classmate DK

Date
Page

UNIT-III

Searching And Sorting

- **Definitions -**

Searching -
Searching is the process of finding the location of target data among list of objects

Sorting -
It is a process of organizing data in certain order to help retrieve it more efficiently

★ Sequential / Linear Search

→ • Sequential search is a technique in which given list of elements is scanned from beginning. The key element is compared with every element of list.

- If the match is found, searching is stopped, otherwise it will be continued to end of list

- Although this is a simple method, there are some unnecessary comparisons involved

- Time complexity of this algorithm is O(n), it increase linearly with value of n

- Eg,

★ Binary Search –

→ • Binary search is a searching algorithm in which list of elements is divided into two sublists and key element is compared with middle element

• If match is found then location of middle element is returned, otherwise we search into either of halves depending upon result produced. This algorithm is considered as efficient searching algorithm.

• Algorithm –

```
1. if (low > high)
2. return;
3. mid = (low + high)/2
4. if (x == a[mid])
5. return (mid);
6. if (x < a[mid])
7. search for x in a[low] to a[mid-1]
8. else
9. search for x in a[mid+1] to a[high]
```

• Binary search has time complexity of $O(1)$ in best case and,
$O(\log N)$ in worst case

★ Fibonacci Search

→ • Algorithm for Fibonacci search:

Let the length of given array be n [0... n-1] and element to be searched is key

Then we use the following steps:

1) Find the smallest Fibonacci number greater than or equal to n. Let this number be f (m^{th} element)

2) Let the 2 fibonacci numbers preceding it be a ($m-1^{th}$) and b ($m-2^{th}$ element)

While array has elements to be checked:

Compare key with last element of range covered by b

ⓐ If key matches, return index value

ⓑ Else if, key is less than element, move third Fibonacci variable two Fibonacci down, indicating removal of approx. two-third of array

ⓒ Else key is greater than element, move third Fibonacci variable one Fibonacci down. Reset offset to index. This results approx. one-third removal of unsearched array

3) Since there might be single element remaining for comparison, check if a is '1'. If yes, compare key with that remaining element.

If match, return index value.

• The time complexity for Fibonacci search is O (logn)

SORTING

★ Insertion Sort -

→ • Although it is very natural to implement insertion using recursive (top down) algorithm, but it is efficient to implement it using iterative (bottom up) approach.

• Algorithm:

```
1. Insertion_sort(A[0...n-1])
2. for i ← 1 to n-1, do
3. {
      temp ← A[i]
       j ← i-1
      while (j>=0) AND (A[j] > temp), do
4.    {
      // comparing all previous elements of A[i]
      // with A[j]. If any greater element is found
      // then insert it at proper position
        A[j+1] ← A[j]
         j ← j-1
      }
5.   A[j+1] ← temp        // copy A[i] element at A[j+1]
   }
6. Stop
```

• Best case time complexity is $O(n)$
Worst case time complexity is $O(n^2)$

Q.] Sort given list using insertion sort :
7, 4, 10, 6, 3, 12, 1, 8, 2, 15, 9, 5

→ Consider list of elements as

0	1	2	3	4	5	6	7	8	9	10	11
7	4	10	6	3	12	1	8	2	15	9	5

Process starts with first element

- Compare 7 and 4 and insert it at proper position

∴ | 4 | 7 | 10 6 3 12 1 8 2 15 9 5

- Compare 10 with 4 & 7 and insert at proper position

∴ | 4 | 7 | 10 | 6 3 12 1 8 2 15 9 5

- Compare 6 with 4, 7, 10 and insert it

∴ | 4 | 6 | 7 | 10 | 3 12 1 8 2 15 9 5

- Compare 3 with 4, 6, 7, 10 and insert it

| 3 | 4 | 6 | 7 | 10 | 12 1 8 2 15 9 5

- Similarly,

| 3 | 4 | 6 | 7 | 10 | 12 | 1 8 2 15 9 5

| 1 | 3 | 4 | 6 | 7 | 10 | 12 | 8 2 15 9 5

1	3	4	6	7	8	10	12

2 15 9 5

1	2	3	4	6	7	8	10	12

15 9 5

1	2	3	4	6	7	8	10	12	15

9 5

1	2	3	4	6	7	8	9	10	12	15

5

1	2	3	4	5	6	7	8	9	10	12	15

★ Stability of Sorting -

→ • Sorting stability means comparing the records of same value and expecting them in same order even after sorting them.

• For example,
(Pune, BalGandharva)
(Pune, Shaniwarwada)
(Nashik, Panchvati)

Now, after sorting according to first alphabet of city.
(Nashik, Panchvati)
(Pune, Balgandharva)
(Pune, Shaniwarwada)

• In above list same record Pune is twice, but we have preserved original sequence as it is after

comparing them. Thus stability is achieved in sorting records.

Q.] Sort the following data using merge sort and selection sort :

142 317 45 222 187

→ Let,

142	317	45	222	187

↑ min

be given elements

① Selection Sort –

- Pass 1 : Consider A[0] as first element. Assume this as min.
  Scan the rest array for smallest element, if found, swap it with A[0]

∴

0	1	2	3	4
45	317	142	222	187

↑ min (at 317)

Pass 2 :

0	1	2	3	4
45	142	317	222	187

↑ min (at 317)

Pass 3 :

45	142	187	222	317

This is sorted list.

Q] Sort following numbers :
17, 24, 49, 7, 8, 67, 23
using Bubble sort

→ • Let 17, 24, 49, 7, 8, 67, 23 be given list of elements.
We will compare adjacent elements. If A[i] > A[j] then swap the elements

• Pass 1 :

- 17 24 49 7 8 67 23

- 17 24 49 7 8 67 23

- 17 24 49 7 8 67 23

∴ 17 24 7 49 8 67 23

∴ 17 24 7 8 49 67 23

∴ 17 24 7 8 49 67 23

∴ 17 24 7 8 49 23 67

• Pass 2 :

17 24 7 8 49 23 67

17 24 7 8 49 23 67

17 7 24 8 49 23 67

17 7 8 24 49 23 67

17 7 8 24 49 23 67

17 7 8 24 23 49 67

- Pass 3 -

17 7 8 24 23 49 67

7 17 8 24 23 49 67

7 8 17 24 23 49 67

7 8 17 24 23 49 67

∴ 7 8 17 23 24 49 67

∴ This is sorted list.

★ Radix Sort –

→ • **Algorithm –**

1) Read total number of elements in array
2) Store unsorted elements in array
3) Now, sort the elements digit by digit
4) Sort elements according to last digit, then second last digit & so on
5) Thus, elements are sorted for up to most significant bit
6) Store sorted element in array & print
7) Stop

• **Pseudocode –**

```
radix Sort Algo( arr as an array)
    Find largest element in arr
    maximum = largest element
    Find no. of digits in maximum
    k = no. of digits in maximum

    Create buckets of size 0-9 k times
    for j ← 0 to k
    Acquire jth place of each element in arr.
    Here j=0 represents least bit
    Use stable sorting algo like counting sort
      to sort the elements according to digits
    arr = sorted elements
```

* Merge Sort -

→ • If the list is of length 0 or 1, then it is already sorted

• Otherwise, the merge sort algorithm divides the unsorted list into two sub-lists of half the size

• Then, it sorts each sub-list recursively by reapplying the merge sort and then merges the two sub-lists back into one sorted list

• Its time complexity in best case is $O(n \log_2 n)$

• Its space complexity is $O(n)$

* Bucket Sort -

→ Pseudo-code -

•
```
function bucketSort (array, n) is
  buckets ← new array of n empty lists
    M ← maximum key value in array
  for i = 1 to length (array), do
      insert array[i] into buckets [floor(array[i] /
                                            M*n)]

  for i = 1 to n, do
     sort ( buckets[i])
```

classmate DK

Date
Page

UNIT - IV

Linked Lists

* Static and Dynamic Memory Allocation

- Static memory management means allocating / deallocating of memory at compilation time, while dynamic refers to allocation / deallocation of memory while program is running

• Static Memory	Dynamic Memory
1) Memory allocation is done at compile time	1) Memory allocation is done at run time
2) Prior to allocation of memory, some fixed amount of it must be decided	2) No need to know amount of memory prior to allocation
3) Wastage or shortage of memory	3) Memory can be allocated as per requirement
4) Eg. Array	4) Eg. Linked List

* Differentiate between Singly & Doubly Linked List

→

Sr. No	Singly Linked List	Doubly Linked List
1.	Singly linked list is a collection of nodes and each node is having one data field and next link field For example, | Data | Next Link |	Doubly linked list is a collection of nodes and each node is having one previous link field and one next link field For example, | Previous | Data | Next |
2.	The elements can be accessed using next link	The elements can be accessed using both, previous & next link
3.	No extra field is required Hence, node takes less memory in SLL	One field is required to store previous link; hence node takes more memory
4.	Less efficient access to elements	More efficient access to elements

* 'C' function to delete a number from singly linked list -

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

void deleteNode (struct Node** head_ref, int key)
{
    struct Node *temp = *head_ref, *prev;

    if (temp != NULL && temp -> data == key) {
        *head_ref = temp -> next;
        free (temp);
        return;
    }

    while (temp != NULL && temp -> data != key) {
        prev = temp;
        temp = temp -> next;
    }

    if (temp == NULL)
        return;

    prev -> next = temp -> next;
    free (temp);
}
```

★ Comparison of Linked List & Array

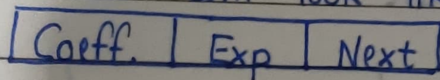
Sr. No	Linked List	Array
1)	The linked list is a collection of nodes and each node is having one data field and next link field Eg. [Data | Next Link]	The array is a collection of similar types of data. Data is always stored at some index of array Eg. 10 (a[0]), 20 (a[1]), 30 (a[2]), 40 (a[3])
2)	Any element can be accessed by sequential access only	Any element can be accessed randomly i.e with help of index
3)	Physically, the data can be deleted	Only logical deletion of data is possible
4)	Insertion and deletion of data is easy	Insertion and deletion of data is difficult

★ Polynomial Representation using Linked List -

- A polynomial has main fields as coefficient, exponent in linked list, it will have one more field called 'link' field to point to next term in polynomial.

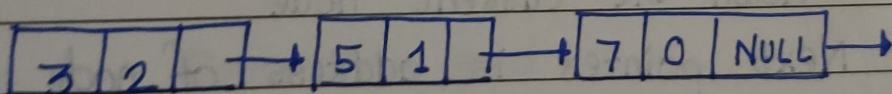
- If there are 'n' terms, then 'n' nodes need to be created

- Typical node will look like this,



- In each node, exponent field will store exponent of that term, coefficient will store coefficient of that term, and link field will point to next term in polynomi

- Eg, To represent $3x^2 + 5x + 7$, link list will be



- Node structure for representing polynomial can be defined as:

```
typedef struct Pnode
{
    float coef;
    int exp;
    struct node *next;
} p;
```

★ Generalized Linked List -

→ • A generalized linked list A, is defined as, finite sequence of $n >= 0$ elements, $a_1, a_2, a_3, \ldots a_n$, such that a_i are either atoms or list

Thus $A = (a_1, a_2, a_3, \ldots a_n)$

where n = total no. of nodes in list

• To represent such a list, we will have assumptions about structure

Flag	Data	Down pointer	Next pointer

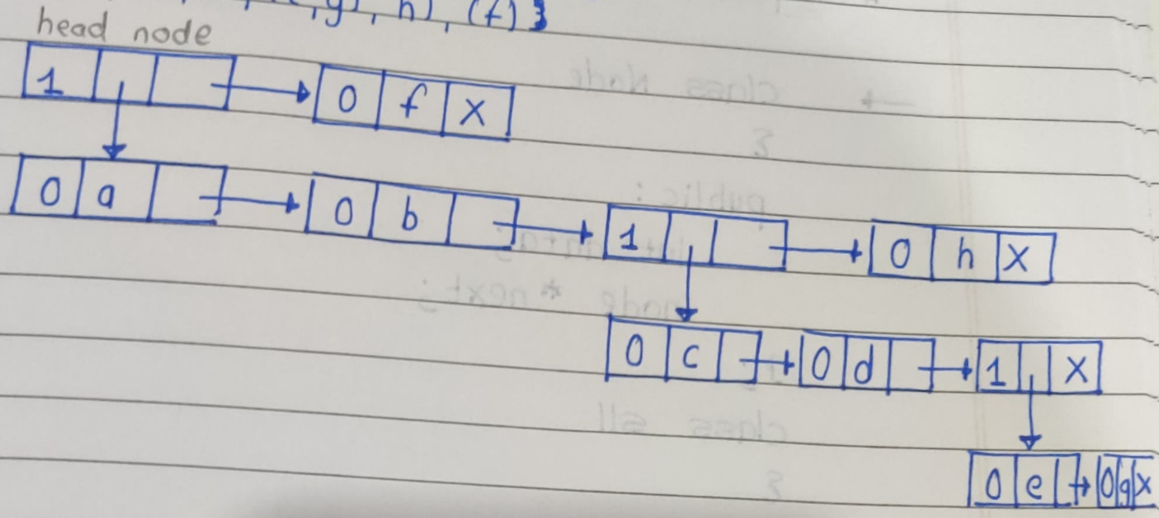
Flag = 1 means Down pointer exists
= 0 means Next pointer exists

Data means the atom

Down pointer is address of node which is down the current node

Next pointer is address of node which is attached as next node

Q.7 Represent following set using GLL
(a, b, (c, d), (e, g), h), (f)}



★ Algorithm to Delete Intermediate Node From DLL

→ Step 1 : If Head = Null,
give message "Underflow" and go to step 6

Step 2 : Set Ptr = Head
Step 3 : Set Head = Head -> Next
Step 4 : Set Head -> Prev = NULL
Step 5 : Free Ptr
Step 6 : Exit.

* Representation of a Linked List -

→

```
class Node
{
public :
    int data;
    node *next;
};
class sll
{
private :
    node *head;        // Data members of list
public :
    void create();
    void print();
    :
};
```

* Operations on a Linked List -

1) Creation
2) Insertion
3) Deletion
4) Reverse
5) Search
6) Display

Stack

- **Definition** –

A stack is an ordered list in which all insertions and deletions are made at one end, called the top.

Eg, if we have stack of elements, 10, 20, 30, 40, 50 then 10 will be bottommost element and 50 will be topmost element

50 ← Top
40
30
20
10

★ **Representation of Stack**

→

```
# define size 50
struct stack {
        int s[size];
        int top;
              } st;
```

★ Push and Pop Operations :-

→ 1) Push -

- Push is a function which inserts new element at the top of stack
- The function is as follows -

```
void push(int item)
{
    st.top++;              /* top pointer set to next loc. */
    st.s[st.top] = item;   /* placing element at that
                              loc. */
}
```

2) Pop -

- The pop operation deletes the element at top of stack and returns the same
- This is only done, if stack is not empty
- pop function is as follows,

```
if (top == -1)
    cout << "Stack underflow \n";
else
    return (Stack[top--]);
```

Date
Page

★ **Applications of Stack** -

- Stack is used for converting one form of expression to another
- Stack is used for parsing the well formed parenthesis
- Decimal to binary conversion can be done by using stack
- Stack is used for reversing the string

★ **Types of Expressions** -

1) **Infix Expression:**
In this type of expression, arrangement of operands & operator is:
operand1 operator operand2
e.g (a + b) , (a+b) * (c-d)

2) **Postfix Expression:**
In this type of expression, arrangement of operands & operator is:
operand1 operand2 operator
e.g ab+ , ab + cd - *

3) Prefix Expression -
In this, arrangement of operators & operand is :

operator operand1 operand2

e.g + ab , * + ab - cd

★ Algorithm to convert from Infix to Postfix -

→ Read an expression from left to right each character one by one

1) If an operand is encountered, then store it in postfix array
2) If '(' is read, then simply push it into stack because '(' has highest priority
3) If ')' is read, then pop all the operators, until '(' is read. Discard '('. Store the popped operators in postfix array
4) If operator is read then,
   ⓐ If instack operator has greater precedence (or equal to) over incoming operator, then pop the operator and add it to postfix array. Repeat until we get instack operator of higher priority than current incoming operator
   ⓑ Else push the operator
5) If stack is not empty, then pop all the operators and store in postfix array
6) Finally, print the array of postfix expn.

- **Definition** –

Recursion –
Recursion is a programming technique in which function calls itself repeatedly for some input

★ Backtracking Algorithmic Strategy –

- Backtracking is a method in which
  1) The desired solution is expressible as 'n' tuple $(x_1, x_2, \ldots x_n)$ where x_i is chosen from finite set S_i.
  2) The solution maximizes or minimizes or satisfies a criterion function $C.(x_1, x_2, \ldots x_n)$

- The basic idea of backtracking is to build up a vector, one component at a time and to test whether vector being formed has any chance of success.

- Backtracking algo. determines the solution by systematically searching the solution space for given problem

Q. Convert given expression to postfix expression using stack

(a$b)* c-d/d $ = Exponent operator

Input Symbol	Action	Stack	Postfix expression
(	Push (	(	
a	Print a	(	a
$	Push $	($	a
b	Print b	($	ab
)	Pop $, print		ab$
	Pop (		
*	Push *	*	ab$
c	Print c	*	ab$c
-	Pop *, print	-	ab$c*
	Push -		
d	Print d	-	ab$c*d
/	Push /	-/	ab$c*d
d	Print d	-/	ab$c*dd
end	Pop all	empty	ab$c*dd/-

*** IMP TABLE ***

Operator	Instack priority	Incoming priority
+ or −	2	1
* or /	4	3
^ or any exp. operator	5	6

If Instack priority > Incoming priority then POP

If Instack priority < Incoming priority then PUSH

Q. a*b/c*d-e/f , convert to postfix

Input	Action	Stack	Postfix Expn
a	Print a		a
*	Push *	*	a
b	Print b	*	ab
/	Pop *, Print	/	ab*
	Push /		
c	Print c	/	ab*c
*	Pop /, print	*	ab*c/
	Push *		
d	print d	*	ab*c/d
−	Pop *, print	−	ab*c/d*
	Push −		
e	Print e	−/	ab*c/d*e
			ab*c/d*e

(ii) (a+b)/(c+d), convert to postfix

Input	Action	Stack	Postfix Expn.
(	Push (	(	
a	Print a	(	a
+	Push +	(+	a
b	Print b	(+	ab
)	pop +, print pop (		ab+
/	Push /	/	ab+
(	Push (	(	ab+/
	pop /, print		
c	Print c	(	ab+/c
+	push +	(+	ab+/c
d	print d	(+	ab+/cd
)	pop +, print pop (	empty	ab+/cd+

UNIT VI

Queue

Definition -

Queue -

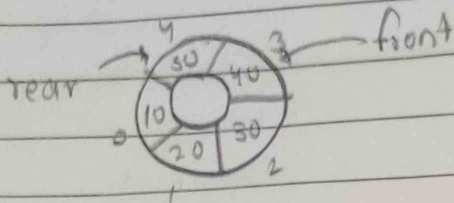
It is an ordered collection of elements that has two ends named front and rear.
From front end, we can delete elements and from rear end, we can insert elements.

Applications -

- Queue can be applied on CD players as buffers
- Applied on operating system to handle interruption
- Applied on Whatsapp servers, (when we send msg to friend & they don't have internet then messages are queued.)

★ Circular Queue -

- Circular queue is a queue in which front and rear are adjacent to each other
- It can be represented as follows -
  for rear and front pointers, following formula is used,
  rear = (rear + 1) % SIZE
  front = (front + 1) % SIZE
  where SIZE represents size of queue



- We can implement circular queue using linked list
- It is similar to circular linked list except there is two pointer front and rear in circular queue

- for enqueue (data),
  1) Create a struct node type node
  2) Insert given data in new node data section and NULL in address section
  3) If queue is empty then initialize front and rear from new node
  4) If queue is not empty then initialize rear next and rear from new node
  5) New node next initialize from front

- Program for circular queue using Linked list:

```
# include <iostream>
using namespace std;

struct Node {
        int data;
        struct Node* link;
};

Struct Queue {
        struct Node * front, *rear;
};
```

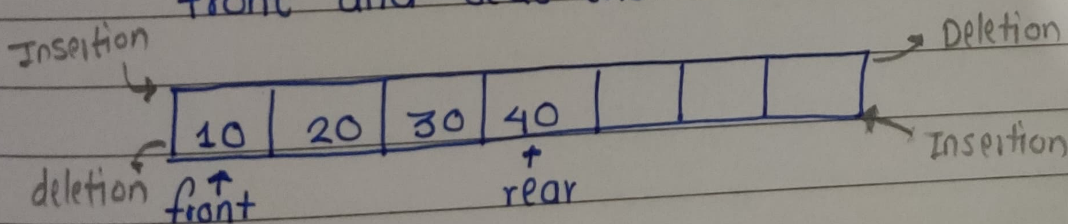
```
void enqueue (Queue * q, int value)
{
    struct Node* temp = new Node;
    temp -> data = value;
    if (q -> front == NULL)
          q -> front = temp;
    else
         q -> rear -> link = temp;

    q -> rear = temp;
    q -> rear -> link = q -> front;
}
```

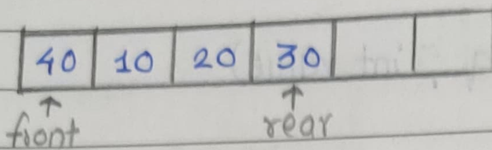
* **Deque** –

→ • Deque is a data structure in which we can insert elements both by front and rear end.

• Similarly, we can delete elements from both front and rear end.



• Eg, we have a deque of elements 10, 20, 30. If we wish to insert any element from front end then first we have to shift all elements to right.

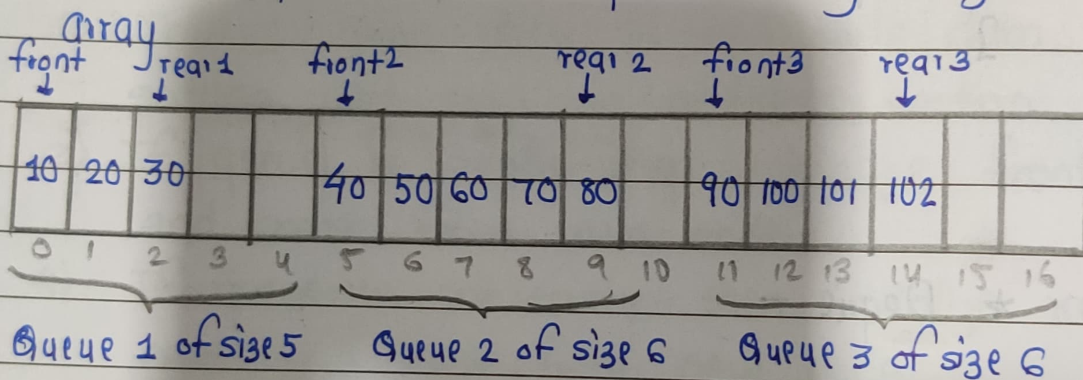


Inserted 40 in queue by front end

★ Multi - Queues -

→ • Multiple queues can be used to store variety of data

• We can implement multi-queues using single dimensional array



• We can perform insertion and deletion of any element for any queue.

classmate

Date
Page

★ Priority Queue -

→ • Priority queue is data structure having collection of elements in specific ordering

• There are 2 types of priority queue -

1) Ascending priority queue -

It is a collection of items in which items can be inserted arbitarily but only smallest element can be removed

2) Descending priority queue -

It is collection of items in which items can be inserted arbitarily but only largest element can be removed.

• Implementation of priority queue can be done using Arrays or Linked List.

• Data structure heap is used to implement priority queue effectively.