Bisakh Mondal
001810501079

ASSIGNMENT 1

Question 1:

Problem:

Write A Program To Compute The Factorial Of An Integer N Iteratively And Recursively. Check When There Is Overflow In The Result And Change The Data Types For Accommodating Higher Values Of Inputs.

APPROACH:

We take an integer n as input from the user. And use both the methods to find the factorial of the number.

Pseudo Code:

The Iterative algorithm:

step 1. Start

step 2. Read the number n

step 3. Initialise i=1, fact=1

step 4. Repeat step 4 through 6 until i=n

step 5. fact=fact*i

step 6. i=i+1

step 7. Print fact

step 8. Stop

The Recursive algorithm*:*

```
Fact(n)

Begin

 if n == 0 or 1 then

   Return 1;


 else Return n*Call Fact(n-1);
 endif

End
```

REPORT:

As n exceeds 18 and starts increasing, overflow occurs even if the factorial is stored in unsigned int. We may use unsigned int in place of int or use long to store the factorial, so as to accommodate higher values of inputs.

Question 2:

Problem:

Write a Program to Generate the Nth Fibonacci Number Iteratively and Recursively. Check When There Is Overflow in the Result and Change the Data Types for Accommodating Higher Values of Inputs**.**

APPROACH:

We take an integer n as input from the user. And use both the methods to find the nth Fibonacci term.

Pseudo Code:

The Iterative algorithm:

- Start
- Declare variables i, a,b , fib
- Initialize the variables, a=0, b=1, and fib =0
- Enter the number of terms of Fibonacci series to be printed
- Print First two terms of series
- Initialise i=2
- Use loop for i less than or equal to the value of n fib=a+b a=b b=fib increase value of i each time by 1 print the value of show
- End

The Recursive algorithm:

```
Fibo(n)

Begin
  if n <= 1 then

     Return n;


  else Return Call Fibo(n-1) + Call Fibo(n-2);

  endif

End
```

REPORT:

As n exceeds 46(n>46) and starts increasing, overflow occurs . We may use unsigned int in place of int or use long to store the Fibonacci term, so as to accommodate higher values of inputs.

Question 3:

3)problem:

WRITE A PROGRAM FOR LINEAR SEARCH AND BINARY SEARCH FOR SEARCHING INTEGERS, FLOATING POINT INTEGERS AND WORDS IN ARRAYS OF RESPECTIVE TYPES.

APPROACH:

We take the maximum size of the array and the array elements as input from the user and then conduct linear search or binary search.

ALGORITHM/PSEUDOCODE:

The Linear search(algorithm):

Step 1: Set i to 1 Step 2: if i > n then go to step 7 Step 3: if A[i] = x then go to step 6 Step 4: Set i to i + 1 Step 5: Go to Step 2 Step 6: Print Element x Found at index i and go to step 8 Step 7: Print element not found Step 8: Exit

The Binary search(pseudocode):

```
Procedure binary_search

  A ←  sorted array
  n ←  size of array

  x ←  value to be searched



  Set lowerBound = 1

  Set upperBound = n



  while x not found

    if upperBound < lowerBound
```

```
    EXIT: x does not exists.

  set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

  if A[midPoint] < x
    set lowerBound = midPoint + 1

  if A[midPoint] > x
    set upperBound = midPoint - 1

  if A[midPoint] = x
    EXIT: x found at location midPoint
 end while
end procedure
```

REPORT:

In general, linear search takes linear time and hence it is slower than binary search which takes logarithmic time. However for conducting binary search on an array, it needs to be sorted initially.

Question 4:

PROBLEM:

Write a program to generate 1,00,000 random integers between 1 and 1,00,000 without repetitions and store them in a file in character mode one number per line. Study and use the functions in C related to random numbers.

Approach:

a)We have generated 1,00,000 random numbers.

b) After generating a number, we checked whether it was previously generated.

c)Then we have once used another functions relating to random numbers.

Pseudo Code:

```
long int i,num;

int arr[max];

//set all array elements to 0

//open file

while(i<max){

   num=rand()%max;

   if(arr[num]) continue; //check for repetition

        else{

        ++i;

         arr[i]=1;

        //write in file

   }

} srand(time(0)); // use other functions relating to random numbers
```

```
fclose(fp);
for(i=0;i<5;++i)

    printf("%d\n",rand());
```

Limitations:

We have not used all the functions relating to random numbers.

Question 5:

Problem:

Write a program to generate 1,00,000 random strings of capital letters of length 10 each,without repetitions and store them in a file in character mode one string per line.

Approach:

a)We generate a random string of 10 letters.

b)Then we check for repetition.

c) If it is unique then we store it in a file.

Pseudo Code:

```
char *str[1000]; char temp[11];

for(i=0;i<1000;++i){

        for(k=0;k<10;++i) temp[k]=(char)('a'+(rand()%26));

for(j=0;j<i;++i){

                if(strcmp(temp,str[j])==0) continue; //check for repetition

                else str[i]=temp //store char by char
```

```
        //write string into file

        //insert newline character after string
}
```

Limitations:

Since the comparison takes total 10*O(n^2) time so we limit maximum number of strings to 1000. So the maximum complexity will be well below the time limit of 1 second.

Question 6:

Problem: Store the names of your classmates according to roll numbers in a text file one name per line. Write a program to find out from the file, the smallest and largest names and their lengths in number of characters. Write a function to sort the names alphabetically and store in a second file.

Approach:

We store the name of a student of a class according to their roll numbers, i.e., 1 before 2 and so on.

Then we transfer the record to the file specified.

We read the names from the file and transfer them to an array of strings.

Find the smallest and largest length.

Use the strcmp function to find out the largest and smallest string.

Pseudo Code:

```
//define number of students to be 50

struct record{

    char name[11]; int roll;
```

}arr[50],temp;

for(i=1;i<=50;++i){

//take name as input

//store name and roll in temp structure and write them into the file

} for(i=0;i<50;++i){

//read names from file and store them according to their length

} //for strings having maximum length use strcmp function to find out which string has a non negative value with every other string

//for strings having minimum length use strcmp to find out the which string has a non positive value //with every other string

Limitations:

The maximum number of students has to be known at the time of compilation. If it is determined during runtime then we need to use malloc or other runtime functions.

Question 7:

7)problem: TAKE A FOUR DIGIT PRIME NUMBER P. GENERATE A SERIES OF LARGE INTEGERS L AND FOR EACH MEMBER OF L COMPUTE THE REMAINDER R BY DIVIDING L WITH P. TABULATE L AND R. REPEAT FOR SEVEN OTHER FOUR DIGIT PRIME NUMBERS KEEPING L FIXED.

APPROACH:

Take an array of large integers L .Take a prime number as input. Proceed as instructed.

ALGORITHM:

Step 1:Generate L[], Declare R[] , Initialise j=0

Step 2:Take P as input

Step 3:Check if P is prime

Step 4:Initialise i=0

Step 5:Use loop for i less than the length of L

R[i]=L[i] modulus P

Print L[i] and R[i] in a tabular form

Increment i by 1

Step 6:Increment j by 1

Step 7:Goto step 2 if j less than equal to 8

Step 8:Exit

REPORT:

The values of L and R are tabulated for 8 prime numbers, where L corresponds to the array of large integers and R corresponds to the remainder when L is divided by P, the prime number.

Question 8:

8) problem: Convert your name and surname into large integers by juxtaposing integer ASCII codes for alphabet. Print the corresponding converted integer. Cut the large integers into two halves and add the two halves. Compute the remainder after dividing by the prime number P in the previous problem.

Approach:

First we replace each character of our name/surname by its equivalent ASCII code. Thus we generate the integer.

Then we find out the number of digits in the integer and divide it into two halves.

We then add the two halves and generate two other integers.

Finally, we compute the remainder after dividing it by the prime number P.

Pseudo Code:

```
//Take the prime P generated in the previous question

//take name and surname as input

int nameint[3*strlen(name)],surnameint[3*strlen(surname)];

for(i=0;name[i]!='\0';++i){

        int num=(name[i]-'a')+(int)a;

        if(num>=100)

                //add the three digits to the nameint array

                name_digits+=3;

        else

                //add the two digits to nameint

                name_digits+=2;

} //do similarly for surname and create the surnameint array

//divide the array into two halves

//nameint1[],surnameint1[] are the other two arrays
//keep one half of name/surname in nameint and transfer the other half to nameint1

//generate the required numbers and add them

//compute the remainder by dividing it by P
```

Assignment – 2

Question 1:

problem:

Define ADT for polynomials.

Write C data representation and functions for operations on polynomials in Header file.

Write menu driven main program in separate file for testing different operations.

Approach:

We have used coefficient representation of polynomials.We can represent a polynomial by its coefficients stored according to the ascending order of degree of the variable (x), associated with the coefficient. Static arrays have been used to store the coefficients.

We have used following operations in our polynomial ADT

1. Is-zero – returns true if polynomial is zero. 2. Coef – returns the coeff. of a specified exponent. 3. add - add two polynomials 4. mult - multiply two polynomials 5. Cmult - multiply a polynomial by a const. 6. attach – attach a term to a polynomial 7. remove – remove a term from a polynomial 8. degree – returns the degree of the polynomial

Pseudo Code:

```
        typedef struct{

                int* a;

                int order;

        } polynomial;
int Iszero(polynomial* p) {

return p->order == 0;

} int coeff(polynomial* p, int indx) {

   if(p->order < indx) return INT32_MIN;
```

```
    else return *(p->a + indx);

} polynomial add(polynomial* a, polynomial* b)

{

    polynomial p;__init__(&p, max(a->order, b->order));


    int i;

    for(i = 0; i < p->order; i++)

    {

        if(i >= a->order) *((p->a) + i) = *((b->a) + i);


        else if(i >= b->order) *((p->a) + i) = *((a->a) + i);


        else *((p->a) + i) = *((a->a) + i) + *((b->a) + i);

    } return p;

} polynomial subs(polynomial* a, polynomial* b)
{

    polynomial p;__init__(&p, max(a->order, b->order));


    int i;

    for(i = 0; i <= p->order; i++)

    {

        if(i > a->order) *((p->a) + i) = *((b->a) + i) * -1;

        else if(i > b->order) *((p->a) + i) = *((a->a) + i);
```

```
      else *((p->a) + i) = *((a->a) + i) - *((b->a) + i);

   } return p;

} polynomial mult(polynomial* a, polynomial* b) {

   polynomial p; __init__(&p, a->order + b->order);

   int i, j;

   for(i = 0; i <= a->order; i++) for(j = 0; j <= b->order; j++) *((p->a) + (i + j)) += *((a->a) + i) * *((b->a)
   + j);

   return p;

} void cmult(polynomial* a, int c) {

   int i;

   for(i = 0; i < a->order; i++) *((a->a) + i) *= c;

} int degree(polynomial* p) {return p->order;}


void attach(polynomial* a, int x, int order)
{

   if(order <= a->order) return;

   int i, old_order = a->order;

   int *temp;

   temp = a->a;

  __init__(a, order);


   *((a->a) + order) = x;

   for(i = 0; i <= old_order; i++) *((a->a) + i) = *(temp + i);

   free temp;
```

```
} void remove(polynomial* a, int order)

{

   if(order > a->order) return;

   if(order < a->order) {*((a->a) + order) = 0; return;}


   int* temp = a->a;

   int* x;


   x = (int*)malloc(a->order*sizeof(int));


   for(i = 0; i <= order; i++) *(x + i) = *(temp + i);


   a->a = x; free temp;

} void display(polynomial p){
   int i;

   printf("%d", *(p.a));

   for(i = 1; i <= p->order; i++) printf(" + %dX^%d", *(p.a + i), i);


   printf("\n");
```

} **Limitations:**

If we have too many trivial coefficients we end up wasting a lot of space. So a better representation would be to use dynamic lists to store the coefficients. But that would also affect the performance of the operations on polynomials.

Question 2:

problem:

Define an ADT for Sparse Matrix.

Write a C data representation and functions for operations on sparse matrix in a C header file.

Write a separate menu-driven program to check different operations on Sparse Matrix.

Approach:

Sparse matrix is a matrix which contains very few non-zero elements. It is natural to represent matrices as 2-d arrays. But for sparse matrices this involves wastage of a lot of memory space. The operations like transpose, add, multiply also takes a lot of time.

So,

An alternative approach:

Store the nonzero elements of a sparse matrix in the form of

3-Tuples (i, j, val) in an array.

i = row-position

j = column position
val = value at position (i, j) [nonzero].

[The first triple contains (m, n, t), for an (m x n) matrix having t non-zero values]

Pseudo Code:

```
int get_dim( )
            {
            int temp; printf("Enter the dimension of your matrix: ");
            scanf("%d",&temp); return temp; } int get_bound() {
            int temp; printf("Enter the bound to create sparse array for matrix: ");
            scanf("%d",&temp); return temp; } void get_data(int sparse[],int n,int
            &last_idx) {
```

```
        int i=0,choice,a,b,data; printf("Enter 1 to add new element or 2 to
        exit: "); scanf("%d",&choice); while(choice==1) {
    printf("Enter the location and data(row,column,data): ");
    scanf("%d%d%d",&a,&b,&data); sparse[3*last_idx+0]=a;
    sparse[3*last_idx+1]=b; sparse[3*last_idx+2]=data; last_idx+=1;
    printf("Enter 1 to add new element or 2 to exit: "); scanf("%d",&choice); }
    } void insert_new_data(int sparse[],int n,int &last_idx) {
        int i=0,choice,a,b,data; printf("Enter 1 to add new element or 2 to
        exit: "); scanf("%d",&choice); while(choice==1) {
            printf("Enter the location and data: ");
            scanf("%d%d%d",&a,&b,&data); int c=0,f=0;
            for(c=0;c<last_idx;i++) {
                if(sparse[3*c+0]==a&&sparse[3*c+1]==b) {
        printf("Do you want to modify(1-YES\t2-NO)!!!\n");
                    scanf("%d",&f); if(f==1)
sparse[3*c+2]=data; f=1; } } if(f==0) {
    sparse[3*last_idx+0]=a; sparse[3*last_idx+1]=b; sparse[3*last_idx+2]=data;
    } last_idx+=1; printf("Enter 1 to add new element or 2 to exit: ");
    scanf("%d",&choice); } } void modify_data(int sparse[],int n,int last_idx)
    {
        int i=0,choice,a,b,data; printf("Enter 1 to modify element or 2 to exit:
        "); scanf("%d",&choice); while(choice==1) {
            printf("Enter the location and new data: ");
            scanf("%d%d%d",&a,&b,&data); for(i=0;i<last_idx;i++) {
                if(sparse[i*3+0]=='a'&&sparse[i*3+1]=='b') {
            sparse[i*3+2]=data; break; } } if(i==n)
printf("Data doesn't exist\n"); printf("Enter 1 to modify element or 2 to exit: ");
    scanf("%d",&choice); } } void delete_data(int *sparse,int n,int last_idx) {
        int i=0,choice,a,b,data; printf("Enter 1 to delete element or 2 to exit:
        "); scanf("%d",&choice); while(choice==1) {
            printf("Enter the location : ");
            scanf("%d%d%d",&a,&b,&data); for(i=0;i<last_idx;i++) {
                if(sparse[i*3+0]=='a'&&sparse[i*3+1]=='b')
  sparse[i*3+2]=-1; } printf("Enter 1 to delete element or 2 to exit: ");
    scanf("%d",&choice); } }
```

Limitations:

This approach is only valid for sparse matrices which mostly have 0 has the value of (i,j). If the frequency of non-zero elements is more, this approach is not efficient.

Question 3:

problem:

Define an ADT for List.

Write a C representation and functions on the List in header file.

Also write a menu-driven program for testing different operations and include above header.

Approach:

We can represent list by static array in the following methods:

1. Using sentinel: We put a sentinel (generally an extreme value) at the end of a list. Whenever we encounter the sentinel we can identify the end of the list. **2. Without sentinel:** We can have a parameter which can denote the current length of the list.

Some of the operations performed on lists are:

• 1. find out the length of a list

• 2. read the list from either direction

• 3. retrieve the i-th element

• 4. store a new value into the i-th position

• 5. insert a new element at position i
• 6. delete the element at position i

• 7. search the list for a specified value

• 8. sort the list in some order on the value of the elements.

Pseudo Code:

Representation:

With sentinel:

```
typedef struct {
        int static_array[_MAX_SIZE_];
} list;

void _init_l(list* a){
        a->static_array[0] = _SENTINEL_VAL_;
}
```

Without sentinel:

```
typedef struct{
        int static_array[_MAX_SIZE_];
        int size;
}list;
void _init_l(list* a){
        a->size = 0;
```

```
} void insert(list* a, int indx, int value){
        // Sentinel
        i = indx
        while static_array[i] != _SENTINEL_VAL_;
                i++
// reverse order
for j in range i + 1 to indx
        swap value at indx j and j + 1

static_array[indx] = value



} int search(int value){

I = 0

While static_array[I] != _SENTINEL_VAL_;
        If static_array[I] == value:
                Return I
        I++
Return -1;

} int length(){
        I = 0;
While static_array[I] != _SENTINEL_VAL_:
```

I++

Return I;

}

Limitations:

As we use static arrays to store value we end up wasting a lot of space. We can resolve it by using dynamic data types.

Question 4:

problem:

Define an ADT for set.

Write a C representation and functions for operation on set.

Write a menu-driven main program separate from header file to check the operations on set.

Approach:

We have used static array for implementation of set ADT. We have used following methods in set:

Insert (insert new element in the set)

Find (search for any value)

Delete (delete previously existing value)

Pseudo Code:

```
typedef struct {
  float term[MAX_ELEMS];
  int num_terms;
  int iterator;
```

```
} SET;

/* The following function will give 1 if the element is present
                    will give 0 if the element is not present*/

int is_element_of(float x, struct SET S){
   for(int i=0; i<S.num_terms; i++)
      if(S.term[i] == x) return 1;
   return 0;
} /* The following function checks whether the Set is empty or not
                          returns 1 if empty else 0*/

int is_empty(struct SET S){
   if(S.num_terms == 0) return 1;
   else return 0;
} /* This function will return the number of elements in the set*/

int size_SET(struct SET S){
   return S.num_terms;
} /* This function will return one value of S in each call
   This will give us a value if we haven't reached the end
   else start from beginning*/

float iterate(struct SET *S){
```

```
   if(S->iterator < S->num_terms){
      S->iterator += 1;
      return S->term[S->iterator-1];
   } else{
      S->iterator = 0;
      return S->term[S->iterator];
   }
} /* The following function creates a new,
  initially empty set structure */

void create(struct SET* S){
   for(int i=0; i<MAX_ELEMS; i++)
      S->term[i] = 0;

   S->num_terms = 0;
   S->iterator = 0;
   return;
} /* This function will add elements till the set is full */

void add(struct SET *S, float x){
   if(S->num_terms < MAX_ELEMS) {
      if(is_element_of(x,*S)) return;
      S->term[S->num_terms] = x;
      S->num_terms += 1;
```

} **Limitations:**

No such distinguishable limitation was observed for this approach.

Question 5:

problem:

Define an ADT for string.

Write C data representation and functions for operation in string in header file.

Write a menu-drive main program to test various operations and include the header file.

Approach:

We will implement string ADT by static char array. We have used the following methods in string ADT.

Create a string from c style char array

Add to strings to generate new string

Create substring from a given string

Find char at particular location

Pseudo Code:

```
struct String {
             int maxLength; int length; char *str; }; void init(struct String **s , int
                                                                                maxlen) {
            if(maxlen > 1000000) {
            printf("Memory Insufficient.\n Try smaller values\n"); return; } else { *s
            = (struct String *)malloc(sizeof(struct String));
         (*s)->maxLength = maxlen; (*s)->length = 0; (*s)->str = (char
```

```
*)malloc((*s)->maxLength * sizeof(char)); } } void insert(char *c , struct
String **s) {
    // Inserting a string int len = 0; while(c[len]) len++; if(len > 1000000)
    {
    printf("Given string too large to be inserted.\n"); return; } for(int i =
    0; i < len; i++) {
(*s)->str[i] = c[i]; } (*s)->str[len] = '\0'; (*s)->length = len; } void
join(struct String **s , char *s1) {
    for(int i = 0; s1[i]; i++) {
(*s)->str[(*s)->length++] = s1[i]; } (*s)->str[(*s)->length] = '\0'; } char at(int
index , struct String *s) {
    if(index <= 999999) {
            return (s->str[index]); } else { printf("Out of Bounds.\n");
return '\0'; } } void eraseAt(int index , struct String **s) {
    if(index <= 999999) {
        for(int i = index+1; i < (*s)->length; i++) {
(*s)->str[i-1] = (*s)->str[i]; } (*s)->str[(*s)->length-1] =
(*s)->str[(*s)->length]; (*s)->length--; } } void substr(int l, int r, struct
String *s, char **sub) {
    if(l > r) {
    printf("Invalid Range.\n"); return; } else { if(r <= 999999) {
int lim = (r <= s->length-1)?r:s->length-1; *sub = (char
*)malloc((lim-l+1)*sizeof(char)); for(int i = l , k = 0; i <= lim; i++ , k++)
(*sub)[k] = s->str[i]; } } } void showString(struct String *s) {
for(int i = 0; i < s->length; i++) printf("%c" , s->str[i]); printf("\n"); }
void destruct(struct String **s) {
free((*s)->str); }
```

Limitations:

No such distinguishable limitation was observed for this approach.

Question 6:

problem:

Given a large single dimensional array of integers, write function for sliding window filter maximum, minimum, median, and average to generate an output array. The window should be an odd integer like 3, 5 or 7. Explain what you will do with boundary values.

Approach:

We will create an ADT which contains average value, maximum, minimum and median for the sliding window at a certain instance.

We are given a sub-segment of the array which is covered by the sliding window at a certain point we can calculate the above mentioned parameters.

Code:

```
#include <stdio.h>


#define MAX_ELEMS 10

#define SLIDER_MAX 5


typedef struct out_char{

   float average;

   int max;

   int min;

   int median;

} out;


typedef struct slider_tag{

   int val;
```

```
    int index;
}slider_elem;


void add_remove(slider_elem slider[SLIDER_MAX],slider_elem add_elem, int rmv_index){
    int i;
    for(i=0 ;i<SLIDER_MAX; i++)
        if(slider[i].index == rmv_index){
            slider[i].index = add_elem.index;
            slider[i].val = add_elem.val;
            break;
        }

    for(int i=0; i<SLIDER_MAX; i++)
        for(int j = i+1; j<SLIDER_MAX; j++){
            if(slider[i].val > slider[j].val){
                slider_elem temp;
                temp = slider[i];
                slider[i] = slider[j];
                slider[j] = temp;
            }
        } return;
} void slider_mmma(int input[MAX_ELEMS], out output[MAX_ELEMS - SLIDER_MAX+1]){
    slider_elem slider[SLIDER_MAX];
    for(int i=0; i<SLIDER_MAX; i++){
```

```
    slider[i].index = 0;
    slider[i].val = input[0];
} int average = 0;
for(int i=0; i<MAX_ELEMS; i++){
    slider_elem S;
    S.val = input[i];
    S.index = i;
    if(i<SLIDER_MAX-1){
        add_remove(slider,S,0);
        average += input[i];
      }
    else{
      if(i == SLIDER_MAX-1){
        add_remove(slider,S,0);
        average += input[i];
      } else{ add_remove(slider,S,i-SLIDER_MAX);
         average = average - input[i-SLIDER_MAX] + input[i];
      } output[i-SLIDER_MAX+1].min = slider[0].val;
      output[i-SLIDER_MAX+1].max = slider[SLIDER_MAX-1].val;
      output[i-SLIDER_MAX+1].median = slider[(SLIDER_MAX-1)/2].val;
      output[i-SLIDER_MAX+1].average = average/(SLIDER_MAX + 0.0);
    }
} return;
```

```
} int main(){
   int input[MAX_ELEMS] = {0};
   printf("Enter %d elements below :: \n",MAX_ELEMS);

   for(int i=0; i<MAX_ELEMS; i++)
      scanf("%d",&input[i]);

   out output[MAX_ELEMS-SLIDER_MAX+1];

   slider_mmma(input,output);

   for(int i=0; i<MAX_ELEMS-SLIDER_MAX+1; i++)
      printf("Max = %d\t Min = %d\t Median = %d\t Average = %f\n",output[i].max, output[i].min,
output[i].median, output[i].average);

   return 0;
```

} **Limitations:**

As we increase the size of the sliding window, program becomes slower and slower.

Question 7:

problem:

Find a given array is sorted or not, and the sorting order.

Approach:

Initially we initialize two flags , increasing and decreasing flag, to non-trivial values.

During the traversing the array when we encounter increasing order is broken, we set it to zero. Similar goes for the decreasing flag. At the end the traversal if we have any one of the flag with

non-trivial value, we can easily conclude the array is sorted. Otherwise the array would be unsorted.

Code:

```
#define IS_EQUAL 0

#define IS_GREATER 2

#define IS_LESSER 1

#define ARR_SIZE 10


int determine_order(int *arr){

   int is_in_order = IS_EQUAL;

   for(int i=1; i<ARR_SIZE; i++){

      if(arr[i] == arr[i-1])

         is_in_order = is_in_order|IS_EQUAL;

      else if(arr[i] > arr[i-1])

         is_in_order = is_in_order|IS_GREATER;

      else is_in_order = is_in_order|IS_LESSER;


   } return is_in_order;

}
```

Question 8:

problem:

Given two sorted array merge them into a single sorted array.

Approach:

We initialize two indexes, one for each of the sorted array, to zero. Now we continue while loop until the two indexes stay in range of the arrays. For each iteration of the loop we actually check which of

the current value of the two array is lesser and we insert that value into a new array and increment the index of that corresponding array.

Code:

```
/* +0 : unsorted
   +2 : each element is equal
   +1 : increasing
   -1 : decreasing
*/
int IsSorted(int a[], int sz)
{
   int i, increasing = 1, decreasing = 1;

   for(i = 0; i < sz - 1; i++)
   {
      if(a[i] > a[i + 1]) increasing = 0;
      if(a[i] < a[i + 1]) decreasing = 0;
   } if(increasing && decreasing) return 2;
   else if(increasing || decreasing) {
      if(increasing) return 1;
      if(decreasing) return -1;
   } else return 0;
} int* merge(int a[], int b[], int sza, int szb)
{
```

```
   int *ma = (int*)malloc(sizeof(int)*(sza + szb));


   int indx = 0, ia = 0, ib = 0;


   int sorder_a = IsSorted(a, sza), sorder_b = IsSorted(b, szb);

   int increasing = 0, decreasing = 0;


   if((sorder_a == 1 && sorder_b == 1) || (sorder_a == 1 && sorder_b == 2) || (sorder_a == 2 &&
sorder_b == 1) || (sorder_a == 2 && sorder_b == 2)) increasing = 1;

   if((sorder_a == 2 && sorder_b == -1) || (sorder_a == -1 && sorder_b == 2) || (sorder_a == -1 &&
sorder_b == -1)) decreasing = 1;


   if(increasing)

   {

      while(indx < sza + szb)

      {

         if(ia >= sza) {ma[indx] = b[ib]; ib++; indx++;}

         else if(ib >= szb) {ma[indx] = a[ia]; ia++; indx++;}

         else{
            if(a[ia] < b[ib]) {ma[indx] = a[ia]; ia++; indx++;}

            else {ma[indx] = b[ib]; ib++; indx++;}

         }

      }

   } else if(decreasing)

   {

      while(indx < sza + szb)
```

```
    {
      if(ia >= sza) {ma[indx] = b[ib]; ib++; indx++;}
      else if(ib >= szb) {ma[indx] = a[ia]; ia++; indx++;}
      else
      {
        if(a[ia] > b[ib]) {ma[indx] = a[ia]; ia++; indx++;}
        else {ma[indx] = b[ib]; ib++; indx++;}
      }
    }
  } else
  {
    printf("Invalid operation!!!\n");
  } return ma;
}
```

Limitations:

This approach for merging of two sorted array into one is the best way because we're just traversing the length of each array only once, and the merging is done in that traversal itself.

Assignment 3

Question 1:

problem:

Implement the following functions of ADT Linked List using singly linked list as a header file:

init_l(cur) – initialise a list **empty_l(head)** – boolean function to return true if list pointed to by head is empty **atend_l(cur)** – boolean function to return true if cur points to the last node in the list **insert_front(target, head)** – insert the node pointed to by target as the first node of the list pointed to by head **insert_after(target, prev)** – insert the node pointed to by target after the node pointed to by prev **delete_front(head)** – delete the first element of the list pointed to by head **delete_after(prev)** – delete the node after the one pointed to by prev

Solution Approach:

We need to create the following functions and add them to a header file so that we can use SLL ADT in other programs. There will be an initialization function which creates and allocates space for a node, a check function to find whether the node is at the end of the list(if the next pointer of the node is null), and other functions to manipulate the before and after pointers and add/remove nodes.

Algorithm:

- **init_l(curr):** uses malloc to allocate memory for the new node of size Node.
- **empty_l(head):**
    - *if(head == null) => list is empty*
- **at_end(head):**
    - *if(the next pointer of head is null) => the node is at the end*
- **insert_front(target, head):**
    - *if(head == null)*
        - *head = target*
    - *else*
        - *target->next = head*
        - *target = head*
- **insert_after(target, prev):**
    - *if prev == null*
    - *target->next = prev*

    - *else*
    - *node* temp = prev->next*
    - *prev->next = target*
    - *target->next = temp*
- **delete_after(prev):**
    - *if prev == null*
    - *abort*
    - *if prev->next == null*
    - *abort*
    - *else*
    - *node* temp = prev->next*
    - *prev = prev->next->next*
    - *delete(temp)*

Question 2:

problem:

Read integers from a file and arrange them in a linked list (a) in the order they are read, (b) in reverse order. Show the lists by printing.

Solution Approach:

For storing the numbers in the same order they are read we just need to store every number at the end of the linked list every time.For storing the numbers in reverse order we need insert the number before head node of the list every time.

Algorithm:

- **Read integers from list Input:**Filename and a bool flag to determine the order readIntegers(char* filename, int order)

begin:

verify the file exists or not open the file in read mode node* forward = init_l() node* reverse = init_l() value = read an integer from the file & store node* curr = createnode(value) if order = 1//denotes forward storing

insert_after(curr,&forward) prev = curr else

if order = -1//denotes reverse storing

insert_front(curr,&reverse) endif endif close the file end

General Discussion:

Everytime we work with dynamic memory allocation we need to take care of the malloc error, memory leakage and memory size such kind of things. We also should check the existence of the file that we are given.

Question 3:

problem:

Implement the following functions in a menu-driven C program using the data structure operation of Singly Linked List in the header file developed in problem 1: a) print a list (i) in the same order, (ii) in the reverse order. b) find the size of a list in number of nodes c) check whether two lists are equal d) search for a key in (i) an unordered list, (ii) an ordered list(Return the node if key found and delete the node from original list) e) append a list at the end of another list. f) delete the nth Node, last node and the first node of a list.

g) check whether a list is ordered
h) merge two sorted lists i) insert a target node in the beginning, before a specified node and at the end of the list (sorted and unsorted). j) remove duplicates from a linked list (sorted and unsorted) k) swap elements of a list pairwise

l) move last element to the front of a list m) delete alternate nodes of a list n) rotate a list o) delete a list. p) reverse a list. q) sort a list.

Solution Approach:

We have to achieve the above functionalities using the header file created for singly linked list.So we need to create separate function in order to achieve that

int return_size(node*);//just to show null error void create_a_list(); void print_a_list(); void print_a_particular_list(int); void print_all_list(); void print_in_reverse_main(); void print_in_reverse(node*); void find_number_of_nodes(); void compare_two_lists(); void delete(); void delete_alternate_nodes(); void search_a_key();

Algorithm:

- **Print a list**
  - **In same order INPUT:** head node of the list which is to be printed

    begin

    node* **create_a_list()** begin

    int user_input = /*take input from user*/ node* head = init_l(user_input)
    if head == NULL

raise error else

    return head end head;

    if head = NULL

raise empty error else

    while head -> next != NULL

    print head->data head = head->next end while endif end
  - **In reverse order Input:** head node of the list that is to be printed

    print_in_reverse(head) begin

    print_in_reverse(head->next) print head end

- **Size of a List Input:** head node of the list whose number of nodes is to be determined

    size_of_a_list(node* head) begin

    count = 0 while head -> next != NULL

count = count + 1 end while count = count + 1 return count end

● **Check Weather the lists are equal or not Input:** head nodes of two lists which are to be compared

compare_two_lists(node* head1, node* head2)

begin

size1 = size_of_a_list(head1) size2 = size_of_a_list(head2) if head1 = 0 and head2 = 0 print equal

else if head1 != head2 print unequal else

for i=0 to i<head1 in steps of 1

if head1->data != head2->data print unequal break else

head1 = head1->next head2 = head-> next endif end for print equal end if

● **Search for a key Input:** head node of the list and the key value to be searched

search_for_a_key(node* head, int key)

begin:

while atend_l(temp)

if head->data = key

node* temp = head while temp != NULL

if temp->next->data = key

print found return temp->next delete_after(temp) else

print key not found endif end while end

● **Append a list at the end of another list Input:** head nodes of two lists append_two_lists(node* head1, node* head2)

begin:

node* temp = head1 while temp != NULL

temp = temp->next end while temp->next = head2

end

● **Delete nth node Input:** The head node and the position of the node in that list delete_nth_node(node* head, int n)

begin:

size = size_of_a_list(head) if n > size print invalid number of node else if size = 0

print list has no element else

for i = 0 to n-1 in steps of 1 do

head = head->next delete_after(head) end for endif end

- **Delete last node Input:** Delete the last node of the given list

delete_last_node(node* head)

begin:

while head->next != NULL

delete_after(head) end while end

- **Delete first node Input:** head node of the list

delete_first_node(node* head)

begin:

delete_front(head) end

- **Check if the list is ordered or not Input:** head node of the list to be checked

begin:

flag = 0 size = size_of_a_list(head) if size = 0 or size = 1 or size = 2

print list is sorted else

if head->data > head->next->data

/*go for descending order checking*/
while head->next != NULL

if head->data < head->next->data

flag = 1 print Unordered break end if end while if flag = 0 print ordered end if

else

if head->data < head->next->data /*go for descending order checking*/ while head->next != NULL

if head->data > head->next->data

flag = 1 print Unordered break end if end while if flag = 0 print ordered end if end if end if end

● **Swap elements of a list Input:** two nodes whose values are to be swapped

swap(node* a,node* b)

begin:

int temp = a->data a->data = b->data b->data = a->data end

● **Sort a list Input:** head node of the list to be sorted and an integer to determine the way to sort(ascending or descending)

sort(node* head,int order)

begin:

size = size_of_a_list(head) if size =0 or size = 1 or size = 2 print already sorted return else

swapped = 0 node* ptr node* lptr = NULL do

swapped = 0 ptr = head while ptr->next != NULL

if order = 1

if ptr->data > ptr->next->data

swap(ptr,ptr->next) swapped = 1 endif elseif

if order = -1

if ptr->data<ptr->next->data

swap(ptr,ptr->next) swapped = 1 endif endif ptr = ptr->next endif lptr = ptr while swapped = 1 endwhile endif end

● **Merge two sorted list Input:** head node for the two sorted list

merge_two_list(node* head1,node* head2)

begin:

node** final if empty_l(head1) = true

*final = head2 return final elseif empty_l(head2) = true

*final = head1 return final else

order = 1 if head1->data > head1-> data

order = -1 //denotes descending order

else

order = 1 //denotes ascending order endif endif *final = head1

append_two_lists(*final,head2) sort(final,order) end

● **Move last element to the front Input:**head node of the list move_to_the_first(node* head)

begin:

node* temp = head while temp->next != NULL

temp = temp->next end while temp->next->next = head temp->next = NULL end

● **Reverse a list Input:** head node of the list to be reversed reverse_a_list(node* head)

begin:

if empty_l(head) = true

print list is empty return else

size = size_of_a_list(head) if size = 1 return// list already reverse

else

node* curr = head node* prev = init_l() while curr != NULL

next = current->next current->next = prev prev = current current = next end while *head = prev endif endif end

General Discussion:

The first question that arises that why should we use linked list in the first place. Well the answer is it has some benefits over arrays. Linked lists can grow in size(theoretically infinitely but practically it's limited by the memory size of the heap memory). Moreover Insertion and deletion in any linked list takes constant time whereas in the array it takes O(n) time.The only problem with linked list is that searching and sorting in linked list takes O(n) and O(n^2) time respectively.In array it is way more faster.

Question 4:

problem:

Write all the above operations of Single Linked List for the implementation using array.

Solution Approach:

We need to achieve all the functionalities of the linked list but using only arrays. For this we need to take a 2D array with two fields namely data field and cursor field.Data field contains the data corresponding to every node. The cursor field somewhat does the job of pointers in a trivial linked list.Every cursor field contains the index of next linked node in that list.We need an another variable named available which always points to the next available node present in the array.

for achieving those functionalities we need to define these functions:

int check_size(); int is_valid(int);//return true if the list is valid int size_of_a_list(int); void new_list(); void display_one_list(); void display_all_list(); void display(); void display_size(); void insert_a_new_data(); void enter_before_head(int,int); void enter_at_a_position(int,int); void enter_at_end(int,int); void delete_something(); void delete_first_element(int); void delete_nth_element(int); void delete_last_element(int); void driver_size_of_a_list(); void compare_two_lists(); void merge_two_lists(); void reverse_print_a_list();

Algorithm:

struct node {

int data; int next; }; struct node s[max];//initialize the array of data int head[head_count];//contains the index of head of the int list_no = 0; int size = max; // contain no of available nodes int available = 0;

- **Create a new list** begin:

    if list_no >= head_count or size <= 0

    print not enough space return else

        head[list_no] = available list_no = list_no + 1 for i = 0 to num in steps of 1 do temp = input() s[available.data] = temp if i!= num-1

available = s[available].next elseif i = num-1

papa = s[available].next s[avialable].next = -1 available = papa endif end for endif end

● **Display a list Input:** index of the head node of the list begin:

index = head[a]//a is the index of head node while s[index].next != -1

print s[index].data index = s[index].next end while if s[index].next == -1

print s[index].data endif end

● **Insert a new data to a list Input:** head index(=a) of the list and the data(=data) to be entered

○ **Insert before head**

begin:

s[available].data = data temp = s[available].next s[availble].next = head[a] head[a]= available available = temp end

○ **Insert at a particular position**

begin:

temp = s[head[a]].next for i=1 to pos-1 in steps of 1 do

temp = s[temp].next end for s[available].data = data store = s[available].next s[temp].next = available s[available].next = temp1 available = store end

○ **insert at end** begin:

temp = s[head[a]].next while s[temp].next != 1

temp = s[temp].next end while

s[available].data = data store = s[availble].next s[available].next = available available = store end

● **Delete a data from the list**

○ **Delete first element**

begin:

s[head[a]].data = '\0'; int store = s[head[a]].next s[head[a]].next = available available = head[a] head[a] = store size = size+1 end

○ **Delete nth element**

begin:

start = head[a] temp = s[start].next for i=1 to n-1 in steps of 1 do temp = s[temp].next end for temp1 = s[temp].next size++ s[temp].next = available

available = temp

for i=1 in steps of 1 do

if s[start].next = temp

s[start].next = temp1 break else

start = s[start].next endif end for end

- **Delete last element**

begin:

```
int temp1 = s[temp].next;//store the last one size++;
s[temp].next = available; available = temp;
```

Question 5:

problem:

Repeat problems 1 and 3 for a circular single linked list, doubly linked list and circular doubly linked list.

Solution Approach:

The approach for doubly linked list is almost the same, except that there's a new node pointing to the previous node which will make traversal easier The approach for circular linked list means, the last pointed (called the tail pointer) will point to the first node and this will also speed up traversal in some cases

Algorithm(Doubly Linked List):

- init_l(curr): *Allocate space for a new node using malloc*
- empty_l(curr): *Check if curr is null or not, if yes then empty.*

● at_end(curr): *Check if the forward pointer is null, if yes, then the node is the end or else not.*

● insert_front(target, head):

target->next = head head->prev = target

● insert_after(target, previous):

node temp = prev->next previous->next = target target->next = temp temp->prev = target target->prev = previous*

● delete_front(head):

- *node* temp = head*
- *head = head->next*
- *head->prev = null*
- *delete(temp)*

● delete_after(previous):

- *node* temp = previous->next*
- *previous->next = previous->next->next*
- *if(previous->next == null)*

  end
- *else*
- *node* nextNode = previous->next*
- *nextNode->prev = previous*
- *delete(temp)*

● Print the list in the same order

- *while(!empty_l(head))*

  print(head) head = head->next

● Print the list in the reverse order

- *while(!at_end(head))*

  head = head->next
- *while(!empty_l(head)*

  print(head) head = head->prev

● Find the size

- *while(!empty_l(head))*

  count++

- Check if two lists are equal
    - *if(empty_l(head1) && empty_l(head2))*

      return true
    - *bool check = true*
        - *else if(size_l(head1) == size_l(head2)) while(!empty(head1))*

          if(head1 != head2)

      check = false return false head1 = head1->next head2 = head2->next else return false

- Search for the node in an unordered/ordered list:
    - *It will take the same time to search in ordered and unordered linked list, because it takes the same time to traverse to any node*
    - *while(!empty_l(head))*

      if(head == toBeSearchedNode)

      node previous = head.prev; delete_after(previous) return true*

    return false
- Append a list at the end of another list
    - *head1, head2 => pointers to the two lists*
    - *while(!at_end(head1))*

  head1 = head1->next head1->next = head2
- Delete the first node, the last node, and the nth node of a list
    - *Deleting the first node*

      delete_front(head)
    - *Deleting the nth node int count = 1; while(count == n || empty_l(head))*

      count++; if(count == n)

      node previous = head->prev delete_after(previous) return*
      if(empty_l(head))

      >> there are less than n nodes
    - *Deleting the last node*

      while(at_end(head))

  head = head->next delete_after(head->prev)
- Check whether the list is ordered
    - *int states(0 implies all equal, 1 implies increasing and -1 implies that its decreasing*

○ *if(size_l(head) == 1)*

return 0 else

find_state(head)

if(head->ele == head->next->ele)

state = 0 else if(head->ele < head->next->ele)

state = 1 else state = -1 head = head->next while(!empty_l(head))

newstate = find_state(head) if(newstate == 0) continue else if(newstate == state) continue else

>>it's not ordered >>return

>>its ordered and print the state end

● Merge two sorted Lists

head3 is the new list. while(!empty_l(head1) && !empty_l(head2)) if(head1->ele < head2->ele)

insert_after(head1, head3) head1->next else if(head2->ele < head2->ele)

insert->after(head2, head3) head2->next else

insert->after(head1, head3) insert->after(head2, head3) head1->next head2->next

● Insert a target node at the beginning, before a specified node and at the end of the list

○ *while(!at_end(head1)*

head1 = head1->next head1->next = target target->prev = head1

○ *head1 is the list node target*

○ *insert_front(target, head) //inserts at the beginning node* ptr is a pointer to the specific node insert_after(target, ptr->prev)*

● Sort a linked list

head1 is a pointer to the head node of the list temph1 = head1 temph2 = head1->next while(!empty_l(head1))

head2 = head1 while(!empty_l(head2->next))

ele1 = head2->ele ele2 = head2->next->ele if(ele1 > ele2)

swap(ele1, ele2) head2 = head2->next head1 = head1->next

● Remove duplicates from unsorted/unsorted list

○ *Sort the list(if unsorted)*

○ *head1 is the list*

while(!empty_l(head1->next))

if(head1->ele == head1->next->ele)

delete_after(head1) head1 = head1->next

- Swap elements of a list pairwise
  - *Node* end_ptr, front_ptr front_prt = head*
  - *while(!at_end(head))*

head = head->next end_ptr = head while(front_ptr != end_ptr || end_ptr->next!=end_ptr)

swap(front_ptr, end_ptr) front_ptr = front_ptr->next end_ptr = end_ptr->prev

- Delete the alternate nodes of a list *delete_front(head) while(!empty_l(head) || !at_end(head))*

delete_after(head) head = head->next if(head->next == null)

end else

head = head->next

- Rotate a list n times
  - *for(int i = 1 ; i <= n ; increment i by 1)*

node endptr, frontptr node* dummy = endptr while(!empty_l(frontptr))*

node temp = frontptr frontptr->ele = dummy->ele dummy->ele = temp->ele frontptr=frontptr->next*

endofwhile

- Delete a list

while(!at_end(head))

head = head->next while(!empty_l(head))

delete_after(head->prev)

- Reverse a list: *swap elements of the list pairwise*

Algorithm(Circular List):

- init_l(curr)
  - *allocate memory for curr, which will be used in the list*
- *empty_l(head)*
  - *if(head==null)*

return true return false

- *at_end(curr, tail)*
    - *if(curr == tail)*

  return true return false
- *insert_front(target, tail)*

  Node temp = tail.prev temp->next = target target->prev = temp target->next = tail tail->prev = target*
- *insert_after(target, tail)*
    - *similar to insert_after of doubly linked list*
- *delete_after(tail)*
    - *Node temp = tail->next*
    - *tail = tail->next*
    - *delete(taili)*
- *print_list(tail)*

  Node temp = tail while(temp != tail)*

  temp = temp->next print(temp) print(temp
- *print_list_reverse(tail)*

  node temp = tail while(temp != tail)*

  print(temp) temp = temp->prev

- *size_l(tail)* *node* temp = tail*

  print(temp) temp = temp->next count++
  while(temp! = tail)

  count++; temp = temp->prev
- *equal(tail1, tail2)*

  if(tail1 == tail2 && tail1 == null)

  return true else if(size_l(tail1) != size_l(taile2)

  return false else

  node n = tail n = n->next while(n != tail)*

  if(equal nodes)

  return true return false
- *search_for_a_key(tail, key) : node* n = tail->next while(n != tail)*

  if(n->ele == key)

return true if(tail->ele == key)

return true return false

- *ordered_or_not(tail):*
  - *int states(0 implies all equal, 1 implies increasing and -1 implies that its decreasing*
  - *if(size_l(tail) == 1)*

return 0 else

find_state(tail)

if(tail->ele == tail->next->ele)

state = 0 else if(tail->ele < tail->next->ele)

state = 1 else state = -1 head = head->next while(!at_end(tail))

newstate = find_state(head) if(newstate == 0) continue else if(newstate == state) continue else

>>it's not ordered >>return >>its ordered and print the state end

Question 6:

problem:

Implement an application to find out the Inverted Index of a set of text files.

Program Approach:

The Program is written using 2 structure 1 for creating a structure for number of file_names from where words are taken and another structure where for each word an indivisual node is created and the word list is expanding downwards whereas the dynamic ling provides to the indivisual word list points to the list of file names where the particular word can be found

Algorithm:

typedef struct node

```
{
 int val;
   struct node *next;
}node;

typedef struct words_lis{
   char word[30];
   struct words_lis*next;
   node * link;
}words_lis;

node*create_n(int val){
   node *n=(node*)malloc(sizeof(node));
   n->next=NULL;
   n->val=val;
   return n;
}
words_lis* create_w(char *c,int v){
   words_lis * w=(words_lis*)malloc(sizeof(words_lis));
   strcpy(w->word,c);
   w->next=NULL;
   w->link=create_n(v);
   return w;}
void process(char * word,words_lis **head,int file_no){
   if(*head==NULL){
      *head=create_w(word,file_no);
      return;
   }
   words_lis *temp=*head;
   while(temp->next!=NULL){
      if(strcmp(temp->word,word)==0){
         node *temp1=temp->link;
         while(temp1->next!=NULL){
            temp1=temp1->next;
         }
         temp1->next=create_n(file_no);
         return;
      }
      temp=temp->next;
   }
   temp->next=create_w(word,file_no);
   return;
}
```

```
void print(words_lis *head){
    while(head!=NULL){
        printf("word: %s  |",head->word);
        node*temp=head->link;
        while(temp!=NULL){
            printf("%d | ",temp->val);
            temp=temp->next;
        }
        printf("\n");
        head=head->next;
    }
}
int file_no=1;
int main(){

    words_lis *head=NULL;
      int ch;
    do{
    char file_name[20];
    scanf("%s",file_name);
    FILE *fp;
    fp=fopen(file_name,"r");
    char c;
    char word[30];
    int count=0;
    while((c=fgetc(fp))!=EOF){


        if(c==' '||c=='\n'){
            // if(count==0){
            //     continue;
            // }
            word[count]='\0';
            printf("%s\n",word);
            process(word,&head,file_no);
            count=0;

        }
        else {
            word[count]=c;
            count++;
        }
```

```
    }
    print(head);
    fclose(fp);
    printf("add another: ");
    file_no++;
    fflush(stdin);
    scanf("%d",&ch);
    }while(ch==1);
}
```

Question 8:

problem:

Write an application for adding, subtracting and multiplying very large numbers using linked lists.

Program Approach:

For multiplying the two lists:

Store the numbers digit by digit in the linked list. And then multiply each digit of the linked list with the remaining nodes of the linked list to get the final result

Algorithm:

*input the two numbers x1, x2 node *head1, *head2 are the two linked lists subroutine: insertNumber(head, x)*

for(each number : x) while(x != 0)

dig = x%10 insert(head, x)

x = x/10 subroutine: insertNumberAtEnd(head, num)

while(!at_end(head))

head = head->next head->ele = head->ele + mult % 10 mult = mult/10 while(mult != 0)

head->next = new Node(mult%10) mult = mult/10 insert(head1, x1) insert(head2, x2)

*node *head3 = final result while(!empty(head1))*

ele1 = head1->ele while(!empty(head2))

*ele2 = head2->ele mult = ele1 * ele2 insertNumberAtEnd(head3, mult)*

For adding two lists:

Add corresponding list values to each other. In case of the carry, add it to the next node.

Algorithm:

x1, x2 be the numbers head1, head2 are the corresponding lists for the two numbers insertNumber(head1, x1) insertNumber(head2, x2)//subroutine calls head3 is the third list head3->ele = 0 while(!empty_l(head1) && !empty_l(head2))

int element = head1->ele + head2->ele head3->ele += element%10 if(element/10 == 0)

head3 = head3->next continue else

head3 = head3->next head3->ele = element/10

head1 = head1->next head2 = head2->next if(!empty_l(head1))

head3->ele += head1->ele%10 if(head1->ele/10 == 0)

head3 = head3->next continue else

head3 = head3->next head3->ele = element/10 if(!empty(head2))

//same as with head1

For subtracting two numbers:

Subtract 9 from all the numbers of the subtractend. Then add the numbers. If you have a carry at the end, then the result is the final list without, the last node. Else, the answer is the negative of the list value

Algorithm:

int x1, x2 node head1, head2 be the two linked list insert(head1, x1) insert(head2, x2) //we are going to do head1 - head2 while(!empty_l(head2))*

head2->ele += 9 head2 = head2->next int size1 = size_l(head1) head3 = add(head1, head2) if(size_l(head3) == size1)

>>answer is negative of the list value else

while(at_end(head->next)) head = head->next delete_after(head) >>answer is the list value

Question 9:

problem:

Given two polygons, find out whether they intersect or not.

Program Approach:

The underlying concept for this problem is to divide a polygon into some number of lines segments and check whether they intersect or not. More specifically we have to find the equation of line for 2 consecutive vertex for each polygon and check whether the intersecting points lies in between the 2 vertices.

Algorithm:

```
typedef struct poly{
   int x,y;
   struct poly* next;
}poly;
poly* create(int x,int y){
   poly* p=(poly* )malloc(sizeof(poly));
   p->x=x;
   p->y=y;
   p->next=NULL;
```

```
    return p;
}
void add_vertex(poly**head,int x,int y){
    if(*head==NULL){
        *head=create(x,y) ;
        return;
    }
    poly* temp=*head;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=create(x,y);
    return;

}
void get_abc(int x1,int y1,int x2,int y2,int *a,int *b,int *c){
*a=(y2-y1);
*b=(x1-x2);
*c=(*a)*x1+(*b)*y1;
return;
}

int check_intersection(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4){
    int A,a,B,b,C,c;
    get_abc(x1,y1,x2,y2,&A,&B,&C);
    get_abc(x3,y3,x4,y4,&a,&b,&c);
    printf("%d %d %d %d %d %d\n",A,a,B,b,C,c);
    if(a*B==A*b){
        printf("parallel lines\n");
        if(c!=C){
            return 0;
        }
        else return 1;

    }
    float int_x=(b*C-B*c)/(A*b-a*B);
    float int_y=(a*C-A*c)/(a*B-A*b);

    if(int_x>=min(x1,x2) && int_x<=max(x1,x2) &&int_x>=min(x3,x4) && int_x<=max(x3,x4)
&&int_y>=min(y1,y2) && int_y<=max(y1,y2)&&int_y>=min(y3,y4) && int_y<=max(y3,y4))
    return 1;
    return 0;
}
```

```
void combination(poly*head1,poly *head2){
   poly*t1=head1;
   poly*t2=head2;
   while(head1->next!=NULL){
      poly *t11=head1;
      head1=head1->next;
      while (head2->next!=NULL)
      { poly *t12=head2;
      head2=head2->next;

if(check_intersection(t11->x,t11->y,head1->x,head1->y,t12->x,t12->y,head2->x,head2->y)==1){
          printf("Polygon intersecting\n");
          return;}

      }

if(check_intersection(t11->x,t11->y,head1->x,head1->y,t2->x,t2->y,head2->x,head2->y)==1){
          printf("Polygon intersecting\n");
          return;
      }
   }

   if(check_intersection(t1->x,t1->y,head1->x,head1->y,t2->x,t2->y,head2->x,head2->y)==1){
          printf("Polygon intersecting\n");
          return;
      }
      printf("non_intersecting_polygon\n");
}
```

Assignment 4

Question 1: **problem:**

Implement Coin Change Problem using Greedy Strategy where coins are of denominations {1p , 5p , 10p, 25p , 50p}. The input to function should be target amount in paise.

Solution Approach:

Keep Selecting the largest denomination which is smaller than equal to target and add it to the result list and decrement target by that amount . Keep doing it until target is zero or there is no such coin smaller than or equal to target . Before returning the list check if target value is reduced to zero . If not print that coin change is not possible.

PseudoCode:

```
CoinChange(Target):
    deno = [1 , 5 , 10 , 25 , 50] res = [] while(Target > 0):
        found = false for(i = deno.length()-1; i >= 0; i--):
            if(deno[i] <= Target): found = true Target -= deno[i] break if(found == false): break

    if(Target == 0) return res
        else print (Coin Change not possible)
```

Limitations:

The Above Method does not work for all denominations. Like

{1 , 5 , 6 , 9} and Target 11.

Question 2:

problem:

Implement the N-queen Problem.

Solution Approach:

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

PseudoCode:

```
isSafe(board , row , col):
    //Check on left side in same row for(i = 0; i < col; i++):
        if(board[row][i] == 1):
    return false //Check UpperDiagonal on left side for(i = row-1 , j = col-1; i >= 0&&j >= 0; i--,j--):
        if(board[i][j] == 1):
    return false //Check LowerDiagonal on left side for(i=row+1,j=col-1; i < N && j >= 0; i++,j--):
        if(board[i][j] == 1):
    return false return true

Nqueen(board[N][N] , col):
    //Base case: All queens placed if(col >= N): return true //Consider this column for keeping a queen //Consider keeping at all possible rows one by one for(i = 0; i < N; i++):
        if(isSafe(borad , i , col) == true): Board[i][col] = 1 if(Nqueen(board , col+1) == true):
                return true
    //If keeping at this row doesn't lead to //solution , backtrack. Board[i][col] = 0 return false
```

Question 3:

problem:

Implement Rat in maze problem for various dimensions of maze.

Solution Approach:

If destination is reached , return true Else ,

a)Mark current cell in solution matrix as 1. b)Move forward in horizontal position and recursively check if

this leads to a solution. c)If the above move does not lead to solution , move down and

Check if this leads to solution. d)If none of the moves leads to solution , mark the current cell as 0 and return false.

PseudoCode:

```
isSafe(maze[N][N] , row , col):
        if(row>=0 && row<N && col >= 0 && col < N):
                if(maze[N][N] == 1):
        return true return false

         RatInMaze(maze[N][N] , row , col , sol[N][N]): if(row == N-1 && col == N-1):
                Sol[row][col] = 1 return true

        if(isSafe(maze , row , col) == true):
                sol[row][col] = 1 if(RatInMaze(maze,row,col+1,sol) == true):
                                return true if(RatInMaze(maze,row+1,col,sol) == true):
return true sol[row][col] = 0 return false
        return false
```

Discussion:

For representing solution for large N , we can use an array of pairs(representing row and column of cell) for storing the path cells.

Question 4:

problem:

Implement various sorting algorithms and draw plots of timing vs number of inputs to ascertain complexity.

1. Insertion Sort

PseudoCode:

```
InsertionSort(Arr[] , N):
    for(i = 1; i < N; i++): Key = Arr[i] j = i-1 while(j >= 0 && Arr[j] > key):
                                Arr[j+1] = Arr[j]; j = j-1 Arr[j+1] = key
```

2. Selection Sort:

PseudoCode:

```
swap(*xp, *yp):
    temp = *xp; *xp = *yp; *yp = temp;

selectionSort(arr[], int n): i, j, min_idx; for (i = 0; i < n-1; i++):
    min_idx = i; for (j = i+1; j < n; j++):
        if (arr[j] < arr[min_idx]):
            min_idx = j;
```

Plot:

3. Bubble Sort:

PseudoCode:

```
void swap(*xp,*yp): temp = *xp; *xp = *yp; *yp = temp;

bubbleSort(int arr[], int n):
    i, j; for (i = 0; i < n-1; i++):
        swapped = false; for (j = 0; j < n-i-1; j++):
            if (arr[j] > arr[j+1]) swap(&arr[j], &arr[j+1]);
swap(&arr[min_idx], &arr[i]);
```

Plot:

4. Merge Sort:

PseudoCode:

merge(arr[], l, m, r):

i, j, k; n1 = m - l + 1; n2 = r - m; /* create temp arr*/ L[n1], R[n2]; for (i = 0; i < n1; i++):

L[i] = arr[l + i]; for (j = 0; j < n2; j++):

swapped = true; if (swapped == false):

Break;

Plot:

R[j] = arr[m + 1+ j]; /* Merge the temp arrays back into arr[l..r]*/ i = 0; // Initial index of first subarray j = 0; // Initial index of second subarray k = l; // Initial index of merged subarray while (i < n1 && j < n2):

if (L[i] <= R[j]):

arr[k] = L[i]; i++; else:

arr[k] = R[j]; j++; k++; /* Copy the remaining elements of L[], if there are any */ while (i < n1)

arr[k] = L[i]; i++; k++; /* Copy the remaining elements of R[], if there are any */ while (j < n2)

arr[k] = R[j]; j++; k++;

mergeSort(arr[], l, r):

if (l < r): m = l+(r-l)/2;

mergeSort(arr, l, m); mergeSort(arr, m+1, r); merge(arr, l, m, r);

Comparative Chart(sorted,reverse sorted,random)

Elements	Bubble sort	Selection Sort	Insertion Sort	Merge sort	Quick sort	HeapSort
10000	0.24,0.52,0.5	0.1,0.25,0.1	$6*10^{-5}$,0.25,0.1	10^{-3},10^{-3},10^{-3}	0.45,0.38,$3*10^{-3}$	$3*10^{-3}$,$3*10^{-3}$,$3*10^{-3}$
20000	0.94,1.83,2.1	0.41,0.88,0.36	10^{-4},0.9,0.45	$2*10^{-3}$,$2*10^{-3}$,$3*10^{-3}$	1.71,1.24,0.001	$7*10^{-3}$,$6*10^{-3}$,0.008
30000	1.64,4.03,4.8	0.76,1.76,0.7	$2*10^{-4}$,1.87,0.9	$3*10^{-3}$,$5*10^{-3}$,$3*10^{-2}$	4.3,2.77,0.01	0.01,0.014,0.01
40000	3,7,8.3	1.37,3.1,1.36	$3*10^{-4}$,3.3,1.7	$7*10^{-3}$,$9*10^{-3}$,$4*10^{-3}$	6.86,4.9,0.02	0.01,0.01,0.02

				,		
60000	6.85,15,20	3,6.9,3.0	$3*10^{-4}$,7.7,3.8	0.01,0.01,0.006	15,11,0.05	0.03,0.03,0.03
80000	11.6,27.5,34	5.21,12,5.4	$5*10^{-4}$,13,6.8	0.01,0.01,0.01	27,20,0.09	0.04,0.04,0.04
100000	18,44.5,55.1	8.76,17,8.5	$8*10^{-4}$,18,10	0.02,0.01,0.01	43,26,0.129	0.05,0.04,0.06