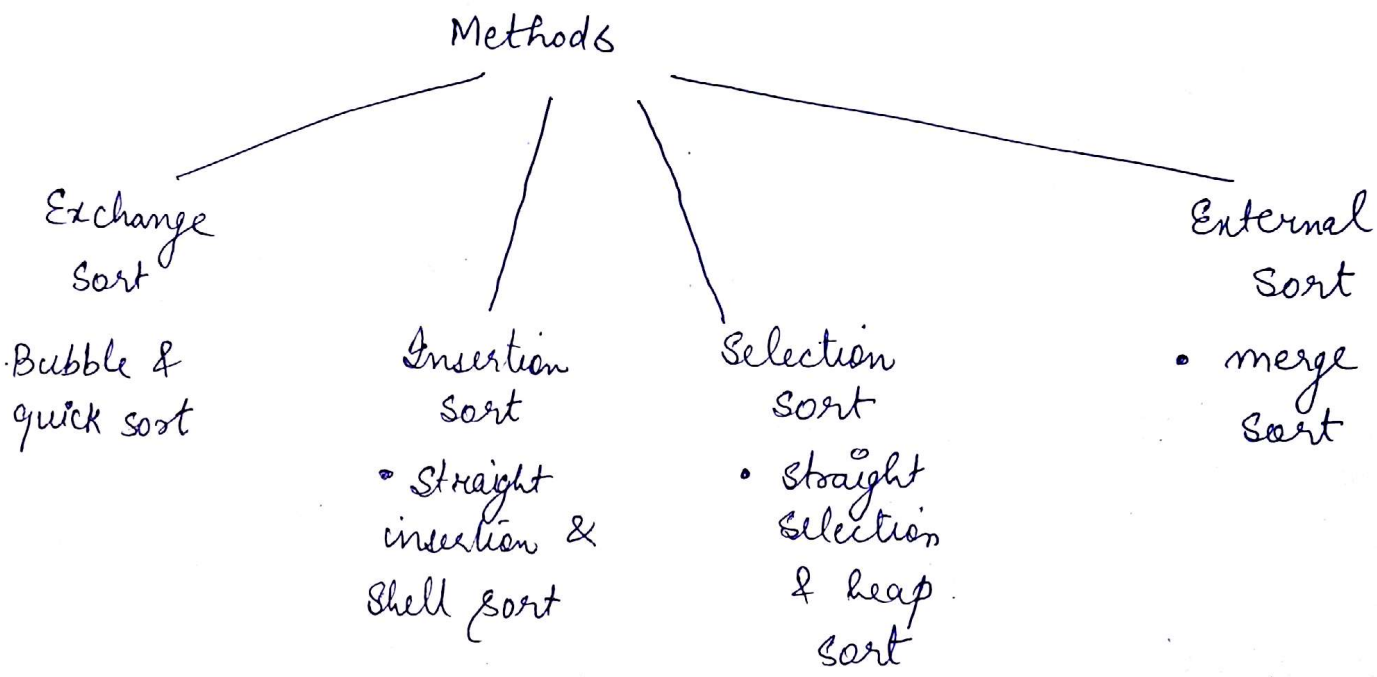
Unit - 4

Sorting

- Sorting refers to arranging some elements in ascending or descending order.



Unit - IV

3/10/17

Date __/__/__
Page _____
STUDY BUDDIES

→ **Bubble Sort (DATA, N):**

1) Repeat steps 2 and 3 for K = 1 to (N-1)
2) Set PTR := 1
3) Repeat while PTR ≤ N-K
   (a) If DATA[PTR] > DATA[PTR+1]
       Interchange both
   (b) Set PTR := PTR+1
       [End of inner loop]
       [End of outer loop]
4) Exit

eg. 32, 51, 27, 85, 66

Pass I: Step-1) (32), (51), 27, 85, 66
2) 32, (27), (51), 85, 66
3) 32, 27, (51), (85), 66
4) 32, 27, 51, (66), (85)

Pass II: Step-1) (27), (32), 51, 66, 85

Ques) 4, 6, 1, 8, 2, 3

Pass I: Step-1) 4, (1), (6), 8, 2, 3
2) 4, 1, (6), (8), 2, 3
3) 4, 1, 6, (2), (8), 3
4) 4, 1, 6, 2, (3), (8)

Complexity

For N elements, there are N-1 passes & in each pass there are N-1 comparisons.

$\therefore$ Complexity $= (N-1)(N-1)$

$= N^2 - 2N + 1$

$= O(n^2)$

Pass II: Step-1) (1, 4) 6, 2, 3, 8:
2) 1, (4, 2) 6, 3, 8
3) 1, 4, 2, (3, 6), 8

Pass III: Step-1) (1, 4) 2, 3, 6, 8
2) 1, (2, 4) 3, 6, 8
3) 1, 2, (3, 4) 6, 8

Max. no. of possible passes = (no. of elements - 1)

- After each pass, the largest value is shifted to the end of the array.

→ Insertion Sort -

eg. 32, 51, 27, 5, 6

Pass-I: A[1] by itself is sorted
II: A[2] is inserted either before or after A[1] such that these two are sorted.
III: A[3] is inserted into its proper place among the three i.e. before A[1] OR b/w A[1] & A[2] or after A[3] such that they are sorted.

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
		37	51	27	5	6
K=1	-∞	37	51	27	5	6
K=2	-∞	32	51	27	5	6
K=3	-∞	27	32	51	5	6
K=4	-∞	5	27	32	51	6
K=5	-∞	5	6	27	32	51

Ques) 77, 33, 44, 11, 88, 22, 66, 55

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

→ First step should have been

-∞ 77 33 44 11 88 22 66 55

& since 77 > -∞, ∴ no interchange or shift occurs in this pass

⇒ total no. of passes = 8

Max. no. of passes = no. of elements

Complexity

- outer loop runs $N-1$ times.
- Max. comparisons $= N-1$

~~∴ in 1st comparison~~

∴ in 1st iteration, no. of comparisons will be 1
in 2nd $= 2$ upto $N-1$.

∴ Total comparisons $= 1+2+3+4 \cdots +(N-1)$

$$= \frac{N(N-1)}{2}$$

$$= O(N^2).$$

Algo:-

1) Set A[0] := -∞
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
   [End of step 2 loop]
6) EXIT

→ Merge sort:-

Select → sort → Merge

10, 11, 9, 6, 3, 1, 2, 4, 14, 16, 18, 17

10, 11; 6, 9, 1, 3, 2, 4, 14, 16, 17, 18

1, 2, 3, 4, 6, 9, 10, 11, 14, 16, 17, 18

Select pairs of 2 elements
Sort each individual pair
Merge two pairs to form a quad.

Complexity of merge Sort

•) The value of subarrays increases by a power of 2, so there cannot be more than $\log_2 N$ pas in the merge sort.

•) Max. comparisons $= N$

∴ Complexity $= O(N \log_2 N)$

•) However, it requires additional space in the form of auxiliary memory.

Ques) 66, 33 ; 40, 22 ; 55, 88 ; 60, 11 ; 18, 20 ; 50, 44 ; 77, 30

33, 66 ; 22, 40 ; 55, 88 ; 11, 60 ; 18, 20 ; 44, 50 ; 30, 77

22, 33, 40, 66 | 11, 55, 60, 88 | 18, 20, 44, 50 | 30, 77

11, 22, 33, 40, 55, 60, 66, 88 | 18, 20, 30, 44, 50, 77

11, 18, 20, 22, 30, 33, 40, 44, 50, 55, 60, 66, 77, 88

→ Selection Sort -

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

Pass-II: Find the location of LOC of the smallest element in the sublist of (N-1) elements - A[2], A[3] . . . . A[N] and then interchange A[LOC] and A[2].
Thus, A[1] & A[2] are sorted.
And so on.

Max. no. of passes = (no. of element - 1)

eg:

K	LOC	A[1]	A[2]	A[3]	A[4]	A[5]
		6	2	8	1	5
1	4	(6)	2	8	(1)	5
2	2	1	(2)	8	6	5
3	5	1	2	(8)	6	(5)
4	4	1	2	5	(6)	8
5	5	1	2	5	6	(8)

Ques) 77, 33, 44, 11, 88, 22, 66, 55

K	LOC	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
		77	33	44	11	88	22	66	55
1	4	(77)	33	44	(11)	88	22	66	55
2	6	11	(33)	44	77	88	(22)	66	55
3	6	11	22	(44)	77	88	(33)	66	55
4	6	11	22	33	(77)	88	(44)	66	55
5	8	11	22	33	44	(88)	77	66	(55)
6	7	11	22	33	44	55	(77)	(66)	88
7	8	11	22	33	44	55	66	77	(88)

Complexity

No. of comparisons in 1st iteration = $n-1$

2nd " = $n-2$

Total = $(n-1) + (n-2) + \dots 3+2+1$

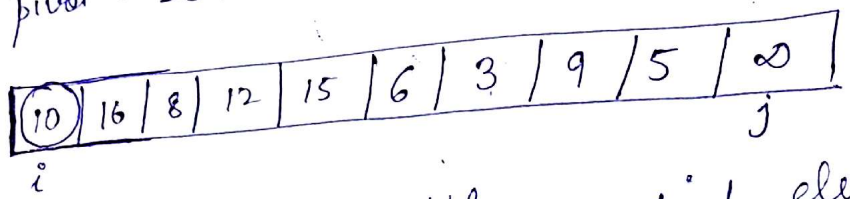
$= O(N^2)$

Quick Sort (Divide & Conquer)

10, 16, 8, 12, 15, 6, 3, 9, 5

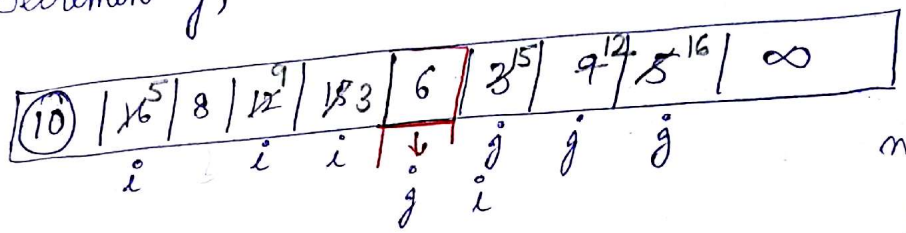
Step 1: choose a pivot element. Here we are taking it to be the ~~first~~ first element. i.e., 10.

∴ pivot = 10.



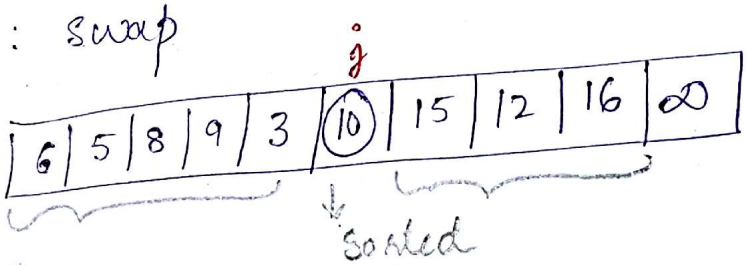
Step 2: Increment i, until you find element > pivot.

Step 3: Decrement j, " " " " < pivot.



now i has crossed j, which means that is the posn of pivo

step 4: swap



Step 5: Perform quicksort recursively on either side, to get the desired results.

Shell Sort

- In this sorting algorithm, we compare elements that are a distance apart rather than adjacent.
- For N elements, we start with a value gap < N.

  Gap = $\lfloor (\frac{N}{2}) \rfloor$, where N = no. of elements

- We reduce gap in each pass, till it reaches the value 1.
- In the last pass, shell sort is like insertion sort.

Like: in pass 1 = gap_0 → floor $(\frac{N}{2})$

pass 2 = gap_1 = floor $(\frac{gap_0}{2})$

pass 3 = gap_2 = floor $(\frac{gap_1}{2})$

eg. 77, 62, 14, 9, 30, 21, 80, 25, 70, 55

So, N = 10 elements

gap = floor $(\frac{10}{2})$ = floor(5) = 5

pass 1: (77) (62) (14) 9 30 (21) (80) (25) 70 55

↓ ↓ ↓ ↓

Swap → 21 62 14 9 30 77 80 25 70 55

pass 2: gap = floor $(\frac{5}{2})$ = floor(2.5) = 2

(21) (62) (14) (9) (30) (77) (80) (25) (70) (55)

place: 14 9 21 25 30 55 70 62 80 77

pass 3: gap = floor $(\frac{2}{1})$ = 1

14 9 21 25 30 55 70 62 80 77

= 9 14 21 25 30 55 62 70 77 80

ans.

→ Radix Sort-

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

No. of passes = no. of digits in values to be sorted

I^st pass sorts / considers ones place

	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			

Arrange the values in sorted order of ones place digits to be used for second pass.

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

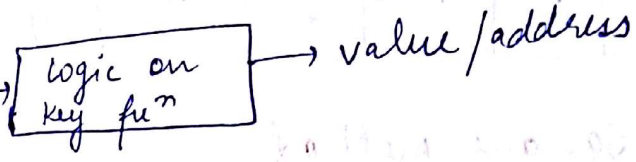
	0	1	2	3	4	5	6	7	8	9
321				321						
423					423					
128		128								
538						538				
143		143								
543						543				
348				348						
361				361						
366				366						

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

Hashing

- It is a searching technique that rely on the concept of Hash tables.

eg. Dictionary



Hashing methods:

1. Direct hashing
2. Subtraction method
3. Modulo division
4. Mid square
5. Pseudorandom hashing
6. Folding method

1. Direct Hashing

The key is addressed without any algorithmic manipulation.

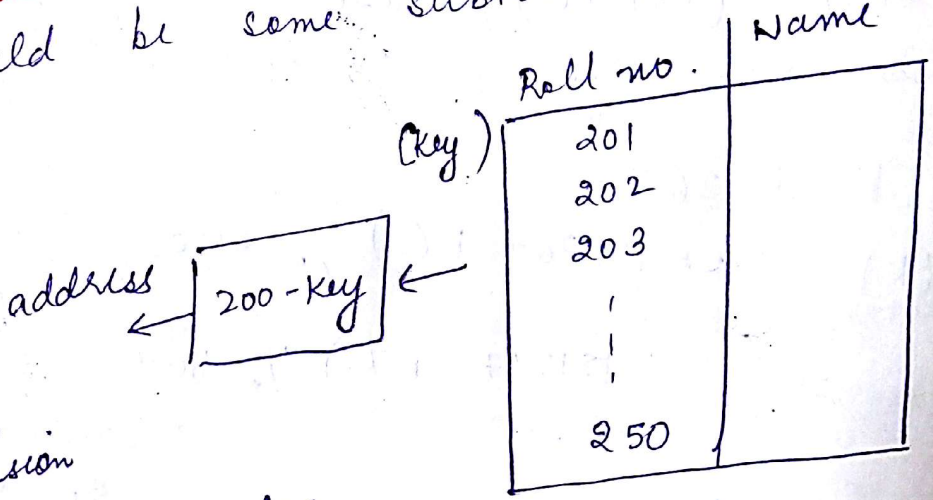
K → address

Roll no.	Name
1	AB
2	ABC
.	.
.	.
100	.

* No collision
* Not suitable for large DB.

2. Subtraction method

Function would be some subtraction value.



* No collision
* Suitable for small lists.

3. Modulo - Division method (division remainder)

Hash address = key modulus size.

Key	234	431	947	981	792	685
H(K)	0	2	11	6	12	9

take size = 13 (remainders)

4. Mid - Square method

The key is squared & address selected from middle of the squared no.

$h(k)$ = middle digits of k^2.

eg. $h(1150130) = 1345423617100$

$h(1001) = 1010001$

⇒ The drawback is that it does not distribute key uniformly.

5. Pseudorandom method

A random no is generated using key using the modulo division method to generate hash key.

$h(k) = (a * k + c) \%$ size

where a = coeff, c = constant.

eg. k = 121267

$h(k) = (a * 121267 + c) \%$ size

$= (17 * 121267 + 7) \% 307$ (assuming values of a, c & size)

$= 4$.

Collision Resolution

A collision is the event that occurs when a hashing algorithm produces an address ~~for indexing~~ and that address already exists.

Collision Resolution Techniques.

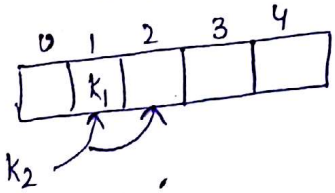
This technique is an attempt to place the record elsewhere in the table when collision occurs.

1. open addressing
   - Linear probing
   - Quadratic probing
2. chaining
3. Bucket hashing
4. Pseudorandom probing.

(1) Open addressing

In open addressing, when address produced by hash fn $h(K)$ has no more space for insertion, an empty slot other than $h(K)$ is located in hash table.

eg.



* Linear probing

$H(K) = K \bmod \text{size}$

21
22
11
32

K mod 5

4	32
3	11
2	22
1	21
0	

$11 \% 5 = 1$ (already full)
∴ Then go to next available slot.
$32 \% 5 = 2$ (full)

Quadratic probe

eg. 21, 22, 11, 32 — K mod 5

0	11
1	21
2	22
3	32
4	

$21\%5 = 1$ no collision

$22\%5 = 2$ " "

$11\%5 = 1$ collision

$H_i(x) = [h(K) + f(i)] \bmod size$

$H_i(x) = (h(K) + i^2) \bmod size$

$H_1(x) = (11 + 1^2) \bmod 5$

$= 12 \bmod 5 = 2$ (again collision)

$H_2(x) = (11 + 2^2) \bmod 5$

$= (11 + 4) \bmod 5$

$= 15 \bmod 5 = 0.$

∴ now 11 will be placed at 0.

$32\%5 = 2$ (collision)

So, $H_1 = (32 + 1^2) \bmod 5$

$= 32 \bmod 5 = 2$

$\Rightarrow (32 + 2^2) = 36 \bmod 5 = 1$ ✗

$32 + 3^2 = (32 + 9) \bmod 5 = 1$ ✗

$32 + 4^2 = 48 \bmod 5 = 3$ ✓

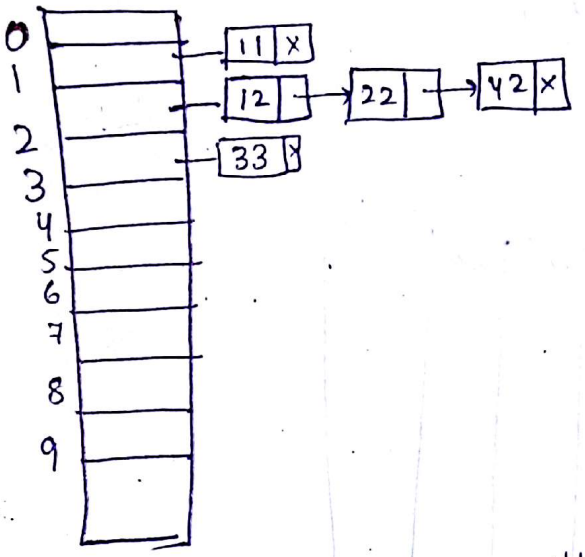
∴ 32 will be placed at 3.

Separate chaining:

All keys that map to the same hash value are kept in a list.

eg. 12, 11, 22, 33, 42

$H(K) = K \bmod 10$.



$12 \% 10 = 2$

$11 \% 10 = 1$

$22 \% 10 = 2$

$33 \% 10 = 3$

$42 \% 10 = 2$

① ordered chain will be benefial in case of searching.

③ Pseudorandom probe:

If a collision occurs then find new address using the formula -

$$\text{Add} = (a \times \text{coll. add} + c) \% \text{ size} + 1$$

$$= (23 * 1 + 200) \% 25 + 1$$

$$= 223 \% 25 + 1$$

$$= 24$$

④ Linked list probe

- Same as chaining method.

(5) Bucket hashing

- Convert hash table slot into bucket.
- m slots of hash tables are divided into buckets "B" with each bucket having m/B slots.

eg. we have 20 keys

B = 5 (create 5 buckets)

No. of slots = $\frac{20}{5} = 4$

20
22
10
30

K mod 5 =

0	20	10	30	
1				
2	22			
3				
4				

Bucket hash

20 mod 5 = 0
22 mod 5 = 2
10 mod 5 = 0
30 " " = 0

Note: In case of overflow, we may use linear hashing or extensible hashing.