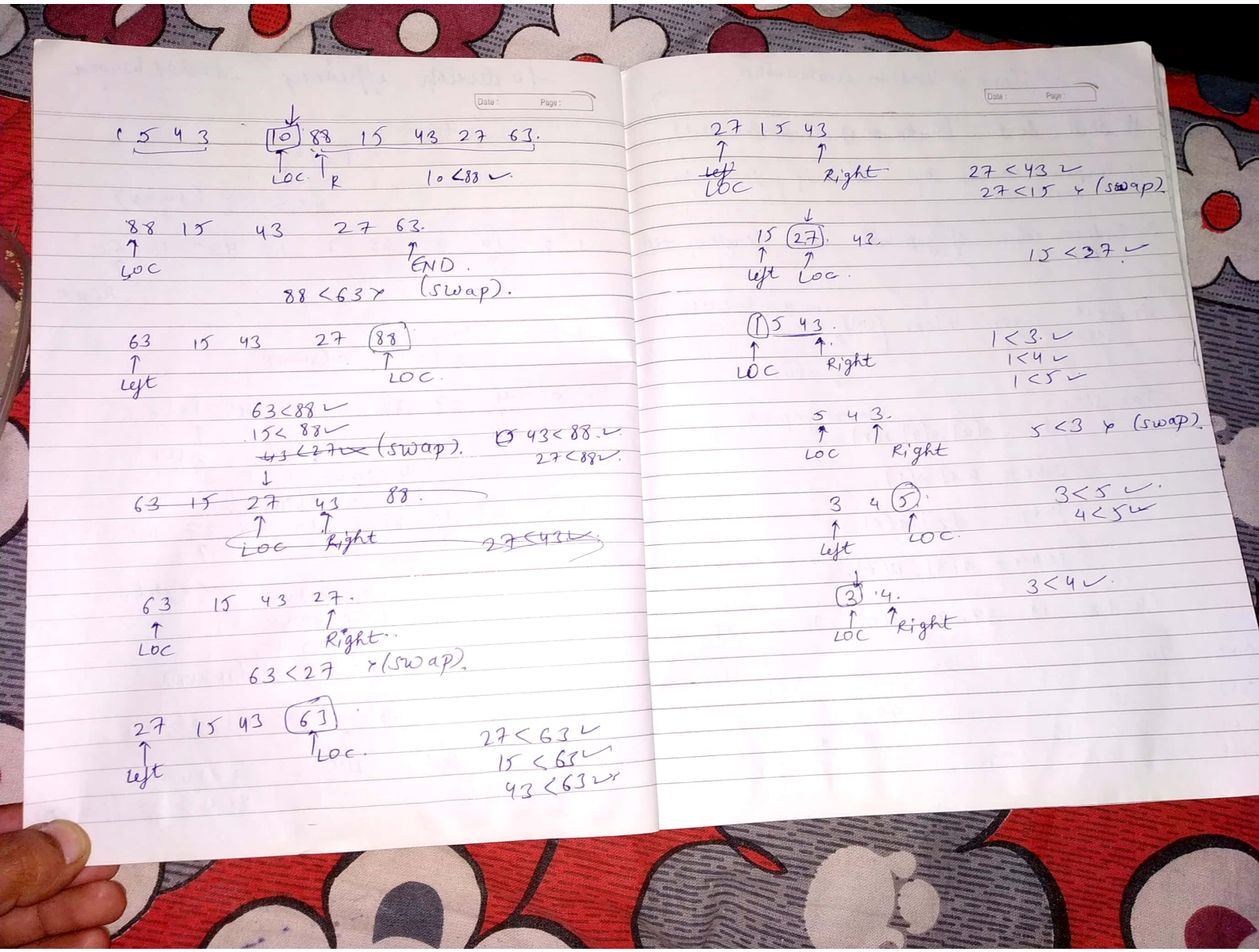
10 < 3 ✗ (swap).

1 5 4 3 88 10 15 43 ~~27~~ 63

↑ Left ↑ LOC

3 < 10. ✓

88 < 10 ✗ (swap).



Gap is used to create sublist.

Shell Sort:- (Extension of Insertion Sort)

1) Compare elements by considering gap where Gap<END.
2) Reduce value of gap with each iteration till Gap = 1.

for 1st iteration :- gap = floor (N / 2) (N = no of the elements)

= 9/2 = 4.5

Pass 1:- ~~floor~~ gap = 4.

Sub1 - a[0], a[4], a[8]

Sub2 - a[1], a[5]

Sub3 - a[2], a[6]

Sub4 - a[3], a[7].

0	1	2	3	4	5	6	7	8
14	18	19	37	23	40	29	30	11

14 23 11

18 40

19 29

37 30.

-1	14,	23	11.
-1	14	23	11
-1	14	23	11
-1	11	14	23.

O/p:-

	0	1	2	3	4	5	6	7	8.
Sub1:-	11				14				23.
Sub2:-		18				40			
Sub3:-			19				29		
Sub4:-				30				37.	

Final Output:-

of Pass 1.

⇒ 11, 18, 19, 30, 14, 40, 29, 37, 23.

0	1	2	3	4	5	6	7	8
11	18	19	30	14	40	29	37	23

gap = floor (n / 2^2) → 2nd pass.

gap = 2.

Sub1:- a[0], a[2] a[4] a[6] a[8]

Sub2:- a[1] a[3] a[5] a[7]

0 1 2 3 4 5 6 7 8.
11 18 19 30 14 40 29 37 23.

11 19 14 29 23

18 30 40 37.

p of Pass-2:-
11 14 19 23 29.

18 30 37 40

inal O/p of Pass 2:-

11, 18, 14, 30, 19, 37, 23, 40, 29.

ss-3:-

$$gap = floor\left(\frac{n}{2^3}\right)$$

gap = 1.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
11	18	14	30	19	37	23	40	29
11	14	18	30	19	37	23	40	29
11	14	18	30	19	37	23	40	29
11	14	18	19	30	37	23	40	29
11	14	18	19	30	37	23	40	29
11	14	18	19	23	30	37	40	29
11	14	18	19	23	29	30	37	40

Hashing (Address searching technique)

Hash Table:-

It is a D.S. which consists of an array in which data is accessed via a special index known as key.

Hash function:-

It is used to store data in Hash table, it establish mapping between set of possible keys & position in hash table.

Outcome of Hash func^{n} is called Hash code / Hash value.

↓

It'll be of Integer type

Application of Hashing:-

Data Dictionary.
Password verification on any portal.
File stored on cloud (virtual storage).
Mapping b/w file name & file path.
Database system of any organisation.
Cryptography.
Symbol table

All Operations are performed with the help of # function.

Division Method :-

Map the key (k) into one of the m slot by taking remainder of k mod m.

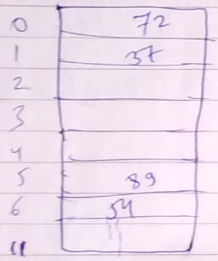
map key ↓ record

m - length of hash table.

54
72
89
37

h(54) = 54 mod 12 = 6.

h(72) = 0
h(89) = 5
h(37) = 1



Total size of m.

Certain values of m must be avoided (power of 2 & power of 10)

Collision :- 2 values (data) having same address.

0 { 160 mod 10
150 mod 10

A = 0.61 (61%)

Multiplication Method :-

Operates in 2 steps :-

(1.) Multiplication method :- x the key by a constant A ; where A is b/w 0 & 1. Then ~~abstract~~ extract the fractional part of kA.

(2) Multiply extracted value by m & take floor / integer of the result.

Example :-

KA — K = 107 M = 50,
A = 0.61

(1) KA = 65.27

(2) m x .27 = 50 x 0.27
= 13.50

floor (13.50) = 13

{ Must be always b/w 0 to 50 }.

Mid-Square Method:-

→ Square the key.
→ Take mid part of the result as an address.

3111 → key
$(3111)^2$ = 9678321

341

Agar se gye toh Mid = 78
else. Mid = 33.
↳ (given hoga ye)

Digit-Folding Method:-

Break the key into small pieces add them & get the hash address.

Truncate from MSB or higher digit based on size of table

323

h(82394564) = 823 + 94 + 564 = 1478.

h(87139465) = 871 + 39 + 465 = 1375.

h(83567271) = 835 + 67 + 271 = 1173.

Highest ASCII code. — (127)

Now if m = 1000.

ie, size of table ✗ address

∴ = 1478
= 375
= 173.

for storing:-

suresh

m = 977 (tablesize)

s + u + r + e + s + h.

→ 115 + 117 + 114 + 101 + 115 + 104 m = 77

= 666.

= 666 % m = 666 % 77

84

If m > data.
so, Multiply data by Highest ASCII code

= 127 × 666
= 84582. % m
= 84582.

When more than 1 key ; map to same # value in # table) collision

Resolution of Collision:-

→ By separate chaining.

→ Open addressing

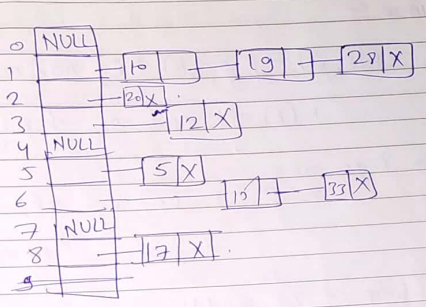
(1.) 5, 28, 19, 15, 20, 33, 12, 17, 10. (m=9)

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

$h(5) = 5 \bmod 9 = 5$

$h(28) = 28 \bmod 9 = 1$
$h(19) = 19 \bmod 9 = 1$
$h(15) = 15 \bmod 9 = 6$
$h(20) = 20 \bmod 9 = 2$
$h(33) = 33 \bmod 9 = 6$
$h(12) = 12 \bmod 9 = 3$
$h(17) = 17 \bmod 9 = 8$
$h(10) = 10 \bmod 9 = 1$.

O/p of H.F - H value



Disadvantage:- Memory wastage. [To store NULL address on link].

(2.) To search the address, when given (original # value) address is already occupied in # table — Probing

(Always in → direc^n)

3 techniques of Probing:-

→ Linear
→ Quadratic
→ Double Hashing

Linear:

$h(k, i) = [h'(k) + i] \bmod m$

$h'(k) = k \bmod m$ for $i = 0, 1, 2, 3, \ldots m-1$

The given keys are:

26, 37, 59, 76, 65, 86,

$m = 11$

$h'(26) = 26 \bmod 11 = 4$

$h'(37) = 37 \bmod 11 = 4$

$h'(59) = 59 \bmod 11 = 4$

$h'(76) = 76 \bmod 11 = 9$. 10

$h'(65) = 65 \bmod 11 = 10$

$h'(86) = 86 \bmod 11 = 9$

$h(k, i) = \{ h(26, 0)$

$= [4 + 0] \bmod 11$

$= 4 \bmod 11$

$\rightarrow T(4)$.

$h(37, 0) = (4 + 0) \bmod 11$

$\rightarrow T(4)$ occupied.

$h(59, 0) = (4$

$h(37, 1) = (4 + 1) \bmod 11$

$= 5 \bmod 11$

$\rightarrow T(5)$.

$h(59, 0) = 5 \bmod 11$

$\rightarrow T[5]$ occupied

$h(59, 1) = T[5]$ occupied

$h(59, 2) = T[6]$.

$h[76, 0] = (10 + 0) \bmod 11$

$= 10 \bmod 11$

$= T(10)$.

$h[65, 0] = 10 \bmod 11$

$\rightarrow T[10]$ occupied

$h[65, 1] = 11 \bmod 11$

$= T(0)$..

$h[86, 0] = 9 \bmod 11$

$\rightarrow T(9)$.

0	65
1	NULL
2	NULL
3	NULL
4	26
5	37
6	59
7	NULL
8	NULL
9	86
10	76

Disadvantage
— Searching tym of index (↑).

To overcome this linear; we'll use

Quadratic :-

$h(k,i) = [h'(k) + c_1 i + c_2 i^2] \bmod m$

where c_1 & c_2 are constant.

$h'(k) = k \bmod m$ for

$i = 0, 2, \ldots m-1$

linear :- $h, h+1, h+2, h+3, \ldots$

quadratic :- $h, h+1, h+4, h+9, \ldots$

Double Hashing :-

↓

Why D# is best?

BCz quad. bhi kahi na kahi linearly kaam krta h.

2 # functions are used ~~to~~ here.