CSA0358 DATA STRUCTURES WITH GRAPH ALGORITHMS

DAY-4:(11/08/2023)

QUESTION 1:

Write a C program to implement Binary tree traversal.

CODE:

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

 struct tnode {
     int data;
     struct tnode *left, *right;
 };

 struct tnode *root = NULL;

 /* creating node of the tree  and fill the given data */
 struct tnode * createNode(int data) {
     struct tnode *newNode;
     newNode  = (struct tnode *) malloc(sizeof(struct tnode));
     newNode->data = data;
     newNode->left = NULL;
     newNode->right = NULL;
     return (newNode);
 }

 /* inserting a new node into the tree */
 void insertion(struct tnode **node, int data) {
     if (!*node) {
         *node = createNode(data);
     } else if (data < (*node)->data) {
         insertion(&(*node)->left, data);
     } else if (data > (*node)->data) {
         insertion(&(*node)->right, data);
     }
 }

 /* post order tree traversal */
 void postOrder(struct tnode *node) {
```

```
    if (node) {
        postOrder(node->left);
        postOrder(node->right);
        printf("%d  ", node->data);
    }
    return;
}

/* pre order tree traversal */
void preOrder(struct tnode *node) {
    if (node) {
        printf("%d  ", node->data);
        preOrder(node->left);
        preOrder(node->right);
    }
    return;
}

/* inorder tree traversal */
void inOrder(struct tnode *node) {
    if (node) {
        inOrder(node->left);
        printf("%d  ", node->data);
        inOrder(node->right);
    }
    return;
}

int main() {
    int data, ch;
    while (1) {
        printf("\n1. Insertion\n2. Pre-order\n");
        printf("3. Post-order\n4. In-order\n");
        printf("5. Exit\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter ur data:");
                scanf("%d", &data);
                insertion(&root, data);
                break;
            case 2:
                preOrder(root);
                break;
```

```
                case 3:
                        postOrder(root);
                        break;
                case 4:
                        inOrder(root);
                        break;
                case 5:
                        exit(0);
                default:
                        printf("U've entered wrong opetion\n");
                        break;
            }
        }
        return 0;
}
```

OUTPUT:

```
1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:1
Enter ur data:20

1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:1
Enter ur data:15

1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:1
Enter ur data:8

1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:1
Enter ur data:12

1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:1
Enter ur data:61
```

```
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:1
Enter ur data:61

1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:2
20  15  8  12  61
1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:3
12  8  15  61  20
1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:4
8  12  15  20  61
1. Insertion
2. Pre-order
3. Post-order
4. In-order
5. Exit
Enter your choice:5

--------------------------------
Process exited after 41.55 seconds with return value 0
Press any key to continue . . .
```

QUESTION 2:

Write a C program to implement AVL Tree.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define FALSE 0
#define TRUE 1
struct node
{
	struct  node *lchild;
	int info;
	struct  node *rchild;
	int balance;
};
void inorder(struct node *ptr);
struct node *RotateLeft(struct node *pptr);
struct node *RotateRight(struct node *pptr);
struct node *insert(struct node *pptr, int ikey);
struct node *insert_left_check(struct node *pptr, int *ptaller);
struct node *insert_right_check(struct node *pptr, int *ptaller);
struct node *insert_LeftBalance(struct node *pptr);
struct node *insert_RightBalance(struct node *pptr);
struct node *del(struct node *pptr, int dkey);
struct node *del_left_check(struct node *pptr, int *pshorter);
struct node *del_right_check(struct node *pptr, int *pshorter);
struct node *del_LeftBalance(struct node *pptr,int *pshorter);
struct node *del_RightBalance(struct node *pptr,int *pshorter);
void display(struct node *ptr,int level);
int main()
{
	int choice,key;
	struct node *root = NULL;
	while(1)
	{
		printf("\n");
		printf("1.Insert\n");
		printf("2.Display\n");
		printf("3.Delete\n");
		printf("4.Inorder Traversal\n");
		printf("5.Quit\n");
		printf("\nEnter your choice : ");
		scanf("%d",&choice);
```

```
            switch(choice)
            {
             case 1:
                  printf("\nEnter the key to be inserted : ");
                  scanf("%d",&key);
                  root = insert(root,key);
                  break;
       case 2:
                  printf("\n");
                  display(root,0);
                  printf("\n");
                  break;
             case 3:
                  printf("\nEnter the key to be deleted : ");
                  scanf("%d",&key);
                  root = del(root,key);
                  break;
             case 4:
                  inorder(root);
                  break;
             case 5:
                  exit(1);

             default:
                  printf("Wrong choice\n");

            }
       }

  return 0;
}
void display(struct node *ptr,int level)
{
       int i;
       if(ptr == NULL )
            return;
       else
    {
            display(ptr->rchild, level+1);
            printf("\n");
            for (i=0; i<level; i++)
                  printf("    ");
            printf("%d", ptr->info);
            display(ptr->lchild, level+1);
```

```
        }
}
struct node *insert(struct node *pptr, int ikey)
{
        static int taller;
        if(pptr==NULL)
        {
                pptr = (struct node *) malloc(sizeof(struct node));
                pptr->info = ikey;
                pptr->lchild = NULL;
                pptr->rchild = NULL;
                pptr->balance = 0;
                taller = TRUE;
        }
        else if(ikey < pptr->info)
        {
                pptr->lchild = insert(pptr->lchild, ikey);
                if(taller==TRUE)
                        pptr = insert_left_check( pptr, &taller );
        }
        else if(ikey > pptr->info)
        {
                pptr->rchild = insert(pptr->rchild, ikey);
                if(taller==TRUE)
                        pptr = insert_right_check(pptr, &taller);
        }
        else
        {
                printf("Duplicate key\n");
                taller = FALSE;
        }
        return pptr;
}
struct node *insert_left_check(struct node *pptr, int *ptaller )
{
        switch(pptr->balance)
        {
         case 0:
                pptr->balance = 1;
                break;
         case -1:
                pptr->balance = 0;
                *ptaller = FALSE;
                        break;
```

```
        case 1:
                pptr = insert_LeftBalance(pptr);
                *ptaller = FALSE;
        }
        return pptr;
}
struct node *insert_right_check(struct node *pptr, int *ptaller )
{
        switch(pptr->balance)
        {
        case 0:
                pptr->balance = -1;
                break;
        case 1:
                pptr->balance = 0;
                *ptaller = FALSE;
                break;
        case -1:
                pptr = insert_RightBalance(pptr);
                *ptaller = FALSE;
        }
        return pptr;
}
struct node *insert_LeftBalance(struct node *pptr)
{
        struct node *aptr, *bptr;
        aptr = pptr->lchild;
        if(aptr->balance == 1)
        {
                pptr->balance = 0;
                aptr->balance = 0;
                pptr = RotateRight(pptr);
        }
        else
        {
                bptr = aptr->rchild;
                switch(bptr->balance)
                {
                case -1:
                        pptr->balance = 0;
                        aptr->balance = 1;
                        break;
                case 1:
                        pptr->balance = -1;
```

```
                aptr->balance = 0;
                break;
        case 0:
                pptr->balance = 0;
                aptr->balance = 0;
        }
        bptr->balance = 0;
        pptr->lchild = RotateLeft(aptr);
        pptr = RotateRight(pptr);
    }
    return pptr;
}
struct node *insert_RightBalance(struct node *pptr)
{
    struct node *aptr, *bptr;
    aptr = pptr->rchild;
    if(aptr->balance == -1)
    {
        pptr->balance = 0;
        aptr->balance = 0;
        pptr = RotateLeft(pptr);
    }
    else
    {
        bptr = aptr->lchild;
        switch(bptr->balance)
        {
        case -1:
                pptr->balance = 1;
                aptr->balance = 0;
                break;
        case 1:
                pptr->balance = 0;
                aptr->balance = -1;
                break;
        case 0:
                pptr->balance = 0;
                aptr->balance = 0;
        }
        bptr->balance = 0;
        pptr->rchild = RotateRight(aptr);
        pptr = RotateLeft(pptr);
    }
    return pptr;
```

```
}
struct node *RotateLeft(struct node *pptr)
{
	struct node *aptr;
	aptr = pptr->rchild;
	pptr->rchild = aptr->lchild;
	aptr->lchild = pptr;
	return aptr;
}
struct node *RotateRight(struct node *pptr)
{
	struct node *aptr;
	aptr = pptr->lchild;
	pptr->lchild = aptr->rchild;
	aptr->rchild = pptr;
	return aptr;
}
struct node *del(struct node *pptr, int dkey)
{
	struct node *tmp, *succ;
	static int shorter;
	if( pptr == NULL)
	{
		printf("Key not present \n");
		shorter = FALSE;
		return(pptr);
	}
	if( dkey < pptr->info )
	{
		pptr->lchild = del(pptr->lchild, dkey);
		if(shorter == TRUE)
			pptr = del_left_check(pptr, &shorter);
	}
	else if( dkey > pptr->info )
	{
		pptr->rchild = del(pptr->rchild, dkey);
		if(shorter==TRUE)
			pptr = del_right_check(pptr, &shorter);
	}
	else
	{
		if( pptr->lchild!=NULL  &&  pptr->rchild!=NULL )
		{
			succ = pptr->rchild;
```

```
                while(succ->lchild)
                        succ = succ->lchild;
                pptr->info = succ->info;
                pptr->rchild = del(pptr->rchild, succ->info);
                if( shorter == TRUE )
                        pptr = del_right_check(pptr, &shorter);
        }
        else
        {
                tmp = pptr;
                if( pptr->lchild != NULL )
                        pptr = pptr->lchild;
                else if( pptr->rchild != NULL)
                        pptr = pptr->rchild;
                else
                        pptr = NULL;
                free(tmp);
                shorter = TRUE;
        }
    }
    return pptr;
}
struct node *del_left_check(struct node *pptr, int *pshorter)
{
    switch(pptr->balance)
    {
        case 0:
                pptr->balance = -1;
                *pshorter = FALSE;
                break;
        case 1:
                pptr->balance = 0;
                break;
        case -1:
                pptr = del_RightBalance(pptr, pshorter);
    }
    return pptr;
}
struct node *del_right_check(struct node *pptr, int *pshorter)
{
    switch(pptr->balance)
    {
        case 0:
                pptr->balance = 1;
```

```
                *pshorter = FALSE;
                break;
        case -1:
                pptr->balance = 0;
                break;
        case 1:
                pptr = del_LeftBalance(pptr, pshorter );
    }
    return pptr;
}
struct node *del_LeftBalance(struct node *pptr,int *pshorter)
{
    struct node *aptr, *bptr;
    aptr = pptr->lchild;
    if( aptr->balance == 0)
    {
        pptr->balance = 1;
        aptr->balance = -1;
        *pshorter = FALSE;
        pptr = RotateRight(pptr);
    }
    else if(aptr->balance == 1 )
    {
        pptr->balance = 0;
        aptr->balance = 0;
        pptr = RotateRight(pptr);
    }
    else
    {
        bptr = aptr->rchild;
        switch(bptr->balance)
        {
            case 0:
                pptr->balance = 0;
                aptr->balance = 0;
                break;
            case 1:
                pptr->balance = -1;
                aptr->balance = 0;
                break;
            case -1:
                pptr->balance = 0;
                aptr->balance = 1;
        }
```

```
            bptr->balance = 0;
            pptr->lchild = RotateLeft(aptr);
            pptr = RotateRight(pptr);
        }
        return pptr;
}
struct node *del_RightBalance(struct node *pptr,int *pshorter)
{
        struct node *aptr, *bptr;
        aptr = pptr->rchild;
        if (aptr->balance == 0)
        {
            pptr->balance = -1;
            aptr->balance = 1;
            *pshorter = FALSE;
            pptr = RotateLeft(pptr);
        }
        else if(aptr->balance == -1 )
        {
            pptr->balance = 0;
            aptr->balance = 0;
            pptr = RotateLeft(pptr);
        }
        else
        {
            bptr = aptr->lchild;
            switch(bptr->balance)
            {
                case 0:
                    pptr->balance = 0;
                    aptr->balance = 0;
                    break;
                case 1:
                    pptr->balance = 0;
                    aptr->balance = -1;
                    break;
                case -1:
                    pptr->balance = 1;
                    aptr->balance = 0;
            }
            bptr->balance = 0;
            pptr->rchild = RotateRight(aptr);
            pptr = RotateLeft(pptr);
        }
```

```
        return pptr;
}
void inorder(struct node *ptr)
{
        if(ptr!=NULL)
        {
                inorder(ptr->lchild);
                printf("%d  ",ptr->info);
                inorder(ptr->rchild);
        }
}
```

OUTPUT:

```
1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 1

Enter the key to be inserted : 8

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 1

Enter the key to be inserted : 2

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 1

Enter the key to be inserted : 4

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 1

Enter the key to be inserted : 9
```

```
Enter your choice : 1

Enter the key to be inserted : 9

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 1

Enter the key to be inserted : 12

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 1

Enter the key to be inserted : 15

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 1

Enter the key to be inserted : 3

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit
```

```
1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 2


        15
    12
9
        8
    4
            3
        2

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 3

Enter the key to be deleted : 3

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 2


        15
    12
9
```

```
5.Quit

Enter your choice : 2


        15
    12
9
        8
    4
        2

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 3

Enter the key to be deleted : 9

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 2


    15
12
        8
    4
        2

1.Insert
2.Display
3.Delete
```

```
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 2


    15
12
        8
    4
        2

1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 4
2  4  8  12  15
1.Insert
2.Display
3.Delete
4.Inorder Traversal
5.Quit

Enter your choice : 5

--------------------------------
Process exited after 168.7 seconds with return value 1
Press any key to continue . . .
```

QUESTION 3:

Write a C program to implement Hashing using Linear probing techniques.

CODE:

```
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
```

```
int h[TABLE_SIZE]={NULL};

void insert()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
   {

    index=(hkey+i)%TABLE_SIZE;

    if(h[index] == NULL)
    {
       h[index]=key;
        break;
    }

   }

   if(i == TABLE_SIZE)

    printf("\nelement cannot be inserted\n");
}
void search()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
   index=(hkey+i)%TABLE_SIZE;
   if(h[index]==key)
   {
     printf("value is found at index %d",index);
     break;
   }
  }
  if(i == TABLE_SIZE)
   printf("\n value is not found\n");
```

```
}
void display()
{

  int i;

  printf("\nelements in the hash table are \n");

  for(i=0;i< TABLE_SIZE; i++)

  printf("\nat index %d \t value =  %d",i,h[i]);

}
main()
{
    int opt,i;
    while(1)
    {
        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                insert();
                break;
            case 2:
                display();
                break;
            case 3:
                search();
                break;
            case 4:exit(0);
        }
    }
}
```

OUTPUT:

```
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
12

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
13

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
22

Press 1. Insert  2. Display     3. Search       4.Exit
2

elements in the hash table are

at index 0       value =  0
at index 1       value =  0
at index 2       value =  12
at index 3       value =  13
at index 4       value =  22
at index 5       value =  0
at index 6       value =  0
at index 7       value =  0
at index 8       value =  0
at index 9       value =  0
Press 1. Insert  2. Display     3. Search       4.Exit
3

enter search element
12
value is found at index 2
```

```
enter search element
12
value is found at index 2
Press 1. Insert  2. Display     3. Search       4.Exit
3

enter search element
23

 value is not found

Press 1. Insert  2. Display     3. Search       4.Exit
4

--------------------------------
Process exited after 23.56 seconds with return value 0
Press any key to continue . . .
```

QUESTION 4:

Write a C program to implement bubble sort.

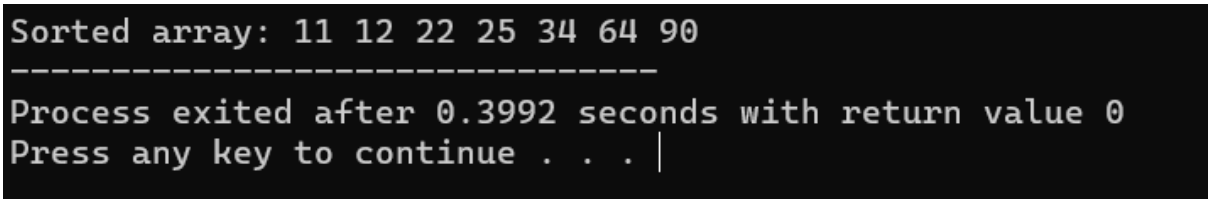
CODE:

```
#include <stdio.h>

void bubble_sort(int arr[], int n) {
  int i, j;
  for (i = 0; i < n - 1; i++) {
    for (j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}
int main() {
  int arr[] = {64, 34, 25, 12, 22, 11, 90};
  int n = sizeof(arr) / sizeof(arr[0]);
  bubble_sort(arr, n);
  printf("Sorted array: ");
  for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
```

```
  }
  return 0;
}
```

OUTPUT:



QUESTION 5:

Write a C program to implement selection sort.

CODE:

```
#include <stdio.h>
void selection(int arr[], int n)
{
   int i, j, small;
   for (i = 0; i < n-1; i++)
   {
      small = i;
      for (j = i+1; j < n; j++)
      if (arr[j] < arr[small])
         small = j;
   int temp = arr[small];
   arr[small] = arr[i];
   arr[i] = temp;
   }
}
void printArr(int a[], int n)
{
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", a[i]);
}
int main()
{
   int a[] = { 12, 31, 25, 8, 32, 17 };
   int n = sizeof(a) / sizeof(a[0]);
   printf("Before sorting array elements are - \n");
```

```
    printArr(a, n);
    selection(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    return 0;
}
```

OUTPUT:

```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
--------------------------------
Process exited after 1.271 seconds with return value 0
Press any key to continue . . .
```

QUESTION 6:

Write a C program to implement insertion sort.

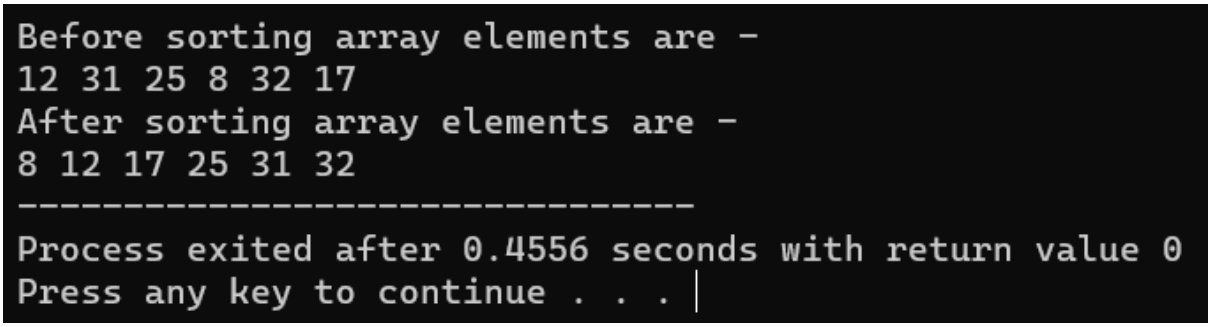
CODE:

```
#include <stdio.h>
void insert(int a[], int n)
{
    int i, j, temp;
    for (i = 1; i < n; i++) {
        temp = a[i];
        j = i - 1;
        while(j>=0 && temp <= a[j])
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = temp;
    }
}
void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}
```

```
int main()
{
   int a[] = { 12, 31, 25, 8, 32, 17 };
   int n = sizeof(a) / sizeof(a[0]);
   printf("Before sorting array elements are - \n");
   printArr(a, n);
   insert(a, n);
   printf("\nAfter sorting array elements are - \n");
   printArr(a, n);

   return 0;
}
```

OUTPUT:



QUESTION 7:

Write a C program to implement quick sort.

CODE:

```
#include <stdio.h>
void swap(int* a, int* b) {
   int t = *a;
   *a = *b;
   *b = t;
}
int partition(int arr[], int low, int high) {
   int pivot = arr[high];
   int i = (low - 1);

   for (int j = low; j <= high - 1; j++) {
      if (arr[j] < pivot) {
         i++;
         swap(&arr[i], &arr[j]);
      }
```

```
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
void printArray(int arr[], int size) {
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main() {
    int arr[] = { 12, 17, 6, 25, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

```
Sorted array:
1 5 6 12 17 25

--------------------------------
Process exited after 0.5511 seconds with return value 0
Press any key to continue . . .
```

QUESTION 8:

Write a C program to implement merge sort.

CODE:

```
#include <stdio.h>
void merge(int a[], int beg, int mid, int end)
{
```

```
    int i, j, k;
    int n1 = mid - beg + 1;
    int n2 = end - mid;
    int LeftArray[n1], RightArray[n2];
    for (int i = 0; i < n1; i++)
    LeftArray[i] = a[beg + i];
    for (int j = 0; j < n2; j++)
    RightArray[j] = a[mid + 1 + j];
    i = 0;
    j = 0;
    k = beg;
    while (i < n1 && j < n2)
    {
        if(LeftArray[i] <= RightArray[j])
        {
            a[k] = LeftArray[i];
            i++;
        }
        else
        {
            a[k] = RightArray[j];
            j++;
        }
        k++;
    }
    while (i<n1)
    {
        a[k] = LeftArray[i];
        i++;
        k++;
    }

    while (j<n2)
    {
        a[k] = RightArray[j];
        j++;
        k++;
    }
}
void mergeSort(int a[], int beg, int end)
{
    if (beg < end)
    {
        int mid = (beg + end) / 2;
```

```
        mergeSort(a, beg, mid);
        mergeSort(a, mid + 1, end);
        merge(a, beg, mid, end);
    }
}
void printArray(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
int main()
{
    int a[] = { 12, 31, 25, 8, 32, 17, 40, 42 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArray(a, n);
    mergeSort(a, 0, n - 1);
    printf("After sorting array elements are - \n");
    printArray(a, n);
    return 0;
}
```

OUTPUT:

```
Before sorting array elements are -
12 31 25 8 32 17 40 42
After sorting array elements are -
8 12 17 25 31 32 40 42

--------------------------------
Process exited after 1.097 seconds with return value 0
Press any key to continue . . .
```