# DATA STRUCTURE AND ALGORITHM ASSIGNMENTS

JADAVPUR UNIVERSITY

Name: ARKA DAS
Roll: 002210503046
MCA 1st Year 2nd Semester
Session: 2022- 2024

# *Assignment: Set - 2*

# Set-2

## Question – 2

### Problem Statement:

Write a menu-driven program representing a polynomial as a data structure using a singly linked list and write functions to add, subtract and multiply two polynomials.

### Source Code:

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

typedef struct node
{
      int coeff;
      int exp;
      struct node *next;
}node;

node* getnode(void);
node* create(node *);
void display(node *);
node* add_sub(node* , node* , int );    //0 = +, 1 = - ;
node* multiply(node* , node* );

int main()
{
    node* first_List = NULL;
    node* second_List = NULL;
    node* res = NULL;
    int choice;

    printf("\nCreating first polynomial:\n");
    first_List = create(first_List);
    display(first_List);

    printf("\nCreating second polynomial:\n");
    second_List = create(second_List);
    display(second_List);

    while(1) {
        printf("1 -> add polynomial_1 with polynomial_2\n");
        printf("2 -> add polynomial_1 with polynomial_2\n");
        printf("3 -> add polynomial_1 with polynomial_2\n");
        printf("4 -> Exit\n");
        scanf("%d", &choice);
        switch(choice) {
        case 1:
            res = add_sub(first_List, second_List, 0);
            display(res);
            break;
        case 2:
            res = add_sub(first_List, second_List, 1);
            display(res);
```

```
            break;
        case 3:
            res = multiply(first_List, second_List);
            display(res);
            break;
        case 4:
            return 0;
        default:
            printf("\nInvalid option given\n");
        }
    }

}

node* multiply(node* list_1, node* list_2)
{
    node* res = NULL;   //product of ploys
    node* resLast = NULL;
    node* currProd = NULL;
    node* currProdLast = NULL;

    node* l1 = list_1;
    while(l1 != NULL) {
        currProd = NULL;
        currProdLast = NULL;

        node* l2 = list_2;
        while(l2 != NULL) {
            node* t = getnode();
            t->coeff = l1->coeff * l2->coeff;
            t->exp = l1->exp + l2->exp;
            if(currProd == NULL) {
                currProd = t;
                currProdLast = currProd;
            }
            else {
                currProdLast->next = t;
                currProdLast = currProdLast->next;
            }
            l2 = l2->next;
        }
        res = add_sub(res, currProd, 0);
        l1 = l1->next;
    }

    return res;
}

node* add_sub(node* list_1, node* list_2, int mode)
{
    if(list_1 == NULL && list_2 == NULL)
        return NULL;
    if(list_1 == NULL)
        return list_2;
    if(list_2 == NULL)
        return list_1;
```

```
node* res = NULL;   //added polynomial
node* resLast = NULL;
while((list_1 != NULL && list_2 != NULL) && (list_1->exp > list_2->exp)) {
    node* t = getnode();
    t->coeff = list_1->coeff;
    t->exp = list_1->exp;
    if(res == NULL)  {
        res = t;
        resLast = res;
    }
    else {
        resLast->next = t;
        resLast = resLast->next;
    }
    list_1 = list_1->next;
}

while((list_1 != NULL && list_2 != NULL) && (list_2->exp > list_1->exp)) {
    node* t = getnode();
    if(mode == 0)
        t->coeff = list_2->coeff;
    else if(mode == 1)
        t->coeff = -list_2->coeff;
    t->exp = list_2->exp;
    if(res == NULL)  {
        res = t;
        resLast = res;
    }
    else {
        resLast->next = t;
        resLast = resLast->next;
    }
    list_2 = list_2->next;
}

while(list_1 != NULL && list_2 != NULL) {
    node* t = getnode();
    if(mode == 0)
        t->coeff = list_1->coeff + list_2->coeff;
    else if(mode == 1)
        t->coeff = list_1->coeff - list_2->coeff;
    t->exp = list_1->exp;
    if(res == NULL)  {
        res = t;
        resLast = res;
    }
    else {
        resLast->next = t;
        resLast = resLast->next;
    }
    list_1 = list_1->next;
    list_2 = list_2->next;
}

while(list_1 != NULL) {
    node* t = getnode();
    t->coeff = list_1->coeff;
    t->exp = list_1->exp;
```

```
        resLast->next = t;
        resLast = resLast->next;
        list_1 = list_1->next;
    }

    while(list_2 != NULL) {
        node* t = getnode();
        t->coeff = list_2->coeff;
        t->exp = list_2->exp;
        resLast->next = t;
        resLast = resLast->next;
        list_2 = list_2->next;
    }

    return res;
}

node *getnode()
{
      node *t;
      t=(node*)malloc(sizeof(node));
    t->coeff = 0;
    t->exp = 0;
      t->next=NULL;
      return t;
}

void display(node *head)
{
      node *t;
      if(head == NULL)
            printf("\npolynomial is empty");
      else
      {
            t = head;
            printf("\nThe polynomial list is-> ");
            while(t != NULL)
            {
                  if(t->coeff == 0) {
                        t = t->next;
                        continue;
                  }
                  printf("%dx^%d",t->coeff, t->exp);
                  t = t->next;
             if(t != NULL)
                 printf(" + ");
            }
        printf("\n");
      }
}

node* create(node *head)
{
    int highestDeg, x;
    printf("\nEnter the degree of the polynomial: ");
    scanf("%d", &highestDeg);

    while (highestDeg > -1) {
```

```
        printf("\nEnter coefficient for term with degree %d: ", highestDeg);
        node* t = getnode();
        scanf("%d", &x);

        t->coeff = x;
        t->exp = highestDeg;
        if(head == NULL)
            head = t;
        else {
            node* temp = head;
            while(temp->next!=NULL)
                temp = temp->next;
            temp->next = t;
        }
        highestDeg--;
    }
      return head;
}
```

## Output:

```
Creating first polynomial:


Enter the degree of the polynomial: 3

Enter coefficient for term with degree 3: 3

Enter coefficient for term with degree 2: 2

Enter coefficient for term with degree 1: 1

Enter coefficient for term with degree 0: 5

The polynomial list is-> 3x^3 + 2x^2 + 1x^1 + 5x^0


Creating second polynomial:

Enter the degree of the polynomial: 2

Enter coefficient for term with degree 2: 4

Enter coefficient for term with degree 1: 2

Enter coefficient for term with degree 0: 4

The polynomial list is-> 4x^2 + 2x^1 + 4x^0


1 -> add polynomial_1 with polynomial_2

2 -> subtract polynomial_1 with polynomial_2

3 -> multiply polynomial_1 with polynomial_2
```

```
4 -> Exit
1
The polynomial list is-> 3x^3 + 6x^2 + 3x^1 + 9x^0

1 -> add polynomial_1 with polynomial_2
2 -> subtract polynomial_1 with polynomial_2
3 -> multiply polynomial_1 with polynomial_2
4 -> Exit
2
The polynomial list is-> 3x^3 + -2x^2 + -1x^1 + 1x^0

1 -> add polynomial_1 with polynomial_2
2 -> subtract polynomial_1 with polynomial_2
3 -> multiply polynomial_1 with polynomial_2
4 -> Exit
3
The polynomial list is-> 12x^5 + 14x^4 + 20x^3 + 30x^2 + 14x^1 + 20x^0
```

# Question – 3

## Problem Statement:

Implement Doubly Linked List for the following operations –

I. Create a linked list.

II. Print the content of the list.

III. Insert an element at the front of the list

IV. Insert an element at the end of the list

V. Insert a node after the kth node.

VI. Insert a node after the node (first from the start) containing a given value.

VII. Insert a node before the kth node.

VIII. Insert a node before the node (first from the start) containing a given value.

IX. Delete the first node.

X. Delete the last node.

XI. Delete a node after the kth node.

XII. Delete a node before the kth node.

XIII. Delete the kth node.

XIV. Delete the node (first from the start) containing a specified value.

XV. Find the reverse of a list (not just printing in reverse)

## Source Code:

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

typedef struct node {
      int data;
    struct node *prev;
      struct node *next;
} node;

node* getnode(void);
node* create(node* );
void display(node* );
node* insbeg(node* , int );
node* insend(node* , int );
node* insAfterK(node* , int , int );
node* insAfterNode(node* , int , int );
node* insBeforeK(node* , int , int );
node* insBeforeNode(node* , int , int );

node* delbeg(node* );
node* delend(node* );
node* delAfterK(node* , int );
node* delBeforeK(node* , int );
```

```
node* delKNode(node* , int );
node* delValue(node* , int );
node* reverse(node* );

void getInput(int* , int* , int* , int , int , int );

int main()
{
    node* head = NULL;
    int s, val, t_val, k;
      head = create(head);

      /*
            2 -> display, 3 -> ins front, 4 -> ins end
            5 -> ins after k'th, 6 -> ins after node,
            7 -> ins before k'th, 8 -> ins before node,

            9 -> del first, 10 -> del last
            11 -> del after k'th, 12 -> del before k'th
            13 -> del k'th, 14 -> del value

            15 -> reverse
      */

      while(1)
      {
            // printf("2 -> display \n3 -> ins front \n4 -> ins end ");
            // printf("\n5 -> ins after k'th \n6 -> ins after node");
            // printf("\n7 -> ins before k'th \n8 -> ins before node");
            // printf("\n9 -> del first \n10 -> del last");
            // printf("\n11 -> del after k'th \n12 -> del before k'th");
            // printf("\n13 -> del k'th \n14 -> del value \n15 -> reverse");
            printf("\nEnter choice: ");
            scanf("%d",&s);
            switch(s)
            {
                  case 2:
                        display(head);
                        break;
                  case 3:
                        getInput(&val, &k, &t_val, 1, 0, 0);
                        head = insbeg(head, val);
                        break;
                  case 4:
                        getInput(&val, &k, &t_val, 1, 0, 0);
                        head = insend(head, val);
                        break;
                  case 5:
                        getInput(&val, &k, &t_val, 1, 1, 0);
                        head = insAfterK(head, k, val);
                        break;
                  case 6:
                        getInput(&val, &k, &t_val, 1, 0, 1);
                        head = insAfterNode(head, t_val, val);
                        break;
                  case 7:
                        getInput(&val, &k, &t_val, 1, 1, 0);
                        head = insBeforeK(head, k, val);
```

```
                        break;
                  case 8:
                        getInput(&val, &k, &t_val, 1, 0, 1);
                        head = insBeforeNode(head, t_val, val);
                        break;
                  case 9:
                        head = delbeg(head);
                        break;
                  case 10:
                        head = delend(head);
                        break;
                  case 11:
                        getInput(&val, &k, &t_val, 0, 1, 0);
                        head = delAfterK(head, k);
                        break;
                  case 12:
                        getInput(&val, &k, &t_val, 0, 1, 0);
                        head = delBeforeK(head, k);
                        break;
                  case 13:
                        getInput(&val, &k, &t_val, 0, 1, 0);
                        head = delKNode(head, k);
                        break;
                  case 14:
                        getInput(&val, &k, &t_val, 0, 0, 1);
                        head = delValue(head, t_val);
                        break;
                  case 15:
                        head = reverse(head);
                        break;
                  default:
                        printf("\nWrong choice !!!!");
            }
      }

}

node* getnode()
{
    node *t;
      t=(node*)malloc(sizeof(node));
    t->prev = NULL;
      t->next = NULL;
      return t;
}

node* create(node* head)
{
      node* t = getnode();
      printf("\nEnter first node information: ");
      scanf("%d", &t->data);
      head = t;
      return head;
}

void display(node* head)
{
      node *t;
```

```
      if(head == NULL) {
            printf("\nList is empty");
        return;
    }
      t = head;
      printf("\nThe linked list is-> ");
      while(t != NULL) {
            printf("%d ",t->data);
            t = t->next;
      }
}

node* insbeg(node* head, int val)
{
      node *t;
      t = getnode();
      t->data = val;
      t->next = head;
    if(head != NULL)
        head->prev = t;
      head = t;
      return head;
}

node* insend(node* head, int val)
{
      node *t, *t1;
      t=getnode();
      t->data = val;
    if(head == NULL) {
        head = t;
        return head;
    }
      t1=head;
      while(t1->next!=NULL)
            t1 = t1->next;
      t1->next = t;
    t->prev = t1;
      return head;
}

node* insAfterK(node* head, int k, int val)
{
      node *t, *t1;
      if(head == NULL) {
            printf("\nList is empty");
            return head;
      }
      t = head;
    int i = 1;
    while(i < k && t->next != NULL) {
        t = t->next;
        i++;
    }
    if( (i != k && t->next == NULL) || k <= 0) {
        printf("\nInvalid index given");
        return head;
    }
```

```
      t1=getnode();
    t1->data = val;
    t1->next = t->next;
    t->next = t1;
    t1->prev = t;
    if(t1->next != NULL)
        t1->next->prev = t1;
      return head;
}

node* insAfterNode(node* head, int target_val, int val)
{
      node *t, *t1;
      if(head == NULL) {
            printf("\nList is empty");
            return head;
      }
      t = head;
    while(t->data != target_val && t->next != NULL)
        t = t->next;
    if(t->data != target_val && t->next == NULL) {
        printf("\nInvalid value given");
        return head;
    }

      t1=getnode();
    t1->data = val;
    t1->next = t->next;
    t->next = t1;
    t1->prev = t;
    if(t1->next != NULL)
        t1->next->prev = t1;
      return head;
}

node* insBeforeK(node* head, int k, int val)
{
      if(head == NULL) {
            printf("\nList is empty");
            return head;
      }
    if(k <= 0) {
        printf("\nInvalid location given");
            return head;
    }
    if(k == 1) {
        head = insbeg(head, val);
        return head;
    }

    head = insAfterK(head, k-1, val);
      return head;
}

node* insBeforeNode(node* head, int target_val, int val)
{
      node *t, *t1;
```

```
      if(head == NULL) {
            printf("\nList is empty");
            head = create(head);
            return head;
      }
    if(head->data == target_val) {
        head = insbeg(head, val);
        return head;
    }

      t = head;
    while(t->next != NULL && t->next->data != target_val)
        t = t->next;
    if(t->next == NULL) {
        printf("\nInvalid value given");
        return head;
    }

    t1=getnode();
    t1->data = val;
    t1->next = t->next;
    t->next = t1;
    t1->prev = t;
    if(t1->next != NULL)
        t1->next->prev = t1;
      return head;
}

node* delbeg(node* head)
{
      if(head == NULL) {
            printf("\nList is empty");
            return head;
      }
      node *t;
      t = head;
      head = head->next;
      if(head != NULL)
            head->prev = NULL;
      printf("\nDeleted value %d",t->data);
      free(t);
      return head;
}

node* delend(node* head)
{
      if(head == NULL) {
            printf("\nList is empty");
            return head;
      }

      node *t, *newLast;
      t = head;
      if(head->next == NULL) {
            printf("\nDeleted value %d",head->data);
            head = NULL;
            free(t);
            return head;
```

```
      }

      while(t->next != NULL)
            t = t->next;
      newLast = t->prev;
      newLast->next = NULL;
      printf("\nDeleted value %d",t->data);
      free(t);
      return head;
}

node* delAfterK(node* head, int k)
{
      node *t, *t1;
      if(head == NULL) {
            printf("\nList is empty");
            return head;
      }
      t = head;
    int i = 1;
    while(i < k && t->next != NULL) {
        t = t->next;
        i++;
    }
    if((t->next == NULL) || k <= 0) {
        printf("\nInvalid index given");
        return head;
    }
      //now delete after t
      t1 = t->next;
      t->next = t->next->next;
      if(t->next != NULL)
            t->next->prev = t;
      printf("\nDeleted value %d",t1->data);
      free(t1);
      return head;
}

node* delBeforeK(node* head, int k)
{
      if(head == NULL) {
            printf("\nList is empty");
            return head;
      }
    if(k <= 1) {
        printf("\nInvalid location given");
            return head;
    }
      if(k == 2) {
            head = delbeg(head);
            return head;
      }

    head = delAfterK(head, k-2);
      return head;
}

node* delKNode(node* head, int k)
```

```
{
	if(head == NULL) {
		printf("\nList is empty");
		return head;
	}
	if(k <= 0) {
		printf("\nInvalid location given");
		return head;
	}
	if(k == 1) {
		head = delbeg(head);
		return head;
	}

	head = delAfterK(head, k-1);
	return head;
}

node* delValue(node* head, int val)
{
	if(head == NULL) {
		printf("\nList is empty");
		return head;
	}

	node *t, *t1;
	t = head;
	if(head->data == val) {
		head = delbeg(head);
		return head;
	}

	while(t->next != NULL && t->next->data != val)
			t = t->next;
	if(t->next == NULL) {
		printf("\nValue not present");
		return head;
	}

	t1 = t->next;
	t->next = t->next->next;
	if(t->next != NULL)
		t->next->prev = t;
	printf("\nDeleted value %d",t1->data);
	free(t1);

	return head;
}

void getInput(int* val, int* index, int* t_val, int f1, int f2, int f3)
{
	int v, i, target;
	if(f1) {
		printf("\nEnter new node data: ");
		scanf("%d", &v);
		*val = v;
	}
	if(f2) {
```

```
            printf("\nEnter K'th index: ");
            scanf("%d", &i);
            *index = i;
      }
      if(f3) {
            printf("\nEnter other node data: ");
            scanf("%d", &target);
            *t_val = target;
      }
}
node* reverse(node* head)
{
      node *curr = head, *t = NULL;
      while(curr != NULL) {
            t = curr->prev;
            curr->prev = curr->next;
            curr->next = t;
            curr = curr->prev;
      }
      if(t != NULL)
            head = t->prev;
      return head;
}
```

## Output:

```
Enter first node information: 10

================================================

2 -> display

3 -> ins front

4 -> ins end

5 -> ins after k'th

6 -> ins after node

7 -> ins before k'th

8 -> ins before node

9 -> del first

10 -> del last

11 -> del after k'th

12 -> del before k'th

13 -> del k'th

14 -> del value

15 -> reverse

============================================================

Enter choice: 3
```

```
Enter new node data: 20
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 4
Enter new node data: 30
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
```

```
15 -> reverse
============================================================
Enter choice: 2
The linked list is-> 20 10 30
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 7
Enter new node data: 40
Enter K'th index: 1
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
```

```
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 2
The linked list is-> 40 20 10 30
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 13
Enter K'th index: 9
Invalid index given
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
```

```
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 13
Enter K'th index: 2
Deleted value 20
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 2
The linked list is-> 40 10 30
================================================
2 -> display
```

```
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 15
================================================
2 -> display
3 -> ins front
4 -> ins end
5 -> ins after k'th
6 -> ins after node
7 -> ins before k'th
8 -> ins before node
9 -> del first
10 -> del last
11 -> del after k'th
12 -> del before k'th
13 -> del k'th
14 -> del value
15 -> reverse
============================================================
Enter choice: 2
The linked list is-> 30 10 40
```

# *Assignment: Set - 3*

# Set-3

## Question – 7

### Problem Statement:

Write a program to evaluate postfix expression using a stack.

### Source Code:

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include <stdlib.h>

typedef struct node{
      int data;
      struct node *next;
}node;
node* stackTop = NULL;

node* getnode(void);
void traverse(void);
void push(int );
int pop(void);
int isEmpty(void);
int isOperator(char );
int getNextData(char* ,int* );

int main() {
    char postfix[50] = {" "};
    printf("\nEnter postfix expression\n: ");
    fgets(postfix, 50, stdin);

    int i = 0;

    while(i < strlen(postfix)) {
        char symbol = postfix[i];
        if(isOperator(symbol)) {
            int m = pop();
            int n = pop();
            switch (symbol)
            {
            case '+':
                push(n + m);
                break;
            case '-':
                push(n - m);
                break;
            case '*':
                push(n * m);
                break;
            case '/':
                push(n / m);
                break;
```

```
            case '^':
                push(pow(n , m));
                break;
            default:
                break;
            }
            i += 2;
        }
        else {
            int data = getNextData(postfix, &i);
            //printf("\n dd = %d", data);
            push(data);
        }
    }

    printf("\nEvaluated Value = %d\n", stackTop->data);

}

int getNextData(char* postfix, int* index) {
    char word[10] = {" "};
    int i = 0;
    while(postfix[*index] != '\0') {
        if(postfix[*index] == ' ') {
            *index = *index + 1;
            break;
        }
        word[i++] = postfix[*index];
        *index = *index + 1;
    }
    word[i] = '\0';
    return atoi(word);
}

int isOperator(char c) {
    if(c == 43 || c == 42 || c == 45 || c == 47 || c == 94) return 1;
    else return 0;
}

node* getnode() {
    node* temp = (node*)malloc(sizeof(node));
    temp->next = NULL;
    return temp;
}

int isEmpty() {
    if(stackTop == NULL) return 1;
    else return 0;
}

void push(int val) {
      node* t;
      t = getnode();
      t->data = val;
      t->next = stackTop;
      stackTop = t;
}
```

```
int pop() {
    char c = '#';
    if(stackTop == NULL) {
            printf("\nStack Underflow\n");
        return c;
    }
    node* t = stackTop;
    c = t->data;
    stackTop = stackTop->next;
    free(t);
    return c;
}


void traverse()
{
      node *t;
      if(stackTop == NULL)
            printf("\nList is empty\n");
      else {
            t = stackTop;
            printf("\nThe stack is-> ");
            while(t != NULL) {
                  printf("%c ",t->data);
                  t = t->next;
            }
      }
}
```

## Output:

```
Enter postfix expression

: 4 3 2 ^ + 1 8 * 2 2 + / - 2 -

Evaluated Value = 9


Enter postfix expression

: 2 3 ^ 1 - 4 2 / 6 * + 3 1 + 2 / -

Evaluated Value = 17


Enter postfix expression

: 10 22 + 8 / 6 * 5 +

Evaluated Value = 29
```

## Question – 8

### Problem Statement:

Write a program to check balanced brackets of an expression using stack.

### Source Code:

```
#include<stdio.h>
#include <string.h>
#include<malloc.h>

typedef struct node{
      char data;
      struct node *next;
}node;
node* stackTop = NULL;

node* getnode(void);
char peek(void);
void push(char );
void pop(void);
int isEmpty(void);

int main()
{
    char expression[30] = {" "};
    printf("\nEnter expression of brackets\n: ");
    fgets(expression, 30, stdin);

    int f = 0;
    for(int i=0; i<strlen(expression); i++) {
        char symb = expression[i];
        if(symb == '(' || symb == '{' || symb == '[')
            push(symb);

        else if (symb == ')' && isEmpty() == 0) {
            if(peek() == '(') pop();
            else { f = 1; break; }
        }
        else if (symb == '}' && isEmpty() == 0) {
            if(peek() == '{') pop();
            else { f = 1; break; }
        }
        else if (symb == ']' && isEmpty() == 0) {
            if(peek() == '[') pop();
            else { f = 1; break; }
        }
    }

    if(isEmpty() != 1)
        f = 1;

    if(f == 1) printf("\nRxpression is NOT balanced");
    else printf("\nExpression is balanced\n");
    return 0;
}
```

```
char peek() {
    return stackTop->data;
}

void push(char val) {
      node* t;
      t = getnode();
      t->data = val;
      t->next = stackTop;
      stackTop = t;
}

void pop() {
      if(stackTop == NULL)
            return;
      node *t = stackTop;
      stackTop = stackTop->next;
      free(t);
}

int isEmpty(void){
      if(stackTop == NULL) return 1;
    else return 0;
}

node *getnode() {
      node *t;
      t = (node*)malloc(sizeof(node));
      t->next = NULL;
      return t;
}
```

## Output:

```
Enter expression of brackets

: ([{}()]{}())

Expression is balanced


Enter expression of brackets

: [{}()[({})][]()]

Expression is balanced


Enter expression of brackets

: [{}()[({})[]()]()

Expression is NOT balanced
```

## Question – 15

### Problem Statement:

Write a program for dynamic implementation (using a link list) of a queue.

### Source Code:

```
#include <stdio.h>
#include <malloc.h>
#include <limits.h>

typedef struct node{
      int data;
      struct node *next;
}node;

typedef struct Queue{
      struct node *front;
      struct node *rear;
}Queue;

node* getnode(void);
Queue* create(Queue* );
Queue* insert(Queue* , int );
int delete(Queue* );
void display(Queue* );


int main()
{
    Queue* queue;
    int choice, data, x;
    queue = create(queue);

      while(1) {
            printf("\n1 -> insert, ");
            printf("2 -> delete, ");
            printf("3 -> display, ");
            printf("4 -> Exit, ");
            printf("\nEnter choice: ");
            scanf("%d", &choice);

            switch(choice) {
            case 1:
                  printf("\nEnter data to insert: ");
                  scanf("%d", &data);
                  queue = insert(queue, data);
                  break;
            case 2:
                  x = delete(queue);
                  if(x != INT_MIN)
                        printf("\nPopped data is %d", x);
                  break;
            case 3:
                  display(queue);
                  break;
```

```
            case 4:
                  return 0;
            default:
                  printf("\nWrong choice");
            }
      }

      return 0;

}

Queue* insert(Queue* queue, int val) {
    node* t = getnode();
    if(t == NULL) {
        printf("\nQueue overflow");
        return queue;
    }
    t->data = val;
    if(queue->rear == NULL) {
        queue->front = t;
        queue->rear = t;
        return queue;
    }
    queue->rear->next = t;
    queue->rear = t;
    return queue;
}

int delete(Queue* queue) {
    if(queue->front == NULL) {
        printf("Queue underflow\n");
        return INT_MIN;
    }
    int data = queue->front->data;
    queue->front = queue->front->next;
    if(queue->front == NULL)
        queue->rear = NULL;
    return data;
}

void display(Queue* queue) {
    if(queue->front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    printf("\n");
    node* t = queue->front;
    while(t != queue->rear->next) {
        printf("%d, ", t->data);
        t = t->next;
    }
}

Queue* create(Queue* queue) {
    queue = (Queue*)malloc(sizeof(Queue));
    queue->front = NULL;
    queue->rear = NULL;
    return queue;
```

```
}

node *getnode()
{
      node *t;
      t = (node*)malloc(sizeof(node));
      t->next = NULL;
      return t;
}
```

## Output:

```
1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 3

Queue is empty


1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 1

Enter data to insert: 10


1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 1

Enter data to insert: 20


1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 1

Enter data to insert: 30


1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 3

10, 20, 30,


1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 2

Popped data is 10
```

```
1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
20, 30,

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 1
Enter data to insert: 40

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 1
Enter data to insert: 50

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
20, 30, 40, 50,

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 2
Popped data is 20

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
30, 40, 50,
```

## Question – 16

### Problem Statement:

Implement a circular queue using an array.

### Source Code:

```
#include <stdio.h>
#include <limits.h>
#define MAX 5

int queue[MAX] = {0};
int front = -1;
int rear = -1;

void display(void);
void insert(int );
int delete(void);

int main()
{
	int choice, data, x;
	while(1) {
		printf("\n1 -> insert, ");
		printf("2 -> delete, ");
		printf("3 -> display, ");
		printf("4 -> Exit, ");
		printf("\nEnter choice: ");
		scanf("%d", &choice);

		switch(choice) {
		case 1:
			printf("\nEnter data to insert: ");
			scanf("%d", &data);
			insert(data);
			break;
		case 2:
			x = delete();
			if(x != INT_MAX)
				printf("\nPopped data is %d", x);
			break;
		case 3:
			display();
			break;
		case 4:
			return 0;
		default:
			printf("\nWrong choice");
		}

	}

	return 0;
}

void display() {
```

```
      // if(rear == -1) {
      //    printf("\nQueue is empty");
      //    return;
      // }
      printf("\nQueue contents are:\n");
      // for(int i=front;i<=rear;i++) {
      //    printf("%d, ", queue[i]);
      // }
    for(int i=0;i<MAX;i++) {
            printf("%d, ", queue[i]);
      }
      printf("\n");
}

int delete() {
      if(front == -1) {
            printf("\nUnderflow");
            return INT_MAX;
      }

      int d = queue[front];
    queue[front] = 0;
      if(front == rear) {
            front = rear = -1;
            return d;
      }
      else
            front = (front + 1) % MAX;
      return d;
}

void insert(int data) {
      if((rear + 1) % MAX == front) {
            printf("\nOverflow\n");
            return;
      }
      if(front == -1 && rear == -1)
            front = rear = 0;
      else
            rear = (rear + 1) % MAX;
      queue[rear] = data;
}
```

## Output:

```
1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 1

Enter data to insert: 10


1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,

Enter choice: 1

Enter data to insert: 20
```

```
1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 1
Enter data to insert: 30

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
Queue contents are:
10, 20, 30, 0, 0,

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 2
Popped data is 10

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
Queue contents are:
0, 20, 30, 0, 0,

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 1
Enter data to insert: 40

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 1
Enter data to insert: 50

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 2
Popped data is 20

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
```

```
Queue contents are:
0, 0, 30, 40, 50,

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 2
Popped data is 30

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
Queue contents are:
0, 0, 0, 40, 50,

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 1
Enter data to insert: 60

1 -> insert, 2 -> delete, 3 -> display, 4 -> Exit,
Enter choice: 3
Queue contents are:
60, 0, 0, 40, 50,
```

# *Assignment: Set - 4*

# Set-4

## Question – 1

### Problem Statement:

Write a menu-driven program for a binary tree using linked representation to

(a)Create (b) Preorder traversal (c) Inorder traversal (d) Postorder traversal

### Source Code:

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

typedef struct node {
    int data;
    struct node *left;
    struct node *right;
}node;

node* getnode(void);
void createTree(node** root);
node** search(node** root, int value);
void inorder(node* root);
void preorder(node* root);
void postorder(node* root);

int main()
{
    node* binaryTree = NULL;
    int choice = 0;
    do {
            printf("\n0 -> exit\n1 -> create\n2 -> preorder\n3 -> inorder traversal\n4
-> postorder\nEnter your choice\n");
            scanf("%d",&choice);
            switch(choice)
            {
                  case 0:
                        return 0;
                        break;
                  case 1:
                        printf("Creating Binary tree\n");
                  createTree(&binaryTree);
                        break;
                  case 2:
                        if(binaryTree == NULL) { printf("Nothing in tree\n");
continue; }
                  preorder(binaryTree);
                        break;
                   case 3:
                        if(binaryTree == NULL) { printf("Nothing in tree\n");
continue; }
                  inorder(binaryTree);
                        break;
                   case 4:
```

```
                        if(binaryTree == NULL) { printf("Nothing in tree\n");
continue; }
                postorder(binaryTree);
                        break;
                 default:
                        printf("Enter valid option between 0 to 4\n");
                        break;
            }
      } while(choice!=0);
}

void createTree(node** root) {
    int d = 0, choice = 0;
    node* t = getnode();
    printf("\nEnter root node data: ");
    scanf("%d", &d);
    t->data = d;
    *root = t;
    do {
        node** x;
        printf("1->add left child\n2->add right child\n3->end\nEnter choice: ");
        scanf("%d", &choice);
        if(choice == 1 || choice == 2) {
            printf("Enter desired node data: ");
            scanf("%d", &d);
            x = search(root, d);
            if(x == NULL) {
                printf("Value not found\n");
                continue;
            }
            if(choice == 1 && (*x)->left != NULL) {
                printf("Node already exists there\n");
                continue;
            }
            if(choice == 2 && (*x)->right != NULL) {
                printf("Node already exists there\n");
                continue;
            }
            node* temp = getnode();
            printf("\nEnter new node data: ");
            scanf("%d", &d);
            temp->data = d;
            if(choice == 1)
                (*x)->left = temp;
            else (*x)->right = temp;
        }
        if(choice == 3)
            return;
    } while(1);
}

node** search(node** root, int value) {
    if(*root == NULL)
        return NULL;
    if((*root)->data == value)
        return root;
    node** t1 = search(&(*root)->left, value);
    if(t1 != NULL && (*t1)->data == value)
```

```
        return t1;
    node** t2 = search(&(*root)->right, value);
    return t2;
}

void inorder(node* root) {
    if(root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

void preorder(node* root) {
    if(root == NULL) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void postorder(node* root) {
    if(root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

node* getnode() {
    node* t = (node*)malloc(sizeof(node));
    if(t == NULL) {
        printf("Overflow error");
        return NULL;
    }
    t->left = NULL;
    t->right = NULL;
    return t;
}
```

## Output:

```
0 -> exit

1 -> create

2 -> preorder

3 -> inorder traversal

4 -> postorder

Enter your choice

1

Creating Binary tree


Enter root node data: 10
```

```
1->add left child
2->add right child
3->end
Enter choice: 1
Enter desired node data: 10
Enter new node data: 20

1->add left child
2->add right child
3->end
Enter choice: 2
Enter desired node data: 10
Enter new node data: 30

1->add left child
2->add right child
3->end
Enter choice: 1
Enter desired node data: 20
Enter new node data: 40

1->add left child
2->add right child
3->end
Enter choice: 2
Enter desired node data: 20
Enter new node data: 50

1->add left child
2->add right child
3->end
Enter choice: 1
Enter desired node data: 30
```

```
Enter new node data: 60

1->add left child
2->add right child
3->end
Enter choice: 3

0 -> exit
1 -> create
2 -> preorder
3 -> inorder traversal
4 -> postorder
Enter your choice
2
10 20 40 50 30 60

0 -> exit
1 -> create
2 -> preorder
3 -> inorder traversal
4 -> postorder
Enter your choice
3
40 20 50 10 60 30

0 -> exit
1 -> create
2 -> preorder
3 -> inorder traversal
4 -> postorder
Enter your choice
4
40 50 20 60 30 10
```

# Question – 4

## Problem Statement:

Write a menu-driven program for a binary search tree to

(a)Create (b) search an element (c) insert element (d) delete an element

## Source Code:

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

typedef struct node {
    int data;
    struct node *left;
    struct node *right;
}node;

node* getnode(void);
node* createTree(void);
void inorder(node* root);
node* search(node* root, node** parent, int value);
node* insertnode(node* root, int value);
node* deletenode(node* root, int value);

int main()
{
    node* bst = NULL;
    node* temp = NULL, *parent = NULL;

    int choice = 0, key = 0;
    while(1) {
            printf("\n0 -> create BST\n1 -> insert\n2 -> delete\n3 -> search\n4 ->
inorder traversal\n5 -> exit\nEnter your choice\n");
            scanf("%d",&choice);
            switch(choice)
            {
                  case 0:
                        bst = createTree();
                        break;
                  case 1:
                  if(bst == NULL) { printf("Create tree first\n"); continue; }
                        printf("\nEnter new node data: ");
                  scanf("%d", &key);
                  bst = insertnode(bst, key);
                        break;
                  case 2:
                  if(bst == NULL) { printf("Create tree first\n"); continue; }
                        printf("\nEnter data to delete: ");
                  scanf("%d", &key);
                  bst = deletenode(bst, key);
                        break;
                  case 3:
                  if(bst == NULL) { printf("Create tree first\n"); continue; }
                        printf("\nEnter data to search: ");
                  scanf("%d", &key);
```

```
                temp = search(bst, &parent, key);
                if(temp == NULL) printf("Value NOT found\n");
                else printf("Value found\n");
                        break;
                 case 4:
                        if(bst == NULL) { printf("Nothing in tree\n"); continue; }
                inorder(bst);
                        break;
            case 5:
                return 0;
                 default:
                        printf("Enter valid option between 0 to 4\n");
                        break;
            }
      }
    return 0;
}

node* search(node* root, node** parent, int value) {
    while(root != NULL) {
        if(root->data == value)
            break;
        *parent = root;
        if(value < root->data)
            root = root->left;
        else
            root = root->right;
    }
    return root;
}

node* insertnode(node* root, int value) {
    if(root == NULL) {
        node* temp = getnode();
        temp->data = value;
        root = temp;
    }
    else if(value < root->data)
        root->left = insertnode(root->left, value);
    else if(value > root->data)
        root->right = insertnode(root->right, value);
    else
        printf("node already exists\n");
    return root;
}

node* deletenode(node* root, int value) {
    node* t = root;
    node* parent = NULL, *inSucc = NULL, *parSucc = NULL, *tempChild = NULL;
    t = search(root, &parent, value);
    if(t == NULL) {
        printf("Given node not found\n");
        return root;
    }

    //if the node is found then there are 3 cases
    //first case : the node have 2 children, then find inorder successor
    if(t->left != NULL && t->right != NULL) {
```

```
        parSucc = t;
        inSucc = t->right;
        while(inSucc->left != NULL) {
            parSucc = inSucc;
            inSucc = inSucc->left;
        }
        t->data = inSucc->data;
        t = inSucc;
        parent = parSucc;
        //now run case 2 or case 3 accordingly
    }
    if(t->left != NULL)   //case 2 only left child present
        tempChild = t->left;
    else    //case 3 only right child present
        tempChild = t->right;

    if(parent == NULL)
        root = tempChild;
    else if(parent->left == t)
        parent->left = tempChild;
    else
        parent->right = tempChild;

    printf("Data deleted\n");
    free(t);
    return root;
}

node* createTree() {
    printf("\nCreating tree\n");
    node* t = getnode();
    printf("Enter root node data\n");
    scanf("%d", &t->data);
    return t;
}

node* getnode() {
    node* t = (node*)malloc(sizeof(node));
    if(t == NULL) {
        printf("Overflow error");
        return NULL;
    }
    t->left = NULL;
    t->right = NULL;
    return t;
}

void inorder(node* root) {
    if(root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
```

**Output:**

```
0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
0

Creating tree
Enter root node data
10

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
1
Enter new node data: 50

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
1
```

```
Enter new node data: 30

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
1
Enter new node data: 20

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
1
Enter new node data: 40

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
4
10 20 30 40 50
0 -> create BST
1 -> insert
```

```
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
1
Enter new node data: 60

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
3
Enter data to search: 40
Value found

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
2
Enter data to delete: 40
Data deleted

0 -> create BST
1 -> insert
2 -> delete
```

```
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
3
Enter data to search: 40
Value NOT found

0 -> create BST
1 -> insert
2 -> delete
3 -> search
4 -> inorder traversal
5 -> exit
Enter your choice
4
10 20 30 50 60
```

# *Assignment: Set - 5*

# Set-5

## Question – 2

### Problem Statement:

Write a menu-driven program to implement the following sorting techniques using an array

(a) Bubble sort
(b) Insertion sort
(c) Selection sort

### Source Code:

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

void bubbleSort(int* arr, int size);
void insertionSort(int* arr, int size);
void selectionSort(int* arr, int size);

int main() {
    int *arr;
    int size, choice;

    printf("\nEnter the size of the array: ");
    scanf("%d", &size);
    arr = (int*)malloc(sizeof(int)*size);
    printf("\nEnter the array elements:\n");
    for(int i=0; i<size; i++)
        scanf("%d", &arr[i]);

    while (1) {
        printf("\n1 -> bubble sort\n2 -> insertion sort\n3 -> selection sort\n4 ->
exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            bubbleSort(arr, size);
            break;
        case 2:
            insertionSort(arr, size);
            break;
        case 3:
            selectionSort(arr, size);
            break;
        case 4:
            return 0;
        default:
            break;
        }

        printf("\nSorted array: \n");
        for (int i = 0; i < size; i++)
```

```
            printf("%d ", arr[i]);
        printf("\n");
    }

    return 0;
}


void bubbleSort(int* arr, int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void insertionSort(int* arr, int size) {
    for (int i = 1; i < size; i++) {
        int key = arr[i];
        int j = i - 1;
        // Shift elements greater than key to the right
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void selectionSort(int* arr, int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        // Find the index of the minimum element in the remaining unsorted part
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }
        // Swap arr[i] with the minimum element
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}
```

**Output:**

```
Enter the size of the array: 5
Enter the array elements:
1 5 2 4 3

1 -> bubble sort
2 -> insertion sort
3 -> selection sort
4 -> exit
Enter your choice: 1
Sorted array:
1 2 3 4 5

Enter the array elements:
1 5 3 2 4
1 -> bubble sort
2 -> insertion sort
3 -> selection sort
4 -> exit
Enter your choice: 2

Sorted array:
1 2 3 4 5

Enter the array elements:
5 1 4 2 3
1 -> bubble sort
2 -> insertion sort
3 -> selection sort
4 -> exit
Enter your choice: 3

Sorted array:1 2 3 4 5
```

## Question – 5

Write a menu-driven program to implement the following sorting techniques using an array (recursive functions)

(a) Quick sort
(b) Merge sort

### Source Code:

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

void mergeSort(int* , int, int );
void merge(int* , int, int , int);
void quickSort(int* , int, int);
int partition(int* , int, int);
void display(int*, int );

int main()
{
    int *arr;
    int size, choice;

    printf("\nEnter the size of the array: ");
    scanf("%d", &size);
    arr = (int*)malloc(sizeof(int)*size);
    printf("\nEnter the array elements:\n");
    for(int i=0; i<size; i++)
        scanf("%d", &arr[i]);

    while(1) {
        printf("\n1 -> Merge sort\n2 -> Quick sort\n3 -> display\n4 -> exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
        case 1:
            mergeSort(arr, 0, size - 1);
            break;
        case 2:
            quickSort(arr, 0, size - 1);
            break;
        case 3:
            display(arr, size);
            break;
        case 4:
            return 0;
        default:
            break;
        }
    }

    return 0;
}

void quickSort(int* arr, int start, int end) {
```

```
    int pivot;
    if(start < end) {
        pivot = partition(arr, start, end);
        //pivot is in the correct position, thats why (pivot + 1) and (pivot - 1)
        quickSort(arr, start, pivot - 1);
        quickSort(arr, pivot + 1, end);
    }
}

int partition(int* arr, int start, int end) {
    int pivot = arr[start];
    int left = start + 1;   //as start is pivot
    int right = end;

    while(left <= right) {
        while(arr[left] < pivot && left < right)
            left++;
        while(arr[right] > pivot)
            right--;
        if(left < right) {
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
            left++;
            right--;
        }
        else
            left++;
    }

    arr[start] = arr[right];
    arr[right] = pivot;
    return right;
}

void mergeSort(int* arr, int start, int end) {
    int mid;
    if(start != end) {
        mid = (start + end) / 2;
        mergeSort(arr, start, mid);
        mergeSort(arr, mid+1, end);
        merge(arr, start, mid, end);
    }
}

void merge(int* arr, int start, int mid, int end) {
    int s = mid + 1;
    int first = start, last = end;
    int temp[50], tIndex = 0;

    while(first <= mid && s <= end) {
        if(arr[first] <= arr[s]) {
            temp[tIndex++] = arr[first++];
        }
        else {
            temp[tIndex++] = arr[s++];
        }
    }
```

```
    while (first <= mid) {
        temp[tIndex++] = arr[first++];
    }
    while(s <= end) {
        temp[tIndex++] = arr[s++];
    }

    for(int k = 0,i = start; i <= end; i++, k++)
        arr[i] = temp[k];
}

void display(int* arr, int size) {
    printf("\nArray elements: \n");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

## Output:

```
Enter the size of the array: 6

Enter the array elements:

1 6 3 5 4 2


1 -> Merge sort

2 -> Quick sort

3 -> display

4 -> exit

Enter your choice: 3

Array elements:

1 6 3 5 4 2


1 -> Merge sort

2 -> Quick sort

3 -> display

4 -> exit

Enter your choice: 1


1 -> Merge sort

2 -> Quick sort
```

```
3 -> display
4 -> exit
Enter your choice: 3

Array elements:
1 2 3 4 5 6



Enter the array elements:
1 5 2 6 3 4

1 -> Merge sort
2 -> Quick sort
3 -> display
4 -> exit
Enter your choice: 2

1 -> Merge sort
2 -> Quick sort
3 -> display
4 -> exit
Enter your choice: 3
Array elements:
1 2 3 4 5 6
```