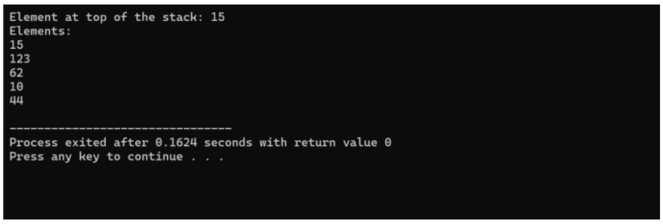
1. Write a code to implement the stack.

#SOURCE CODE

```
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;
int isempty() {
   if (top == -1)
      return 1;
   else
      return 0;
}
int isfull() {
   if (top == MAXSIZE)
      return 1;
   else
      return 0;
}
int peek() {
   return stack[top];
}
int pop() {
   int data;
   if (!isempty()) {
      data = stack[top];
      top = top - 1;
      return data;
   } else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}
```

```c
int push(int data) {
   if (!isfull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}
int main() {
   push(44);
   push(10);
   push(62);
   push(123);
   push(15);
   printf("Element at top of the stack: %d\n", peek());
   printf("Elements: \n");
   while (!isempty()) {
      int data = pop();
      printf("%d\n", data);
   }
   return 0;
}
```

#OUTPUT



```
Element at top of the stack: 15
Elements:
15
123
62
10
44

--------------------------------
Process exited after 0.1624 seconds with return value 0
Press any key to continue . . .
```

2. Write a program array to implement Circular Queue.

#SOURCE CODE

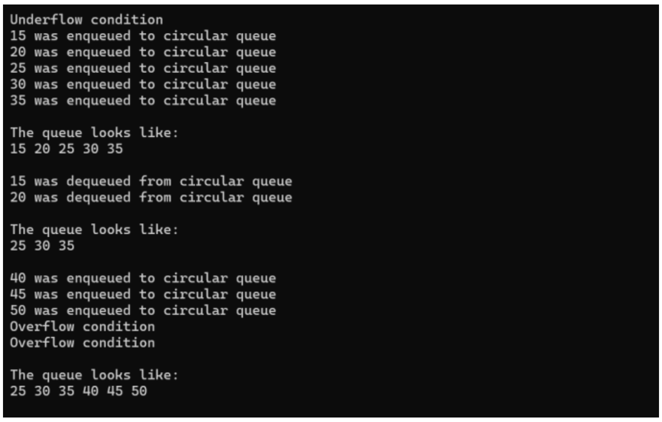
```
#include<stdio.h>
#define capacity 6
int queue[capacity];
int front = -1, rear = -1;
int checkFull ()
{
  if ((front == rear + 1) || (front == 0 && rear == capacity - 1))
    {
      return 1;
    }
  return 0;
}
int checkEmpty ()
{
  if (front == -1)
    {
      return 1;
    }
  return 0;
}
void enqueue (int value)
{
  if (checkFull ())
    printf ("Overflow condition\n");
  else
    {
      if (front == -1)
```

```
front = 0;
rear = (rear + 1) % capacity;
 queue[rear] = value;
      printf ("%d was enqueued to circular queue\n", value);
    }
}
int dequeue ()
{
  int variable;
  if (checkEmpty ())
    {
      printf ("Underflow condition\n");
      return -1;
    }
  else
    {
      variable = queue[front];
      if (front == rear)
          {
            front = rear = -1;
          }
      else
          {
            front = (front + 1) % capacity;
          }
      printf ("%d was dequeued from circular queue\n", variable);
      return 1;
    }
}
void print ()
```

```
{
  int i;
  if (checkEmpty ())
    printf ("Nothing to dequeue\n");
  else
    {
      printf ("\nThe queue looks like: \n");
      for (i = front; i != rear; i = (i + 1) % capacity)
          {
            printf ("%d ", queue[i]);
          }
      printf ("%d \n\n", queue[i]);
    }
}
int main ()
{
  dequeue ();
  enqueue (15);
  enqueue (20);
  enqueue (25);
  enqueue (30);
  enqueue (35);
  print ();
  dequeue ();
  dequeue ();
  print ();
  enqueue (40);
  enqueue (45);
  enqueue (50);
  enqueue (55);
```

```
print ();

return 0;

}
```

#OUTPUT



```
Underflow condition
15 was enqueued to circular queue
20 was enqueued to circular queue
25 was enqueued to circular queue
30 was enqueued to circular queue
35 was enqueued to circular queue

The queue looks like:
15 20 25 30 35

15 was dequeued from circular queue
20 was dequeued from circular queue

The queue looks like:
25 30 35

40 was enqueued to circular queue
45 was enqueued to circular queue
50 was enqueued to circular queue
Overflow condition
Overflow condition

The queue looks like:
25 30 35 40 45 50
```

3. Write a program to implement Bubble Sort.

#SOURCE CODE

```
#include <stdio.h>

  int main()

  {

    int i, j, a, n, number[30];

    printf("Enter the value of N \n");

    scanf("%d", &n);

    printf("Enter the numbers \n");

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

      scanf("%d", &number[i]);

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

    {

      for (j = i + 1; j < n; ++j)

      {

        if (number[i] > number[j])
```

```
            {
                a =  number[i];
                number[i] = number[j];
                number[j] = a;
             }
         }
     }
     printf("The numbers arranged in ascending order are given below \n");
     for (i = 0; i < n; ++i)
        printf("%d\n", number[i]);
          return 0;
  }
```

#OUTPUT

```
Enter the value of N
5
Enter the numbers
6
5
2
3
1
The numbers arranged in ascending order are given below
1
2
3
5
6

--------------------------------
Process exited after 18.64 seconds with return value 0
Press any key to continue . . .
```

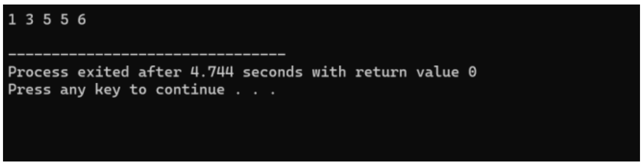
4. Write a program to sort an array in ascending order using Insertion sort.

#SOURCE CODE.

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

void insertionSort(int arr[], int n)
{
   int i, key, j;
   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;
   }
}
void printArray(int arr[], int n)
{
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}
```

```
int main()

{

   int arr[] = {5, 1, 3, 5, 6};

   int n = sizeof(arr) / sizeof(arr[0]);

   insertionSort(arr, n);

   printArray(arr, n);

   return 0;

}
```

#OUTPUT



5. Write a program to implement Selection Sort.

#SOURCE CODE

```
#include <stdio.h>

void selectionSort(int arr[], int n) {

   int i, j, minIndex, temp;

   for (i = 0; i < n - 1; i++) {

      minIndex = i;

      for (j = i + 1; j < n; j++) {

         if (arr[j] < arr[minIndex]) {

            minIndex = j;

         }

      }

      temp = arr[minIndex];

      arr[minIndex] = arr[i];
```

```
    arr[i] = temp;
   }
}
int main() {
   int arr[] = {4, 2, 1, 2, 13},i;
   int n = sizeof(arr) / sizeof(arr[0]);
   printf("Original array: ");
   for (i = 0; i < n; i++) {
       printf("%d ", arr[i]);
   }
   selectionSort(arr, n);
   printf("\nSorted array: ");
   for (i = 0; i < n; i++) {
       printf("%d ", arr[i]);
   }
   return 0;
}
```

#OUTPUT

```
Original array: 4 2 1 2 13
Sorted array: 1 2 2 4 13
--------------------------------
Process exited after 9.552 seconds with return value 0
Press any key to continue . . .
```

6. Write a program to implement the merge sort.

#SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
```

```
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
                    merge(arr, l, m, r);
    }
}
void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);
```

```c
   printf("Given array is \n");

   printArray(arr, arr_size);

   mergeSort(arr, 0, arr_size - 1);

   printf("\nSorted array is \n");

   printArray(arr, arr_size);

   return 0;

}
```

#OUTPUT

```
Given array is
12 11 13 5 6 7

Sorted array is
7 6 7 7 6 7

--------------------------------
Process exited after 7.038 seconds with return value 0
Press any key to continue . . .
```

7. Write a program to implement the Adjacency Matrix representation of graph.

#SOURCE CODE

```c
#include <stdio.h>

int N, M,i,j;

void createAdjMatrix(int Adj[][N + 1],

                int arr[][2])

{

   for (i = 0; i < N + 1; i++) {


      for (j = 0; j < N + 1; j++) {

         Adj[i][j] = 0;

      }

   }

   for (i = 0; i < M; i++) {
```

```
        int x = arr[i][0];
        int y = arr[i][1];
        Adj[x][y] = 1;
        Adj[y][x] = 1;
    }
}
void printAdjMatrix(int Adj[][N + 1])
{
            int i,j;
    for (i = 1; i < N + 1; i++) {
        for (j = 1; j < N + 1; j++) {
            printf("%d ", Adj[i][j]);
        }
        printf("\n");
    }
}
int main()
{
    N = 5;
    int arr[][2]
        = { { 1, 2 }, { 2, 3 },
            { 4, 5 }, { 1, 5 } };
    M = sizeof(arr) / sizeof(arr[0]);
    int Adj[N + 1][N + 1];
    createAdjMatrix(Adj, arr);
    printAdjMatrix(Adj);
    return 0;
}
```

#OUTPUT

```
0 1 0 0 1
1 0 1 0 0
0 1 0 0 0
0 0 0 0 1
1 0 0 1 0

--------------------------------
Process exited after 9.212 seconds with return value 0
Press any key to continue . . .
```

8. Write a program to implement Singly Linked list.

#SOURCE CODE

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* next;

};


void insertAtBeginning(struct Node** head_ref, int new_data) {

   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

   new_node->data = new_data;

   new_node->next = *head_ref;

   *head_ref = new_node;

}


void printList(struct Node* node) {

   while (node != NULL) {

      printf("%d ", node->data);

      node = node->next;

   }

   printf("\n");
```

```
}
void deleteList(struct Node** head_ref) {
   struct Node* current = *head_ref;
   struct Node* next;

   while (current != NULL) {
      next = current->next;
      free(current);
      current = next;
   }

   *head_ref = NULL;
}

int main() {
   struct Node* head = NULL;
   int choice, data;

   while (1) {
      printf("\n1. Insert\n");
      printf("2. Print\n");
      printf("3. Delete\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter data to insert: ");
            scanf("%d", &data);
```

```
            insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Linked List: ");
                printList(head);
                            break;
            case 3:
                deleteList(&head);
                printf("List deleted.\n");
                break;
            case 4:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    }
    return 0;
}
```

#OUTPUT

```
4. Exit
Enter your choice: 1
Enter data to insert: 4

1. Insert
2. Print
3. Delete
4. Exit
Enter your choice: 2
Linked List: 4

1. Insert
2. Print
3. Delete
4. Exit
Enter your choice: 3
List deleted.

1. Insert
2. Print
3. Delete
4. Exit
Enter your choice: 2
Linked List:

1. Insert
2. Print
3. Delete
4. Exit
Enter your choice:
```

9. Write a program to Implement Doubly Linked List (DLL).

#SOURCE CODE

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

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

void insertAtBeginning(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = *head_ref;
   new_node->prev = NULL;
   if (*head_ref != NULL)
      (*head_ref)->prev = new_node;
   *head_ref = new_node;
}

void insertAtEnd(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   struct Node* last = *head_ref;
   new_node->data = new_data;
   new_node->next = NULL;
   if (*head_ref == NULL) {
      new_node->prev = NULL;
```

```
      *head_ref = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
    new_node->prev = last;
}

void deleteNode(struct Node** head_ref, struct Node* del) {
    if (*head_ref == NULL || del == NULL)
        return;
    if (*head_ref == del)
        *head_ref = del->next;
    if (del->next != NULL)
        del->next->prev = del->prev;
    if (del->prev != NULL)
        del->prev->next = del->next;
    free(del);
}

void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}
```

```
int main() {
    struct Node* head = NULL;
    int choice, data;

    while (1) {
        printf("\n1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Delete\n");
        printf("4. Print\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                printf("Enter data to delete: ");
                scanf("%d", &data);
                struct Node* temp = head;
                while (temp != NULL && temp->data != data)
                    temp = temp->next;
```

```
            if (temp != NULL)
              deleteNode(&head, temp);
          else
              printf("Element not found in the list.\n");
          break;
        case 4:
          printf("Doubly Linked List: ");
          printList(head);
          break;
        case 5:
          printf("Exiting program.\n");
          exit(0);
        default:
          printf("Invalid choice. Please enter a valid option.\n");
      }
    }
  return 0; }
```

#OUTPUT

```
1. Insert at Beginning
2. Insert at End
3. Delete
4. Print
5. Exit
Enter your choice: 1
Enter data to insert at the beginning: 1

1. Insert at Beginning
2. Insert at End
3. Delete
4. Print
5. Exit
Enter your choice: 1
Enter data to insert at the beginning: 2

1. Insert at Beginning
2. Insert at End
3. Delete
4. Print
5. Exit
Enter your choice: 4
Doubly Linked List: 2 1

1. Insert at Beginning
2. Insert at End
3. Delete
4. Print
5. Exit
Enter your choice:
```

10. Write a program to implement Circular linked list.

#SOURCE CODE

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

   int data;

   struct Node* next;

};

void insertAtBeginning(struct Node** head_ref, int new_data) {

   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

   struct Node* last = *head_ref;

   new_node->data = new_data;

   new_node->next = *head_ref;

   if (*head_ref == NULL) {

      new_node->next = new_node;

   } else {

      while (last->next != *head_ref)

         last = last->next;

      last->next = new_node;

   }

   *head_ref = new_node;

}

void printList(struct Node* head) {

   struct Node* temp = head;

   if (head != NULL) {

      do {

         printf("%d ", temp->data);

         temp = temp->next;

      } while (temp != head);
```

```
  }
   printf("\n");
}
void deleteNode(struct Node** head_ref, int key) {
   if (*head_ref == NULL)
      return;
   struct Node* temp = *head_ref, *prev;
   if (temp->data == key && temp->next == *head_ref) {
      free(temp);
      *head_ref = NULL;
      return;
   }
   if (temp->data == key) {
      while (temp->next != *head_ref)
         temp = temp->next;
      temp->next = (*head_ref)->next;
      free(*head_ref);
      *head_ref = temp->next;
   }
   while (temp->next != *head_ref && temp->data != key) {
      prev = temp;
      temp = temp->next;
   }
   if (temp->data != key) {
      printf("Element not found in the list.\n");
      return;
   }
   prev->next = temp->next;
   free(temp);
}
```

```
int main() {

  struct Node* head = NULL;
  int choice, data, key;
  while (1) {
    printf("\n1. Insert at Beginning\n");
    printf("2. Print\n");
    printf("3. Delete\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
      case 1:
        printf("Enter data to insert at the beginning: ");
        scanf("%d", &data);
        insertAtBeginning(&head, data);
        break;
      case 2:
        printf("Circular Linked List: ");
      printList(head);
        break;
      case 3:
        printf("Enter data to delete: ");
        scanf("%d", &key);
        deleteNode(&head, key);
        break;
      case 4:
        printf("Exiting program.\n");
        exit(0);
      default:printf("Invalid choice. Please enter a valid option.\n");
```

```
    }
  }
  return 0;}
```

#OUTPUT

```
1. Insert at Beginning
2. Print
3. Delete
4. Exit
Enter your choice: 1
Enter data to insert at the beginning: 1

1. Insert at Beginning
2. Print
3. Delete
4. Exit
Enter your choice: 1
Enter data to insert at the beginning: 2

1. Insert at Beginning
2. Print
3. Delete
4. Exit
Enter your choice: 3
Enter data to delete: 1

1. Insert at Beginning
2. Print
3. Delete
4. Exit
Enter your choice: 2
Circular Linked List: 2

1. Insert at Beginning
2. Print
3. Delete
4. Exit
Enter your choice:
```

11. Write a program to implement KRUSKAL'S algorithm

#SOURCE CODE

```
#include <stdio.h>

#include <stdlib.h>


#define MAX_EDGES 30


struct Edge {

   int source, destination, weight;

};


struct Graph {

   int V, E;

   struct Edge* edge;

};


struct Graph* createGraph(int V, int E) {

   struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));

   graph->V = V;

   graph->E = E;

   graph->edge = (struct Edge*)malloc(E * sizeof(struct Edge));

   return graph;

}


struct Subset {

   int parent;

   int rank;

};
```

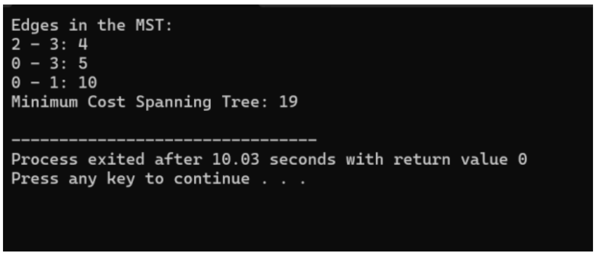
```
int find(struct Subset subsets[], int i) {
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}

void Union(struct Subset subsets[], int x, int y) {
   int xroot = find(subsets, x);
   int yroot = find(subsets, y);
   if (subsets[xroot].rank < subsets[yroot].rank)
      subsets[xroot].parent = yroot;
   else if (subsets[xroot].rank > subsets[yroot].rank)
      subsets[yroot].parent = xroot;
   else {
      subsets[yroot].parent = xroot;
      subsets[xroot].rank++;
   }
}

int compare(const void* a, const void* b) {
   struct Edge* a1 = (struct Edge*)a;
   struct Edge* b1 = (struct Edge*)b;
   return a1->weight - b1->weight;
}

void KruskalMST(struct Graph* graph) {
   int V = graph->V;
   struct Edge result[V];
   int e = 0;
   int i = 0;
```

```
  qsort(graph->edge, graph->E, sizeof(graph->edge[0]), compare);
  struct Subset* subsets = (struct Subset*)malloc(V * sizeof(struct Subset));
int v;
   for ( v = 0; v < V; v++) {
       subsets[v].parent = v;
       subsets[v].rank = 0;
   }

   while (e < V - 1 && i < graph->E) {
       struct Edge next_edge = graph->edge[i++];

       int x = find(subsets, next_edge.source);
       int y = find(subsets, next_edge.destination);

       if (x != y) {
           result[e++] = next_edge;
           Union(subsets, x, y);
       }
   }

   printf("Edges in the MST:\n");
   int minimumCost = 0;
   for (i = 0; i < e; ++i) {
       printf("%d - %d: %d\n", result[i].source, result[i].destination, result[i].weight);
       minimumCost += result[i].weight;
   }
   printf("Minimum Cost Spanning Tree: %d\n", minimumCost);
}
```

```
int main() {

    int V = 4;  // Number of vertices in graph
    int E = 5;  // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // Edge 0-1
    graph->edge[0].source = 0;
    graph->edge[0].destination = 1;
    graph->edge[0].weight = 10;

    // Edge 0-2
    graph->edge[1].source = 0;
    graph->edge[1].destination = 2;
    graph->edge[1].weight = 6;

    // Edge 0-3
    graph->edge[2].source = 0;
    graph->edge[2].destination = 3;
    graph->edge[2].weight = 5;

    // Edge 1-3
    graph->edge[3].source = 1;
    graph->edge[3].destination = 3;
    graph->edge[3].weight = 15;

    // Edge 2-3
    graph->edge[4].source = 2;
    graph->edge[4].destination = 3;
    graph->edge[4].weight = 4;
```

```
    KruskalMST(graph);

    return 0;

}
```

#OUTPUT



```
Edges in the MST:
2 - 3: 4
0 - 3: 5
0 - 1: 10
Minimum Cost Spanning Tree: 19

--------------------------------
Process exited after 10.03 seconds with return value 0
Press any key to continue . . .
```

13. Write a program to implement of Linear Search.

#SOURCE CODE

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {

int i;

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

        if (arr[i] == key)

            return i;

    }

    return -1;

}


int main() {

    int arr[] = {12, 45, 7, 23, 56, 89, 34, 67};

    int n = sizeof(arr) / sizeof(arr[0]);

    int key = 34;


    int index = linearSearch(arr, n, key);
```

```
  if (index != -1)
      printf("Element found at index: %d\n", index);
   else
      printf("Element not found in the array.\n");

   return 0;
}
```

#OUTPUT

```
Element found at index: 6

--------------------------------
Process exited after 9.566 seconds with return value 0
Press any key to continue . . .
```

14. Write a program to implement Binary search.

#SOURCE CODE

```
#include <stdio.h>
int binarySearch(int arr[], int low, int high, int key) {
   while (low <= high) {
      int mid = low + (high - low) / 2;
      if (arr[mid] == key)
         return mid;
      if (arr[mid] > key)
         high = mid - 1;
      else
         low = mid + 1;
   }
   return -1;
}

int main() {
   int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18};
   int n = sizeof(arr) / sizeof(arr[0]);
   int key = 12;
   int index = binarySearch(arr, 0, n - 1, key);
   if (index != -1)
      printf("Element found at index: %d\n", index);
   else
      printf("Element not found in the array.\n");
   return 0;
}
```

#OUTPUT

```
Element found at index: 5

--------------------------------
Process exited after 10.13 seconds with return value 0
Press any key to continue . . .
```