Experiment 1

1.)
```
#include <stdio.h>
int main(){
int numbers[10];
int sum=0;
float average;
printf("Enter 10 numbers :\n");
for(int i=0; i<10; i++){
printf("Number %d :", i+1);
scanf("%d", &numbers[i]);
    sum += numbers[i];
}
average = sum/10.0;
printf("Sum = %d \n", sum);
printf("Average = %.2f\n", average);
return 0;
}
```

Output:

enter num 1: 10
enter num 2: 22
enter num 3: 12
enter num 4: 44
enter num 5: 42
enter num 6: 16
enter num 7: 1
enter num 8: 29
enter num 9: 36
enter num 10 = 65
Sum: 277
average = 27.70.

Pattern

2)
```
#include <stdio.h>
int main(){
int i, j;
for (i=1; i<=4; i++){
 for (j=1; j<=4; j++){
   if (i%2==1)
     printf("*");
   else
     printf("#");
  }
  printf("\n");
  }
  return 0;
  }
```

Output

*

# #

* * *

# # # #

3) #

3)
```
# include <stdio.h>
int main() {
int arr[100], n, i, j, found = 0;
printf("enter number of elements:");
scanf("%d", &n);
printf("enter %d elements:", n);
for(i=0; i<n, i++) {
  scanf("%d", &arr[i]);
}
for(i=0; i<n; i++) {
  scanf("%d", &arr[i]);
    }
  for(i=0; i<n; i++) {
    for(arr[i] == arr[j]) {
      printf("First repeating
      element is : %d \n", arr[i]);
      found = 1;
      Break;
    }
  }
if (found)
  break;
}
if (!found)
printf("\n\n No repeating elements");
  return 0;
}
```

Output

12
13
12
11

First repeating number is : 16

4.) #include <stdio.h>
int main () {
int n, i, j
printf ("enter the number of elements
scanf ("%d", & n);
int arr [n];
printf ("enter %d element: \n", n);
for (i=0; i<n; i++) {
scanf ("%d", & arr[i]);
}
int smaller = arr [0];
int greatest = arr [0];

for (i=1; i<n; i++) {
if (arr [i] < smallest) {
smallest = arr [i];
}
if (arr [i] > greatest) {
greatest = arr [i];
}
}
printf ("Smallest element: %d \n",
smallest);
printf ("Greatest element: %d \n",
greatest);
return 0;
}

Output
enter number of elements: 4
enter 4 elements
3
2
2
1
Smallest element = 1
greatest element = 3.

5.) #include <stdio.h>
int main() {
int n, j;

printf("enter the number of elements in the array:");
scanf("%d", &n);
int arr[n];
printf("enter %d integers:\n", n);
for(i=0; i<n; i++){
scanf("%d", &arr[i]);
}
printf("squared odd numbers in the array:\n");
for(i=0; i<n; i++){
if(arr[i] % 2 != 0){
arr[i] = arr[i] * arr[i];
printf("%d", arr[i]);
}
}
Printf("\n");
return 0;

Output.

```
Enter the number of elements in
the array: 4
enter 4 integers:
23
55
32
22
Squared odd numbers in the
array: 529 3025
```

Extra Questions:

1.)

```
    *
# # #
$ $ $
? ? ? ?
```

```
#include <stdio.h>
int main(){
char arr[4]={'*','#','$','?'};
for(int i=0;i<4;i++){
for(int j=0;j<i+1;j++){
printf("%c",arr[i]);
}
printf("\n");
}
return 0;
}
```

Output:

```
  *
# #
$ $ $
? ? ? ?
```

2)

```
#include
int main(
int n;
printf(
scanf(
for(int
for(i
pri
y
prin
}
retu
}
out
enter
1
2 2
3 3
4
5
```

2)
```
#include <stdio.h>
int main(){
int n;
printf("enter number of rows:");
scanf("%d",&n);
for(int i=0;i<n;j++){
for(int j=0;j<i+1;j++){
printf("%d",i+1);
}
printf("\n");
}
return 0;
}
```

output.

enter number of rows: 6

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

3)
```
#include <stdio.h>
int main(){
int n;
printf("Enter number of rows:");
scanf("%d", &n);
for(int i=0; i<n; i++){
for(int j=0; j<i+1; j++){
printf("%d", j+1);
}
printf("\n");
}
return 0;
}
```

output

```
enter number of rows
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

4)
```
#include <stdio.h>
int main(){
int n;
printf("enter number of rows:");
scanf("%d", &n);
for(int i=0; i<n; i++){
for(int j=0; j<i; j++){
printf("*");
}
```

```
for (int j=0; j<n-i; j++){
printf("*");
}
printf("\n");
}
return 0;
}
```

output:

Experiment No: 2

(Q.) Search data using linear search

Consider following list to perform linear search.

56, 36, 89, 57, 1, 0, 67, 59

i) search item '1' from the above list & write whether item is found or not.

ii) Search item '55' from the above list & write whether item is found or not.

→
```
#include <stdio.h>
int main(){
  int arr[] = {56, 36, 89, 57, 1, 0, 67,
                                  59};
  int target1 = 1;
  int target2 = 55;
  int found1 = 0;
  int found2 = 0;
    for(int i=0; i<8; i++){
      if (arr[i] == target1){
        found1 = 1;
      }
      if (arr[i] == target2){
        found2 = 1;
      }
    }
```

```
if (found == 1) {
    printf("Target 1 found\n");
  }
else {
printf("Target 1 not found\n");
  }
if (found2 == 1) {
    printf("Target 2 found\n");
  }
  else {
printf("Target 2 not found\n");
  }
return 0;
}
```

Output:
Target 1 found
Target 2 not found.

Q2.) Search data using binary search (use input array

```
#include <stdio.h>
int binarysearch (int arr[ ], int size,
int target){
int left = 0
 int right = size-1;
 while (left <= right) {
 int mid = (left + right) / 2;
if (arr[mid] == target) {
   return mid;
 }
```

```
if (arr[mid] > target){
    right = mid-1;
  }
  else {
  left = mid+1;
  }
  else {
  left = mid+1;
  }
}
return -1;
}
int main(){
int arr[10];
for(int i=0; i<10; i++){
printf("enter the target:");
scanf("%d", &target:);
int index = binary search (arr,
10, target);
printf(index != -1 ?= "element
found at index %d": element
not found", index);
}
```

output:

enter a number : 12
enter a number : 14
enter a number : 16
enter a number : 18
" " 20
" " 22
" " 24
" " 26
" " 28
" " 30
" " 16

element found at index : 2

Q3) compare linear search & & Binary

Cas 9.	linear search	Binary search
(1)	Start at the beginning of the list, compare target	finds the middle elements and compare it to the target
(2)	works on unsorted elements as well as sorted arrays	work only on sorted data.
(3)	Is efficient for large data structur	efficient for large data struct
(4)	O(n)	O(log N)

Q4.) State limitations of linear search in terms of time complexity

1.) Inefficient for large datasets
2.) Is runtime grows linearally with the number of elements
3.) even if data is sorted linear search does not take an a
4.) input size increases, the performance degrades proportionally
5.) No optimization based on data distribution.
6.) Best complexity: O(1)
Average case: O(n/2) → O(n)
worst case: O(n)

25/1/26

Extra

Q1.) w AP to
array
ord

#inclu
int ar
int ar
for (
ar
}
for (i
pri
}
ret
}

Out
S

Extra questions

Q1) WAP to copy the elements of one array into another array in reverse order

```
#include <stdio.h>
int arr[5] = {1, 2, 3, 4, 5};
int arr2[5];
for (int i=0; i<5; i++){
 arr2[i] = arr1[4-i];
}
for (int i=0; i<5; i++){
 printf("%d", arr2[i]);
}
return 0;
}
```

Output

5 4 3 2 1

Q2) WAP in C to count total number of duplicate elements in an array.

```
#include <stdio.h>
int main(){
int arr[15] = {1,2,3,4,5,6,7,8,9,10,1,2,3,4,5};
int count = 0;
for(int i=0; i<15; i++){
for(int j=i+1; j<15; j++){
    if (arr[i] == arr[j]){
    count++;
    }
  }
}
printf("Total number of duplicate
      elements: %d \n", count);
    return 0;
}
```

Output

Total number of duplicate elements: 5

Q3 WAP in c to print all unique elements in an array

```
#include<stdio.h>
int main(){
int arr[15]={1,2,3,4,5,6,7,8,9,10,1,2,3,3,
                                    5};
int count=0;
for(int i=0; i<15; i++){
  int frequency=0;
for(int j=0;j<15;j++){
if(arr[i]==arr[j]){
        frequency++;
  }
}
printf("Total number of unique elements:
      %d\n",count);
return 0;
}
```

Output

Total number of unique elements:%d\n, count);

Output

Total number of unique elements: 6

→ WAP to seperate odd and even integers in seperate arrays.

```
#include <stdio.h>
int main (){
int arr[10] = {1,2,3,4,5,6,7,8,9,10};
int odd[10];
int even[10];
int oddcount=0;
int evencount=0;
for(int i=0; i<10; i++){
    if (arr[i]%2==0){
     even[evencount] = arr[i];
     evencount++;
   } else{
    odd[oddcount] = arr[i];
    oddcount++;
   }
}
printf("Odd number:");
for(int i=0; i<oddcount; i++){
printf("%d", odd[i]);
}
printf("\n Even numbers:");
for(int i=0; i<evencount; i++){
 printf("%d", even[i]);
}
return 0;
}
```

Output

Odd number: 1 3 5 7 9

even number: 2 4 6 8 10

→ second smallest number in an arrays.

```
#include <stdio.h>
int main(){
int arr[11]= {1,2,3,4,5,6,7,8,9,-10,10};
int smallest = arr[0];
int second smallest - arr[0];
for(int i=0; i<10; i++){
  if (arr[i]<smallest){
   second smallest= smallest;
   smallest = arr[i];
  }
}
printf("second smallest number:%d\n
 second smallest);
   return 0;
   }
```

Output

Second smallest number: 1

→ Count total number of words in a string

```
#include <stdio.h>
#include <string.h>
int main(){
 char str[100];
 printf("Enter a string:");
 fgets(str, 100, stdin);
 int count=0;
 for(int i=0; i<strlen(str); i++){
  if (str[i] == " "){
     count++;
  }
}
printf ("Total number of words:
                %d\n", (count+1));
}
```

Output

```
enter a string : I  D
Total number of strings : 2
                  words
```

Experiment 3

1) Sort elements in ascending order using bubble sort.

```
#include <stdio.h>
int main(){
  int n;
printf("Enter the number of
  elements:");
    scanf("%d", &n);
int arr[n];
printf("Enter %d elements: \n", n);
for(int i=0; i<n; i++){
  scanf("%d", &arr[i]);
  }
for(int i=0; i<n-1; i++){
  for(int j=0; j<n-i-1; j++){
  if (arr[j] > arr[j+1]){
    int temp = arr[j];
  arr[j] = arr[j+1];
  arr[j+1] = temp;
  }
  }
  }
printf("Sorted arrays in Ascending
order:");
for(int i=0; i<n; i++){
  printf("%d", arr[i]);
  }
return 0;
  }
```

Output

```
Enter elements: 5
enter 5 elements:
7
9
3
5
9
Sorted array is: 35799
```

2) Sort elements in descending order using selection sort

```
#include<stdio.h>
int main(){
 int n;
printf(" Enter the number of
elements: ");
 scanf("%d", &n);
int arr[n];
printf(" Enter %d elements: \n", n);
for(int i=0; i<n; i++){
  scanf("%d", &arr[i]);
 }
for(int i=0; i<n-1, i++){
  int max_idx=i;
  for(int j = i+1; j<n; j++){
if (arr[j] > arr[max_idx])
  max_idx=j;

t temp = arr[max_idx];
```

Experiment 4

```
arr[max_idx] = arr[i];
arr[i] = temp;
}
printf("Sorted array in descending
  order:");
for(int i=0; i<n; i++){
  printf("%d", arr[i]);
}
return 0;
}
```

Output.

```
enter elements: 6
2
3
5
6
8
9
Sorted array: 9 8 6 5 3 2
```

3.) number of comparisons required in bubble sort method of the following.
100, 200, 300, 400, 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

4.) 500, -20, 30, 14, 50

Pass 1. 500, -20, 30, 14 50

-20 500 30 14 50

-20 500 30 14 50

-20 500 30 14 50

-20 500 30 14 50

P1

-20
-20
-20
-20
-20
-20
-2
-

-20 500 30 14 50

~~-20 500 30 14 50~~

-20 30 500 14 50 P2

-20 14 500 30 50

[-20] | [14] 500 30 50

-20 14 500 30 50

-20 14 30 500 50 P3

[-20] | [14] | [30] 500 50

-20 14 30 500 50 P4

~~-20~~ 14 30 50 500

List sorted

Now
1:19

Experiment 4

Q. 1) Sort element in ascending order using insertion sort.

→
```
#include <stdio.h>
int main(){
  int n; i, j, key;
  printf(" Enter the number of
                elements: ");
  scanf("%d", &n);
  int arr[n];
  printf(" Enter %d integer: \n", n);
  for(i=0; i<n ; i++){
   scanf("%d", &arr[i]);}
  for (i=1; i<n; i++){
      key = arr[i];
      for(j=i-1; j>=0; j--){
      if (arr[j] > key){
         arr[j+1] = arr[j];}
      else { break;}
   }
   arr[j+1] = key;
printf(" Array after pass %d: \n", i);
    for(j=0; j<n; j++){
printf("%d", arr[j]);}
printf("\n");
   }
printf(" Array sorted in ascending
order using insertion sort: \n")
```

for (i = 0; i
return 0;
}

Output
Enter the nu
Enter 5 In
3 5 9
Array a
3 5 9
Array a
3 5 9 1
Array a
1 3 5
Array a
1 2 3
Array
using
1 2

```
for (i = 0; i<n; i++) { printf("%d", arr[i]); }
return 0;
}
```

Output

Enter the number of elements: 5
Enter 5 Integers:
3 5 9 1 2
Array after pass 1
3 5 9 1 2
Array after pass 2
3 5 9 1 2
Array after pass 3
1 3 5 9 2
Array after pass 4
1 2 3 5 9
Array sorted in ascending order using insertion sort
1 2 3 5 9

2.) Radix sort

```
#include <stdio.h>
int main() {
int n, int arr[100];
printf("Enter the number of
elements (max 100): ");
Scanf("%d", &n);
printf("Enter %d integers: \n", n);
for(int i=0; i<n ; i++)
 {
 scanf("%d", & arr[i]);}
int max = arr[0];
for(int i =1; i<n ;i++) {
 if (arr[i]>max) {
 max = arr[i];}}
for(int exp=1; max/exp>0; exp =10){
 int output [100]; int count [10] =
 {0};
for(int i= 0 ; i<n;i++) {
count[ (arr[i] / exp) %10] ++;}
for(int i =1; i<10; i++) {
count [i]+= count [i-1];}
for(int i=n-1; i>=0; i++ ){
 output [count [arr[i] / exp)%10]-1
 = arr[i];
count[(arr[i]/exp) %10]--;
}
```

```
for (int i=0; i<n; i++) { arr[i] =
 output[i]; }
printf ("After pass:");
for (int i =0; i<n; i++) {
printf ("%d", arr[i]); }
 printf ("\n");
}
printf ("Sorted array:");
for (int i =0; i <n; i ++){
printf ("%d", arr [i]); }
printf ( "\n");
  return 0;
  }
```

O/P

Enter the number of elements (max 100): 5

Enter 5 integers

247 891 3 444 67

After pass: 891 3 274 444 67

After pass: ~~3 444 67 274 444 67~~

3 444 67 247 ~~891~~ 891

444

After pass: 3 67 274 444 891

Sorted array: 3 67 274 444 891

3) 7, 3, 5, 1, 9, 8, 4, 6

7 3 5 1 9 8 4 6

3 7 5 1 9 8 4 6

3 7 5 1 9 8 4 6

3 5 7 1 9 8 4 6

Output after second iteration for insertion Sort.

4) array = 100, 225, 390, 4130, 956, 99, 5431

	0	1	2	3	4	5	6	7	8	9
100	100									
225						225				
390	390									
4130	4130									
956								956		
99										99
5431		5431								

Arr = 100, 390, 4130, 5431, 225, 956, 99

0	1
100	100
390	
4130	
5431	
225	
956	
99	

Arr = 10

0
100
225
4130
5431
956
390
99

99

Arr

Arr

99
100
4130
5431

	0	1	2	3	4	5	6	7	8	9
100	100									
390										390
4130				4130						
5431				5431						
225			225							
956					9	956				
99										99

Ans = 100, 225, 4130, 5431, 956, 390, 99

	0	1	2	3	4	5	6	7	8	9
100		100								
225			225							
4130		4130								
5431						5431				
956										956
390				390						
99	99									

Ans = 99, 100, 4130, 225, 390, 5431, 956

Ans = ~~99, 100, 4130, 5431, 956~~

~~99~~
~~100~~
~~4130~~
~~5431~~
~~9~~

	0	1	2	3	4	5	6	7	8
99	99								
100	100	100							
4130					4130				
225	225		225						
390	390			390					
5431						5431			
956	956								

Arr = 99, 100, 225, 390, 956, 4130, 5431

↓

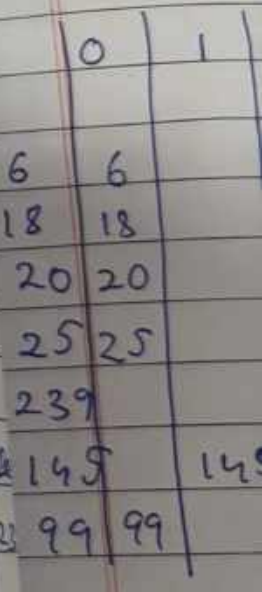
Sorted array

b.) Arr = 25, 6, 99, 145, 239, 20, 18

	0	1	2	3	4	5	6	7	8
						25			
25									
6							6		
99									
145					145				
239									
20	20								
18								18	

Arr = 20, 25, 145, 6, 18, 99, 239

Arr = 6, 1



Arr

	0	1	2	3	4	5	6	7	8	9
			20							
20			25							
25					145					
145										
6	6									
18		18								
99										99
239										

Arr = 6, 18, 20, 25, 239, 145, 99

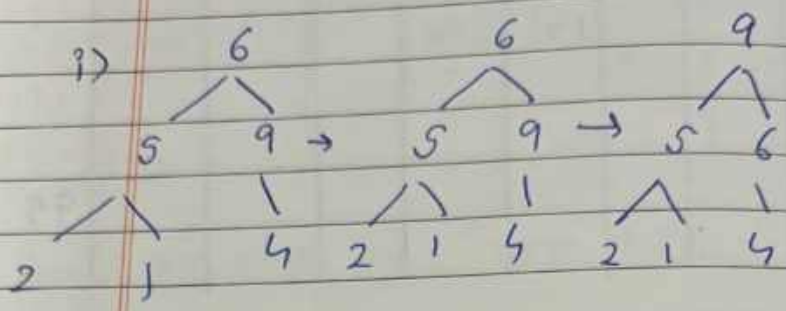
	0	1	2	3	4	5	6	7	8	9
6	6									
18	18									
20	20									
25	25									
239			239							
145		145								
99	99									

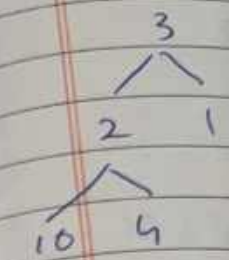
Arr = 6, 18, 20, 25, 99, 145, 239 - Sorted

AS.) Heap sort

1.) 6, 5, 9, 2, 1, 4 ⇒ n = 6

2.) 3, 2,

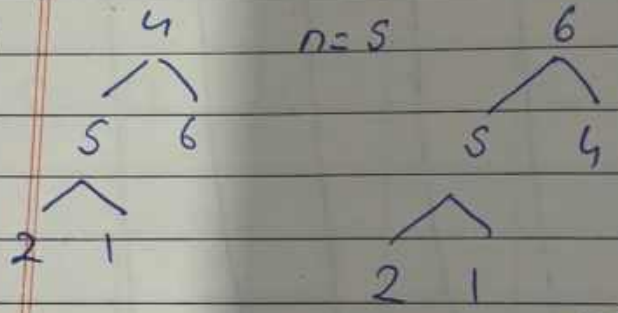




Arr = 9, 5, 6, 2, 1, 4

= 4, 5, 6, 2, 1, [9]

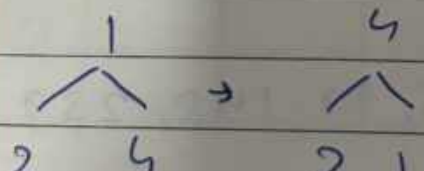
ii)



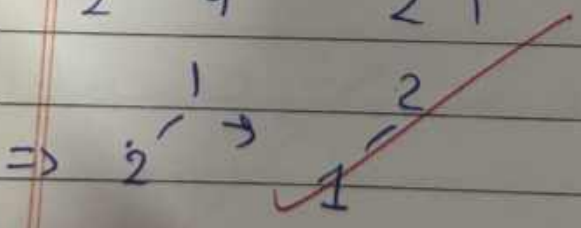
arr = 6, 5, 4, 2, 1, [9]

1, 5, 4, 2, [6] [9]

iii)

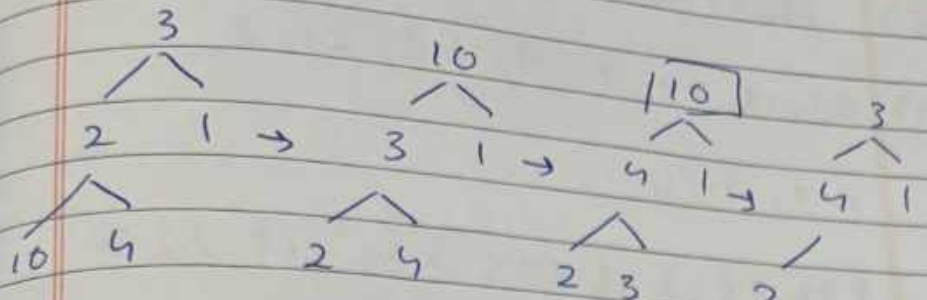


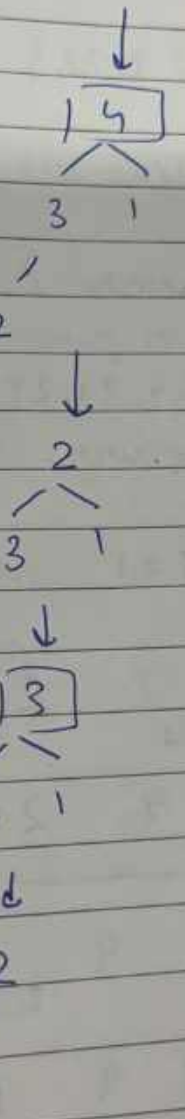
arr: 2, 1, [4] [3] [6] [9]



⇒ 1, 2, 4, 5, 6, 9 → Sorted array

2) 3, 2, 1, 10, 4





array: 1, 2, 3, 4, 10

6.) Shell sort

1) 22, 18, 29, 7, 9, 55, 21, 8

n/2, n/4 1,

{22, 9} {18, 55} {29, 21} {7, 8}

{9, 22} {18, 55} {21, 29} {7, 8}

n/4 = 2

Sublists: {9, 21, 22, 29} {18, 7, 55, 8}
compare and sort sublist:
{9, 21, 22, 29} {7, 8, 18, 55}
arr: 9, 7, 21, 8, 22, 18, 29, 55

n/8 = 1

9 7 21 8 22 18 29 55

7 9 21 8 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 18 21 22 29 55

array sorted

2) 3, 10, 15, 12, 1, 5, 2, 6.

n = 8

n/2 = 4

{3, 1} {10, 5} {15, 2} {12, 6}

1, 3, 5, 10, 2, 15, 6, 12

n/4 = 2

{1, 2, 3, 15} {5, 6, 10, 12}

compare & sort

arr = 1, 5, 2, 6, 3, 10, 15, 12

1 5 2 6 3 10 15 12

1 5 2 6 3 10 15 12

1 2 3 5 6 10 15 12

1 2 3 5 6 10 12 15

array sorted.

Null
11/9

Extra

→ Algorithm for all ~~searching~~ searching techniques

① Bubble sort.

1) → Compare current number (ci) with the next number element with (ci+1) and swap them if <i> <i+1
2) exclude the last sorted element (i.e the largest element) and repeat step such that next largest (i.e second or third) element is placed before the last element
3) Repeat step 2 until sorted list is produced (at maximum no. of repetition of step 2 will occur).

② Selection sort

1) Start from the first element in the list and compare will all other elements in the list such that if it is smaller than other element swap in comparison if necessary to satisfy the condition.
2) exclude the smallest element and repeat step such that the next smallest element it found unit only one element is unsorted (the largest)

3) Repeat step 2 forward.

③ Insertion s

1) It is the sorted. re
2) Pick next
3) compare sublist.
4) shift al list that to be so
5) Insert
6) Repeat

④ Bucket

1) Divide bucket.
2) Consid bucket
3) load thei of l
4) Carr the e the
5) Rep sig
6) Rep num si

3) Repeat step 2 for each position (element) forward.

(3) Insertion sort

1) If it is the first element, it is already sorted. return 1.
2) Pick next element.
3) compare with all elements in sorted sublist.
4) shift all elements in the sorted sub-list that are greater than the value to be sorted by one place.
5) Insert the value.
6) Repeat until sorted.

(4) Bucket sort/ Radix Sort

1) Divide (n) number each representing a bucket.
2) Consider each significant bit or each bucket in the list to be sorted.
3) load significant digit number into their respective buckets based on value of least significant digit.
4) Carry all the numbers from queue to queue in the order they are loaded to their respective queue.
5) Repeat from 3. Based on next least significant.
6) Repeat from step 2 until all the numbers are sorted (based on most significant digit.

7.) Print sorted list
8) Stop.

(5) Shell sort
1. start
2. initialize value of gap for size (eg: n/2)
3) Divide list into sublist with each having equal intervals (n)
4) sort the sub-lists (using insertion sort.)
5) Modify gap size (according to optimal sequence shell / knuth / Pratchew).
6.) Repeat from step 2 until list is sorted.
7.) Print sorted list.
9.) Stop.

(6) Heap sort
1) Construct a complete binary tree from array of size n.
2) Find last non-leaf node index (in level) = (n/2) - 1 and heapify
3) ~~Repeat~~ Repeat if child is larger than parent node child.
4.) Reconstruct array using max heap and swap largest element in last position with element that inserted.

5.) Repeat step
is left uns
6) Print sort

(7) Merge sort
1.) start
2) Divide ar
that each
3.) Divide e
until 1 el
4) start m
sub-arr
of divisio
5) Repeat st
array (
6) Print so
7) stop

Sorting	Bes
Bubble	O(
Selection	O(
Insertion	O
Bucket	O
Heap	O
Shell	O
merge	O

5.) Repeat step 2 until only one element is left unsorted (smallest)
6) Print sorted list.

(7) Merge Sort
1.) start
2) Divide array into two halves such that each half is a sub array
3.) Divide each sub-array for the until 1 one element is obtained.
4) start merging each seperated sub-array in their reversed order of division
5) Repeat step 3 till original length array (sorted) is obtained
6) Print sorted array
7) stop

Sorting	Best	Avg	Worst	Complexity Space	parameter for size
Bubble	O(n)	O(n^2)	O(n^2)	O(1)	small
Selection	O(n^2)	O(n^2)	O(n^2)	O(1)	small
Insertion	O(n)	O(n^2)	O(n^2)	O(1)	small
Bucket	O(n+k)	O(n+k)	O(n^2)	O(n)	small
Heap	O(n logn)	O(n logn)	O(n logn)	O(1)	nearly small
Shell	O(n logn)	O(n^(3/2))	O(n^2)	O(1)	large
Merge	O(n logn)	O(n logn)	O(n logn)	O(n)	medium

Experiment 5

1.@.) Write a menu driven C program that implements singly linked list for the following operation: Creat, Insert node at middle, Insert node at end.

2.@.) Write a menu driven C program single linked list for the following operation: Delete first node, Delete node at middle, delete node at end.

3 @) Write a menu driven C program that implements singly linked list for the following operations: Creat, Display, Concadenate, Count no. of nodes, Reverse, search a node.

code

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node* next;
};
struct node* head = NULL;
void creat_list();
void display_list();
void add_at_beginning();
void add_at_end();
void add_at_position();
void search_node();
void search_node();
void reverse_list();

int main(){
  int choice
while(1){
  printf("\n Menu:\n");
  printf("1. Creat list \n");
  printf("3. Add at beginning\n");
  printf("4. Add at end \n");
  printf("5. Add at position \n");
  printf("6. Search node \n");
  printf("7. Delete node\n");
  printf("8. Reverse list\n");
  printf("9. Exit\n");
  printf("Enter choice:");
```

scanf("%d

Switch(choic
Case 1: crea
Case 2: dis
case 3: add
case 4: add
case 5: ad
case 6: re
case 7: de
Case 8: r
case 9 :
default: p

}
}
return
}
// creat
void c
int
Struc
print
scanf
head=
for(
ten
(
prin
sca

```
scanf("%d", &choice);
switch(choice) {
case 1: create_list(); break;
case 2: display_list(); break;
case 3: add_at_beginning(); break;
case 4: add_at_end(); break;
case 5: add_at_position(); break;
case 6: reverse_list(); break;
case 7: delete_node(); break;
case 8: reverse_list(); break
case 9 : exit(0);
default: printf("Invalid choice, try
                again.\n");
  }
}
return 0;
}
// create new list with n nodes
void create_list(){
  int n, i;
struct node *temp, *tail;
printf("How many nodes?");
scanf("%d", &n);
head = Null;
for(i=0; i<n; i++){
  temp = (struct node *) malloc(sizeof
  (struct node));
printf("Enter data for node %d:", i+1);
scanf("%d", &temp->data);
```

```
    temp->next=NULL;
    if(head == NULL){
      head=temp;
      tail=temp;
    }
  }
  printf("List created with %d
    nodes.\n", n);
}
void display_list(){
  struct node*temp= Head;
if (temp== NULL) {
  printf("List is empty.\n");
    return;
  }
printf("List elements:");
while(temp != NULL) {
  ~~printf("%d ",temp->data);~~
printf("%d", temp->data);
  temp=temp->next;
}
printf("\n");
}
void add_at_beginning(){
  Struct node* temp= ~~Head~~;
(struct node*) malloc(sizeof(struct
  node));
Printf("Enter data to add at
          beginning:");
Scanf("%d", & temp->data);
```

```
temp -> next = head;
head = temp;
printf ("Node added at beginning\n");
}
void add_at_end(){
    struct node *temp = (struct node*) malloc
        (sizeof(struct node));
struct node * current = head;
    printf("Enter data to add at
        end:");
    scanf("%d", &temp->data);
    temp->next = NULL;
if (head == NULL){
    head = temp;
}
else {
while ( current->next != NULL)
    current = current->next;
current -> next = temp;
}
printf ("Node added at end.\n");
}
// Add a node at a specific position
void add_at_position(){
    int pos, i;
    struct node *temp, *current = head;
printf("Enter position to add node
        (starting from 1):");
scanf("%d", &pos);
if (pos<=0){
```

```
printf("Invalid position!\n");
    return;
  }
temp = (struct node*)malloc(sizeof
        (struct node));
printf("Enter data to add:");
scanf("%d", &temp->data);
if (pos == 1) {
    temp->next = head;
    head = temp;
    printf("Node added at
    position 1.\n");
    return;
  }
for(i=1; i<pos-1 && current!=
        NULL; i++)
    current = current->next;
if (current == NULL) {
    printf("Position out of
        range.\n");
    free(temp);
    return;
  }
temp->next = current->next;
current->next = temp;
printf("Node added at position
    %d.\n", pos);
}
```

```
void search_node(){
    int value, pos=1;
    struct node* current = head;
if (head == NULL){
        printf("List is empty.\n");
        return;
    }
printf("Enter value to search:");
    scanf("%d", &value);
while (current != NULL){
    if (current->data == value){
    printf("Value %d found at
     position %d.\n", value, pos);
    return;
    }
current = current->next;
    pos++;
    }
~~current.~~
printf("Value %d not found
          in the list.\n", value);
    }
void delete_node(){
int pos, i;
    struct node * current = head, *temp;
if (head == NULL){
        printf("list is empty\n");
        return;
}
```

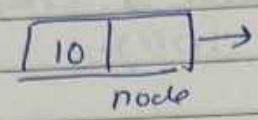
```
printf("Enter position to delete
node (starting from 1): ");
scanf("%d", &pos);

if (pos<=0){
printf("Invalid position\n");
return;
}
if (pos==1){
    temp= head;
    head = head->next;
    free(temp);
    printf("Node at position 1
        deleted.\n");
    return;
}
for(i=1; i<pos-1 && current!=NULL
        ; i++)
    current = current->next;

if (current==NULL || current->next
                    ==NULL){
printf("Position out of range.
                    \n");
return;
}
temp = current->next;
current->next = temp->next;
free(temp);
printf("Node at position %d
    deleted.\n", pos);
}
```

```
void reverse(
    struct node
    =head, *next
if (head==NU
    printf
    rever
    retur
}
while(curr
    next =
    curre
    pr
    cu
}
head=r
printf
}
```

```
void reverse_list() {
    struct node *prev = NULL, *current
    = head, *next = NULL;
if ( head == NULL){
    printf ("List is empty, cannot
        reverse.\n");
    return;
    }
while ( current != NULL){
    next = current -> next;
    current -> next = prev;
        prev = current;
        current = next;
    }
head = prev;
printf ("List reversed
            successfully\n");
}
```

4.) Basic terminology of Linked list

a.) NODE: Basic building block of list. It contains data (value) next (pointer to next node)

eg: 

b.) Head: the head is a reference or pointer to first node of list. NULL or nun.

c.) Tail
Last node where next == NULL

d.) Next:
each node has a next reference to following node.

e.) NULL pointer:
Special value that indicates that pointer does not have object memomory

) empty linked list:
singly linked list with no nodes in it.
head → NULL

5.) Arrays
✓ Resizing no possible
. memor all is fixed si
✓ Stored. i contigua
. memory is effici
✓ Simple to impler
. O(1) a tim

5.) Array

- Resizing not possible
- memory allocation is fixed size
- stored in contiguous
- memory usage is efficient
- simple to implement
- O(1) access time

linked list.

- can grow or shrink.
- size of array runs anemory
- Stored in non-contiguous memory
- uses extra memory for pointers
- complex implementation
- O(n).

10/11

experiment 6

extra * questions :-

1.) circular linked list -:
A circular linked list where the last node points back to the first node, forming a continuous loop, unlike a standard singly linked list. standard singly linked list, there is no NULL pointers at the end to indicate termination

Key characteristics -:
→ Circular structure
→ no termination.

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

void create();
void display();
void insert_begin();
void insert_end();
void insert_position();

struct node {
  int info;
  struct node* next;
  struct node* prev;
};

struct node* start = NULL;

int main() {
  int choice;
  while (1) {
    printf("\n MENU ");
    printf("\n 1. CREATE");
    printf("\n 2. DISPLAY");
    printf("\n 3. INSERT_BEGIN");
    printf("\n 4. INSERT_END");
    printf("\n 5. INSERT_POSITION");
    printf("\n 6. EXIT");
    printf("\n Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        create();
        break;
      case 2:
        display();
        break;
      case 3:
        insert_begin();
        break;
      case 4:
        insert_end();
        break;
      case 5:
        insert_position();
        break;
      case 6:
        printf("Exiting program.\n");
        exit(0);
      default:
        printf("Invalid choice, please try again.\n");
    }
  }
}
```

```
    return 0;
}

void create() {
    struct node* temp;
    struct node* ptr;

    temp = (struct node*)malloc(sizeof(struct node));
    if (temp == NULL) {
        printf("Memory allocation failed\n");
        return;
    }

    printf("\n Enter data: ");
    scanf("%d", &temp->info);
    temp->next = NULL;
    temp->prev = NULL;

    if (start == NULL) {
        start = temp;
    } else {
        ptr = start;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        ptr->next = temp;
        temp->prev = ptr;
    }
    printf("Node created and added at the end.\n");
}

void display() {
    struct node* ptr;
    if (start == NULL) {
        printf("\n Empty List\n");
        return;
    }
    ptr = start;
    printf("\n List elements are: ");
    while (ptr != NULL) {
        printf("%d ", ptr->info);
        ptr = ptr->next;
    }
    printf("\n");
}

void insert_begin() {
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    if (temp == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
```

```
    printf("\n Enter data to insert at beginning: ");
    scanf("%d", &temp->info);

    temp->next = start;
    temp->prev = NULL;

    if (start != NULL)
        start->prev = temp;

    start = temp;
    printf("Node inserted at the beginning.\n");
}

void insert_end() {
    struct node* temp;
    struct node* ptr;

    temp = (struct node*)malloc(sizeof(struct node));
    if (temp == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("\n Enter data to insert at end: ");
    scanf("%d", &temp->info);
    temp->next = NULL;
    temp->prev = NULL;

    if (start == NULL) {
        start = temp;
    } else {
        ptr = start;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        ptr->next = temp;
        temp->prev = ptr;
    }
    printf("Node inserted at the end.\n");
}

void insert_position() {
    struct node* temp;
    struct node* ptr;
    int pos, i;

    temp = (struct node*)malloc(sizeof(struct node));
    if (temp == NULL) {
        printf("Memory allocation failed\n");
        return;
    }

    printf("\n Enter data to insert: ");
    scanf("%d", &temp->info);
```

```
temp->next = NULL;
temp->prev = NULL;

printf(" Enter position to insert node (starting from 1): ");
scanf("%d", &pos);

if (pos == 1) {
   temp->next = start;
   temp->prev = NULL;
   if (start != NULL)
      start->prev = temp;
   start = temp;
   printf("Node inserted at position 1.\n");
   return;
}

ptr = start;
for (i = 1; i < pos - 1 && ptr != NULL; i++) {
   ptr = ptr->next;
}

if (ptr == NULL) {
   printf("Position out of bounds.\n");
   free(temp);
} else {
   temp->next = ptr->next;

   if (ptr->next != NULL)
      ptr->next->prev = temp;

   ptr->next = temp;
   printf("Node inserted at position %d.\n", pos);
}
```

experiment 7.

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();
int s[MAX];
int top=-1;
int main()
{
int choice
    printf("MENU");
    printf("1. Push");
    printf("2. Pop");
    printf("3. Display");
    printf("4. Exit");
    do{
    printf("Enter your choice:");
    Scanf("%d", &choice);
switch(choice){
    case1: Push();
        break;
    case2: pop();
        break;
    case 3: display();
        break;
    Case 4: exit(0);
    default:
    printf("Invalid choice!");
}
```

```
while (choice != 4);
return 0;
}
void push(){
  int element;
  if (top == Max-1) {
    printf ("overflow");
  }
else {
  printf ("Enter value");
  scanf ("%d", &element);
   top = top+1;
    s[top] = element;
}
}
void pop (){
  int item;
    if (top == -1) {
      printf ("Underflow");
  }
  else {
  item = s[top];
    top --;
  printf ("The deleted element is
          item);
}
}
void display (){
    if (top == -1) {
    printf ("stack Empty");
    }
```

```
else {
  printf ("Sta
    for (int i=
      printf (
  }
}
}
```

2.)

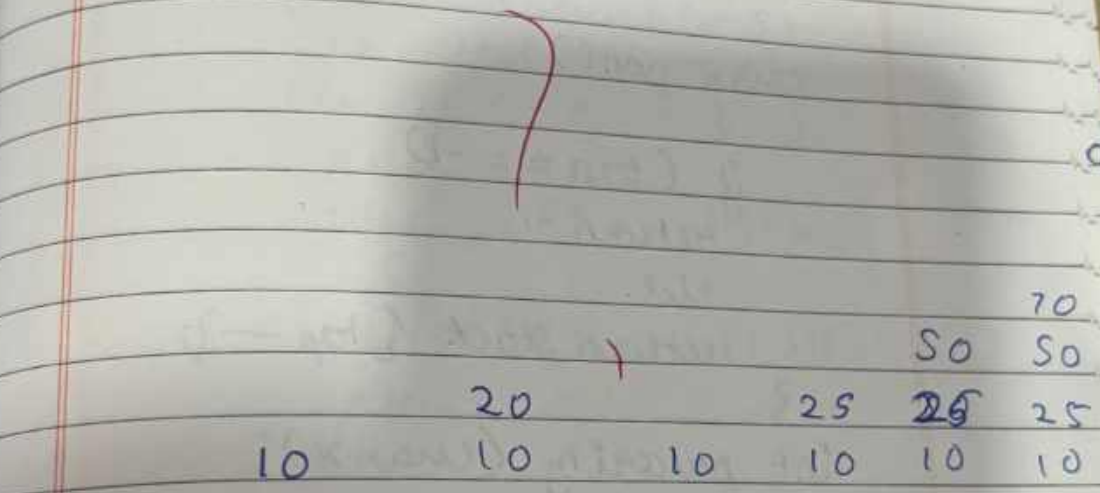
10

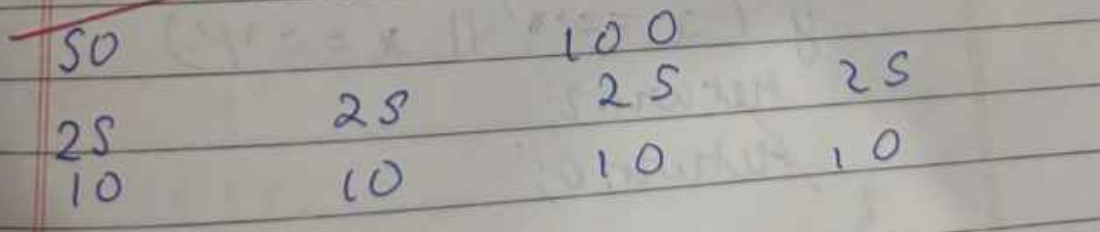
50

25
10

```
else {
    printf("Stack element are: ");
    for(int i=0; i<=top;i++){
        printf("\t %d", s[i]);
    }
}
}
```

2)





experiment 8

```
#include <stdio.h>
#include <ctype.h>
char stack[100];
int top = -1;
void push (char x)
  {
  stack [++top] = x;
  }
char pop()
  {
  if (top == -1)
  return -1;
  else
  return stack [top--];
  }
int priority (char x)
  {
  if (x == '(')
  return 0;
if (x == '*' || x == '-')
  return 1;
  if (x == '*' || x == '/')
  return 2;
  return 0;
}
```

```
int main(){
  char exp[100];
  char *e, x;
  printf("enter the expression:");
  scanf("%s",exp);
  printf("\n");
  e= exp;
  while(*e!='\0')
    {
    if(isalnum(*e))
    printf("%c",*e);
    else if(*e=='(')
    push(*e);
    else if(*e==')')
      {
    while((x=pop())!='(')
    printf("%c",x);
  }
  else
  {
while(pricerity(stack[top])>=
      pricerity(*e))
    printf("%c",pop());
    push(*e);
  }
  e++;
  }
while(top!=-1)
    {
    printf("%c",pop());
  }
```

2.) evaluate Postfix

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define SIZE 40
  int pop();
  void push(int);
char postfix[size];
int stack[SIZE], top=-1;
int main(){
int i, a, b, result, newd;
  char ch;
for(i=0; i<SIZE; i++)
  {
  stack[i]=-1;
  }
printf("enter a postfix
   expression:");
scanf("%s", postfix);
for(i=0; postfix[i]!='\0';i
  {
  ch= postfix[i];
  if( is digit(ch))
  {
  push(ch-'0');
  }
```

```
else if ( ch=='+' || ch=='-' || ch=='*'
        || ch=='/')
{
b=pop();
a=pop();
switch(ch)
{
case '+': result = a+b;
          break;
case '-': result = a-b;
          break;
case '*': result = a*b;
          break;
case '/': result = a/b;
          break;
    }
    push(result);
  }
}

peval = pop();
printf("The postfix evaluation
    is: %d \n", peval);
return 0;
}
void push(int n)
{
if (top<SIZE-1)
  {
    stack[++top]=n;
  }
else{
```

```
printf ("stack is full! \n");
    exit(-1);
    }
}
int pop
{
int n;
if (top>-1)
    {
    n = stack[top];
    stack[top--] = -1;
        return n;
    }
else
{
 printf ("stack is empty!");
exit(-1);
}
}
```

A+(B

Input

H
*
)
G
*
)
F
$
E
/
D
(
-
(
*
B
(
+
A

8. A+(B*(-(D/E$F)*G)*H.

Input	Stack	
H	Empty	
*	*	H
)	*)	H
G	*)	H
*	*)*	HG
)	*)*)	HG
F	*)*)	HG
$	*)*)$	HGF
E	*)*)$	HGF
/	*)*)/	HGFE
D	*)*)/	HGFE$
(	*)*	HGFE$D/
-	*)-	HGFE$D/*
(	*)-(	HGFE$D/*(
*	*)-*	HGFE$D/*(B
B	*)-*	HGFE$D/*(B*-
(	*	HGFE$D/*(B*-*
+	+	HGFE$D/*(B*-*A
A	+	HGFE$D/*(B*-*A+

+A*-*BC*/D$EFGH.

+A*-*BC*/D$EFGH

iv) * + - * + 1 2 / 4 2 1 $ 4 2

2	2
4	4, 2
$	$4^2 = 16$
1	1, 16
2	2, 1, 16
4	4, 2, 1, 16
/	2, 1, 16
1	1, 2, 1, 16
2	1, 2, 1, 16, 2
1	3, 1, 2, 1, 16
+	6, 1, 16
*	5, 16
-	16
+	21

Algorithm

1) Infix to Postfix

1.) Scan Infix expression from left to right
2) Operant → add to postfix directly
3) "(" → push to stack
4) ")" → pop from stack until "(" found.
5) operator → for all operators from stack with greater or equal precedance, then push current operator.

5.) After scanning → pop remaining operator to postfix.

2) Infix to prefix.

1) Reverse the Infix expression
2) swap '(' with ')' in the reversed expression.
3) convert the new expression to postfix during same algorithm as above
4.) Reverse result → that's the prefix expression

3) Evaluation of postfix

1) Scan left → right.
2) operand → push into stack
3) operator → pop top 2 values apply operator push result back.
4) At end → only one value remains in stack result.

4) Evaluate Prefix

1.) Scan expression right → left.
2) operand → Push into stack.
3.) operator → pop top 2 values, apply operation push result back.

4.) At end → only values remaining in stack result

Experiment - 9

Queue Program

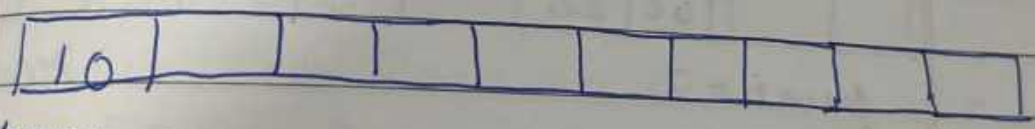
```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 5
void enqueue();
void dequeue();
void display();
int queue[SIZE];
int front = -1, rear = -1;
int main(){
    int ch;
printf("1. Enqueue\n 2. Dequeue
    \n 3. Display \n 4. Exit \n");
do {
printf("\n Enter your choice:");
scanf("%d", &ch);
    switch(ch){
case 1: enqueue();
break;
Case 2: dequeue()
break;
case 3:- display();
break;
case 4: exit(0);
default: printf("\n
    Invalid choice! \n")
    }
}
```

```
while (ch1 = 4)
  return 0;
}
void enqueue (){
int element;
if (rear == size-1){
  printf ("\n Queue is full");
}
else{
printf ("\n Enter element to
  insert: ");
scanf ("%d", &element);
if (fcent == -1)
  front=0;
  rear++
  queue[rear] = element;
  }
}
void dequeue (){
  if (front == -1 || front > rear){
  printf ("\n Queue is empty");
  }
else{
  printf ("\n deleted element: %d
  queue[front];
  front++;
  }
}
```

```
void display ()
{
  if (front == -1 || front > rear)
  {
    printf ("\n Queue is empty.");
  }
  else {
    printf ("\n deleted element : %d",
            queue[
    printf ("\n Queue elements are : \n");
    for (int i = front; i <= rear; i++) {
      printf ("%d \t", queue[i]);
    }
    print ("\n");
  }
}
```

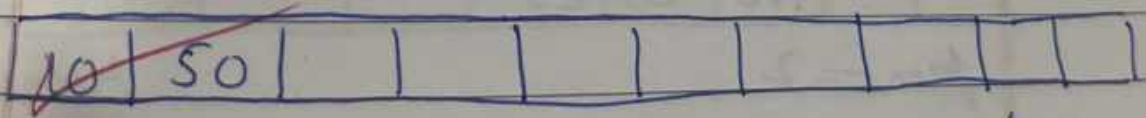
2) Operations on linear queue.

1) Insert (10)



front = 0 Rear = 0

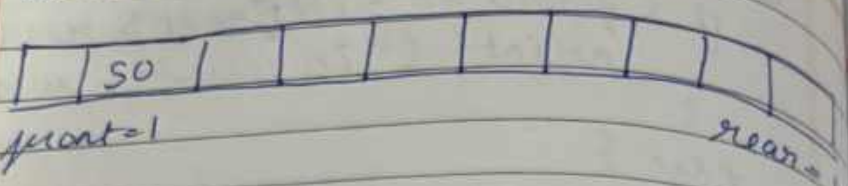
2) Insert (50)



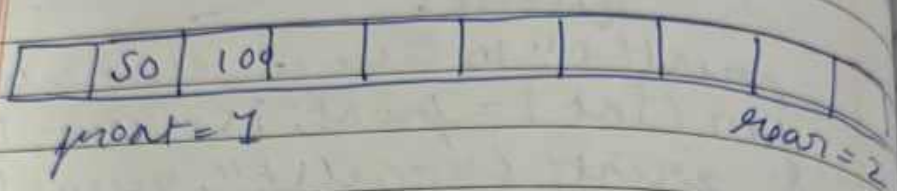
front = 0

rear = 1

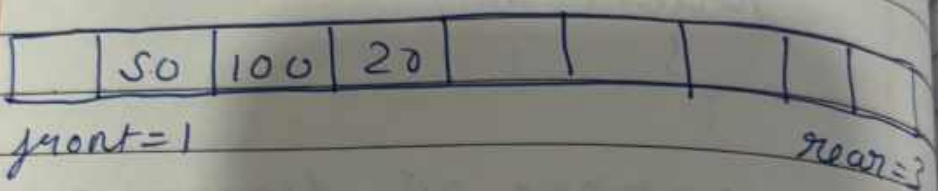
3.) Delete



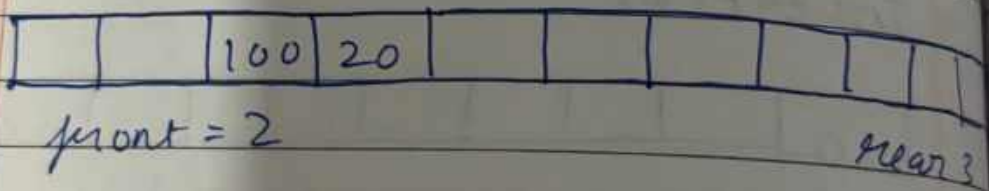
4) Insert(100)



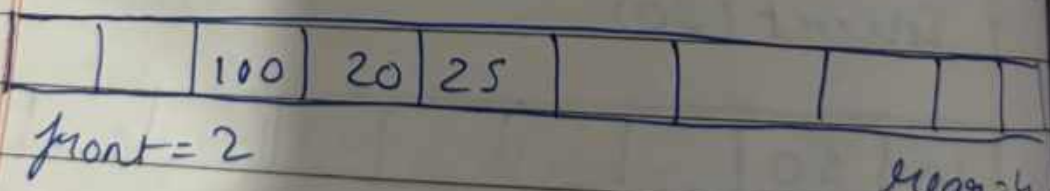
5.) Insert (20)



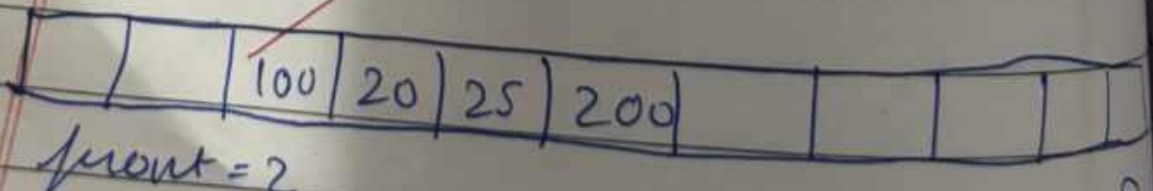
6.) Delete.



7.) Insert (25)



8) Insert (200)



3) Explain
→ A queue
structure
queue b
a queue
the FIF
one th
us

4) Algo

1.) Start
2.) if rear
overfl
3.) if fr
fron
& R
Re
4.) els
(([
5.) En
6.) Pri
7.) Stop

Q) Explain concept of priority queue.

A priority queue is a type of data structure similar to a regular queue but each element is assigned a priority instead of following the FIFO rule, the one that one that entered first is usually served first.

(a) Algorithm: Insertion.

1.) Start

2.) if rear = max-1 or = -2 then print "queue overflows" & exit.

3.) if front = -1 & Rear = -1 then set front = 0 Rear → 0 set Q[Rear] = item & Rear = -1 then set front = 0 Rear = 0

4.) else set Rear ← Rear + 1 - set Q[Rear] ← item

5.) End if.

6.) Print "Insertion Successful"

7.) Stop.

5) Algorithm: Deletion

1) Start
2) If front = -1 or front > Rear, then print "queue underflow" set front = -1 Rear = -1 Exit.
3) Set Item ← Q[FRONT]
4) If front = Rear then set front ← -1 Rear = -1 Exit
5) Else set front = front + 1
6) End if
7) print "Deleted element"
8) Stop.

MW 10/11

Experiment 10

Q.) write code for circular queue.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

void enqueue();
void dequeue();
void display();
int queue[size];
int front = -1, rear = -1;
int main(){
    int ch;
printf("1. Enqueue \n 2. Dequeue
    \n 3. Display \n 4. Exit");
do{
printf("\n Enter your choice:");
scanf("%d", &ch);
switch (ch){
    case 1: enqueue();
        break;
    case 2: dequeue();
        break;
    case 3: display();
    break;
    case 4: exit(0);
default:
printf("Invalid choice
    ! \n");
}
```

```
}
while (ch!=4);
return 0;
}
void enqueue(){
    int value;
    if ((front==0 && rear==SIZE-1)||
    (rear+1==front))
    {
printf("\n Queue is full\n");
}
else
{
printf("\n Enter value to insert:");
scanf("%d", &value);
    if (Front==-1)
        front=0;
rear=(rear+1)%SIZE;
queue[rear]=value;
printf("%d insert into queue",
        value);
}
}
void dequeue(){
    if (front==-1)
    {
    printf("\n Queue is
                Empty");
}
else
{
```

```
printf("\n
if (fron
{
fron
}
else
{
fron
}
}
void
```

```
printf("\n deleted element from
        queue: %d", queue[front]);
if (front == rear)
  {
  front = rear = -1;
  }
else
  {
  front = (front+1) % SIZE;
  }
}
void display () {
        if (front == -1)
          {
  printf("\n Queue is empty!
          \n");
    }
  else
    {
  int i = front;
  printf("\n Queue elements
    are: ");
  while(1){
    printf("%d", queue[i]);
    if (i == rear)
        break;
    i = (i+1) % SIZE;
  }
printf("\n");
}
```

Q.) Write & explain applications of priority queue

(1) Traffic light control:
Manages signal priorities based on real time traffic sensor

(2) Operating system algorithms used for scheduling process to execute high priority first

(3) Huffman codes:
Priority queue helps build Huffman trees for efficient data compression.

Q.) What are advantages of circular queue.

1) Efficient memory utilization
→ Unlike linear queue, space is reused after deletion

(2.) Overflow prevention
→ In linear queue, once rear reaches the end.

(a) Insertion algorithm.

1) if front=0 & rear = max-1 the queue will overflow.
2) If front is rear+1 it will also overflow queue.
3) If front = -1 and rear = -1, front = rear = 0 else if rear = max-1 and front != 0 Rear = 0
4) Else, rear = rear + 1
   End
5) queue[rear] = val
6) exit.

(b) Deletion.

1) If front = -1 write underflow
2) Go to step 4 at end of if
3) set val = queue[front]
4) If front = rear, set front = rear = -1
5) Else, if front = max-1, set front = 0
6) Else, set front = front + 1
7) Exit.

experiment 11

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node {
    int key;
    struct Node * left * right
} Node;

Node * newnode (int key) {
    node * t = (Node*) malloc
        (sizeof (Node));
    t -> key = key; t -> left = t ->
        right = Null;
    return t;
}
Node * insert (Node* root, int key)
if (! root) return newnode (key);
if (key < root => key) root -> left
= insert (root -> cleft, key);
else if (key > root => key) root -> right
= insert (root -> right, key);
else printf ("key %d already exists,
        key);
    return root;
}
Node * min value Node (Node * root)
{
Node* cur = root;
while (cur && cur -> left)
cur = cur -> left;
```

return cur;
}
Node * deleteNode (Node* root, int key){
if (!root) return NULL
if (key < root → key) root → left = deleteNode (root → left, key);
else if (key > root → key) root → right = deleteNode(root → right);
else {
if (!root → left) {
Node * = root → right;
free (root);
return r;
} else if (!root → right) {
Node * l = root → left;
free (root);
return l;
} else {
Node * succ = minValueNode (root → right);
root → key = succ → key;
root → right = deleteNode (root → right, succ → key);
}
}
return root;
}

```
int search(Node* root, int key){
  while(root){
    if(key == root->key) return
root = (key < root->key) ? root->
  left : root->right;
    }
  return 0;
  }
void inorder(Node*){ if(){ inorder
  r->left); printf("%d", r->key
inorder(r->right); }}
void preorder(Node*r){ if(r){
  printf("%d", r->key); preorder(r->
  preorder(r->left); preorder(r->
  }}
void postorder(Node*r){ if(r){post
  order(r->left); postorder(r->ri
printf("%d", r->key); }}

typedef struct{
  Node** a;
int front, rear, cap;
  }Queue;
Queue *q(.create(int cap){
  Queue *q = (Queue*)malloc
  (sizeof(Queue));
  q->a = (Node**)malloc(sizeof
                Queue));
q->front = q->rear = 0; q->cap
          = cap;
        return q;
}
```

```
int empty(Queue *q) { return q->
    front == q->rear; }
void q(Queue *q, Node* x) {
    if ((q->rear+1)%q->cap == q->
                (front) {
int n = (q->cap - q->cap * 2;
}
free(q->a);
q-> = b; q->cap = new(ap; q->
front = 0; q->rear = id;
}
Node q free(Queue *q) { free(q->a)
; free(q); }
void level order(Node* root)}
    if (!root) { printf("(empty)\n");
        return; }
    Queue* q = q(create(8);
    Push(q, root);
}
printf("\n");
q.free(q);
}
void freenode(Node* n){
    if (!) return;
free free(n->left);
free free(o->right);
free(o);
}
```

```
int main(void){
    Node* root = NULL;
        int ch, x;
        do {
    void menu(){
        printf("BST Menu");
        printf("1. Insert");
        printf("2. Delete");
        printf("3. search");
        printf("4. display");
        printf("5. inorder");
        printf("6. preorder");
        printf("7. post order");
        printf("8. level order");
        printf("9. del");
        printf("10. exit");
        printf("enter choice:");
    if (scanf("%d", &ch)!=1)
        break;
switch(ch){
    case 1:                          insert
        printf("enter key to delete");
        scanf("%d", &x);
            root = insert(root, x);
            break;
    case 2:
    printf("Enter Key to delete");
    scanf("%d", &x);
        root = deletenode(root, x);
        break;
```

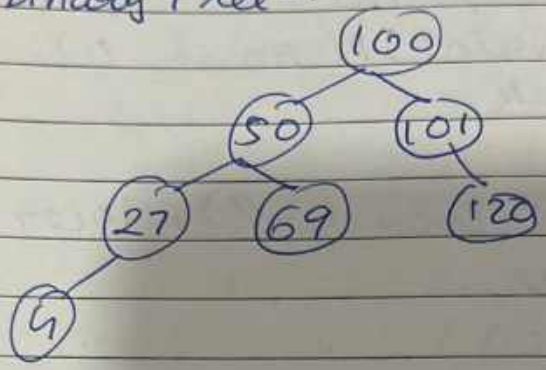
```
case 3:
printf("Enter key to search: ");
scanf("%d", &x);
printf("%s", search(root, x));
break;
case 4:
if(!root) printf("Empty tree");
else printf("%d", root->data);
break;
case 5:
printf("Inorder"); inorder(root);
break;
case 6:
printf("Preorder"); preorder(root);
break;
case 7:
printf("postorder"); postorder
(root);
break;
case 8;
printf("level order: "); levelorder
(root);
case 9:
free tree(root); root=NULL;
printf("Cleared");
break;
case 10
break;
default:
printf("Invalid");
}
```

while (ch!=0);
free tree (root);
return 0;
}

Theory :-

a) 100 50 101 27 120 4 69

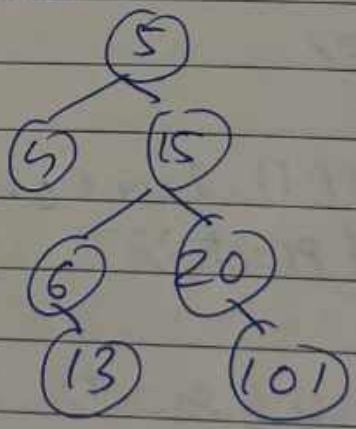
Binary Tree :-



Pre - 100, 50, 24, 4, 69, 101, 120
In - 4, 27, 50, 69, 100, 101, 120
Post - 4, 27, 69, 50, 120, 101, 100

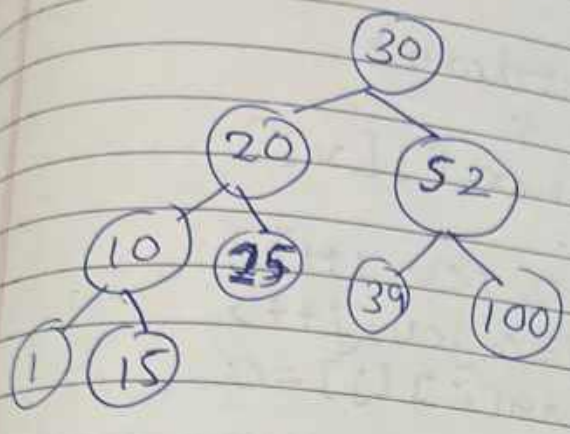
b) 5, 15 4 6 20 13 101

Binary Tree :-



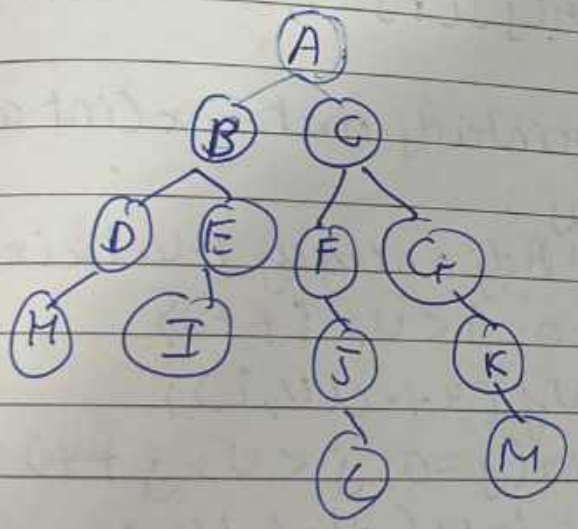
Pre - 5, 4, 15, 6, 20, 13, 101
In - 4, 5, 6, 13, 15, 20, 101
Post - 4, 13, 6, 101, 20, 15, 5.

c) 30, 20, 10, 25, 15, 52, 39, 1, 100



Pre: 30 20 10 1 15 25 52 39 100
In: 1 10 15 20 25 30 39 52 100
post: 1 5 10 25 20 39 100 52 30

d)



10/11

Pre - A, B, D H E C F J G K L M
In: H D B E A F L J.
Pre: A B D H E I C F J G K L M
In: H D B E I A J F C L K M G
Post: H D I E B J F L M K G C A

Experiment 12.

```
#include <stdio.h>
# define V 5
void init (int arr [][V])
{ int i, j;
  for (i=0; i<V; i++)
    for (j=0; j<V; j++)
      arr[i][j]=0;
  }
void insertedge (int arr[][V], int i, int j)
  {
    arr[i][j]=1;
    arr[j][i]=1;
  }
void printadjmatrix (int arr[][V])
  int i, j;
printf ("Adjancency Matrix:\n");
for (i=0; i<V; i++){
  printf ("%d", i);
    for (j=0; j<V; j++){
      printf (" %d", arr[i][j]);
      }
    printf ("\n");
    }
  }
int main()
{
int adjmatrix[V][V];
    init (adjmatrix);
```

inserted
"

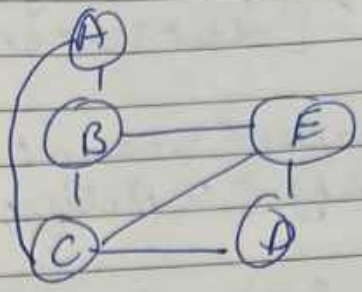
```
insertedge(adjmatrix,0,1);
''      ''          ,0,3)
                    ,0,2)
                    ,0,4)
                    ,1,3)
                    ,2,0)
                    ,2,3)
                    ,2,4)
                    ,3,0)
                    ,3,1)
                    ,3,2)
                    ,3,4)
                    ,4,0)
                    ,4,2)
                    ,4,3)
printadjmatrix(adjmatrix);
return 0;
}
```

2.)
```
#include<stdio.h>
#include<stdlib.h>
#define VS
struct Node{
    int dest;
    struct Node* next;
};
struct graph{
    struct Node* adj[V];
};
struct Node* newNode(int dest)
{ struct Node* temp = (struct Node*)
malloc(sizeof(struct Node));
temp->dest = dest;
temp->next = NULL;
return temp;
}
struct Graph* creatGraph(){
  struct Graph* graph = (struct
      Graph*) malloc(sizeof
      (struct Graph));
for(int i=0; i<V; i++)
    graph->adj[i] = NULL;
    return graph;
    }
Void addEdge(struct Graph*
  graph, int src, int dest)
{ struct Node* temp = newNode
(dest);
temp->next = graph->adj[src];
graph->adj[src] = temp;
```

```
temp = newNode(src);
temp -> next = graph -> adj[dest];
graph -> adj[dest] = temp;
}
void printGraph(struct Graph*
graph) { printf("Adjacency list
\n");
for(int i=0; i<V; i++){
    printf("%d -> ", i);
struct Node* temp = graph -> adj[i]
  while(temp){
  printf("%d", temp -> dest);
  temp = temp -> next;
  }
  printf("\n");
  }}
int main(){
struct Graph* graph = create
  Graph();
addEdge(graph, 0, 1);
   "    "     0, 2);
"       "   0, 3);
   "    "     0, 4);
   "    "     1, 3);
   "    "     2, 3);
"       "     2, 4);
"       "     3, 4);
print Graph(graph);
  return 0;
  }
```

Theory

a.)



	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	0	1
C	1	1	0	1	1
D	0	0	1	0	1
E	0	1	1	1	0

A → [B] → [C] ↘

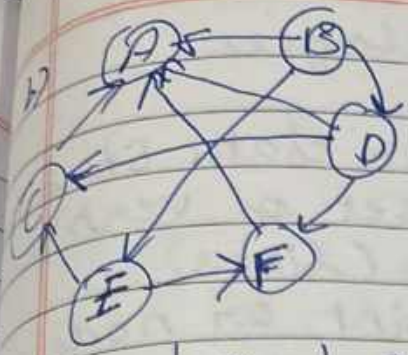
B → [A] → [D] → [E] ↘

C → [A] → [B] → [D] → [E] ↘

D → [C] → [E] ↘

E → [B] → [C] → [D] ↘

b)



	A	B	C	D	E	F
A	0	0	0	0	0	0
B	1	0	0	1	1	0
C	1	0	0	0	1	0
D	0	0	1	0	0	1
E	0	0	1	0	0	1
F	1	0	0	0	0	1

A → NULL

B → [A] → [D] → [E] ↴

C → [A] → [E] ↴

D → [C] → [F] ↴

E → [C] → [F] ↴

F → [A] → [F] ↴

(·) Graph Terminologies

1.) Graph → non-linear data structure consisting of a set of vertices (nodes) & edges (links)
2.) Vertex → is a point or node in the graph where edges meet
3.) Edge → is a connection or link between two vertices
4.) Adjacent Vertices → Two vertices are adjacent if there is a direct edge connecting them

~~5.) Path~~

27/10/11