Vikash
School of Business Management

Class: BCA-2nd Semester
Practical: Data Structure Lab
Paper: Core Course-4 Practical
(Sambalpur University, Burla)

DATA STRUCTURE
LAB RECORD

Submitted By :- Karri Sunil Reddy

Roll No :- S3321BCA10

BCA 2nd Semester

Submitted by : Mr. Karri Sunil Reddy
Roll No :- S3321BCA10
for Academic Session-2021/2022

Questions

1. To insert and delete elements from appropriate position in an array.

2. To search an element and print the total time of occurrence in the array.

3. To delete all occurrence of an element in an array.

4. Array implementation of Stack.

5. Array implementation of Linear Queue.

6. Array implementation of Circular Queue.

7. To implement linear linked list and perform different operation such as node insert and delete, search of an item, reverse the list.

8. To implement circular linked list and perform different operation such as node insert and delete.

9. To implement double linked list and perform different operation such as node insert and delete.

10. Linked list implementation of Stack.

11. Linked list implementation of Queue.

12. Polynomial representation using linked list.

13. To implement a Binary Search Tree.

14. To represent a Sparse Matrix.

15. To perform binary search operation.

16. To perform Bubble sort.

17. To perform Selection sort.

18. To perform Insertion sort.

19. To perform Quick sort.

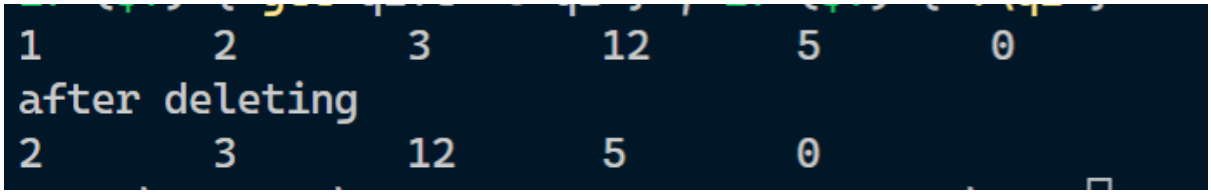
20. To perform Merge sort.

Expt no.-1

```
/*
To insert and delete elements from appropriate position
in an array.
*/

struct Array
{
  int A[10];

  int size;
  int length;
};

int insert(struct Array *arr, int num, int index)
{

  for (int i = arr->length; i < index; i++)
  {
    arr->A[i] = arr->A[i - 1];
  }
  arr->A[index] = num;
  arr->length++;

  return 0;
}
int delete(struct Array *arr, int index)
{
  for (int i = index; i < arr->length; i++)
  {
    arr->A[i] = arr->A[i + 1];
  }
  arr->length--;
}
```

```
void display(struct Array arr)
{
  for (int i = 0; i < arr.length; i++)
  {
    printf("%d \n", arr.A[i]);
  }
}

int main()
{
  struct Array arr = {{1, 2, 3, 4, 5}, 10, 5};
  insert(&arr, 12, 3);
  display(arr);
  delete (&arr, 0);
  printf("\n after deleteing \n");
  display(arr);

  return 0;
}
```

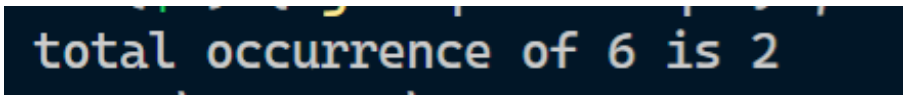
Output :



Expt no.-2

```
// To search an element and print the total time of
occurrence in the array.


#include <stdio.h>

struct Array
{
  int A[10];

  int size;
  int length;
};
int search_element(struct Array arr, int val)
{
  int i, count = 0;
  for (i = 0; i < arr.length; i++)

    if (arr.A[i] == val)
    {
      count++;
    }

  printf("total occurrence of %d is %d ", val, count);
}
int main()
{
  struct Array arr = {{1, 2, 3, 4, 5,5,6,6}, 10,8 };
  search_element(arr, 6);

  return 0;
}
```

Output:



Expt no.-3

```
// To delete all occurrence of an element in an array.

#include <stdio.h>
struct Array
{
  int A[10];

  int size;
  int length;
};

int remove_element(struct Array *arr, int val)
{
  int i;

  for (i = 0; i < arr->length; i++)

    if (arr->A[i] != val)
      printf("%d \t ", arr->A[i]);
}

int main()
{
```

```
struct Array arr = {{1, 2, 3, 4, 5}, 10, 5};
  remove_element(&arr, 1);
}
```

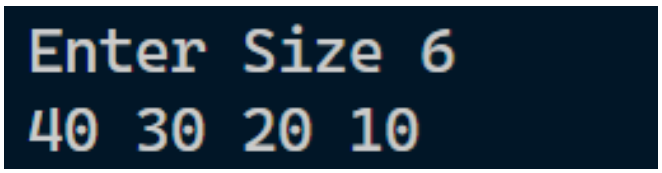
Output:

Expt no.-4

```
// Array implementation of Stack.
#include <stdio.h>
#include <stdlib.h>
struct Stack
{
  int size;
  int top;
  int *S;
};
void create(struct Stack *st)
{
  printf("Enter Size");
  scanf("%d", &st->size);
  st->top = -1;
  st->S = (int *)malloc(st->size * sizeof(int));
}

void push(struct Stack *st, int x)
{


 if (st->top == st->size - 1)
    printf("Stack overflow\n");
```

```
  else
  {
    st->top++;
    st->S[st->top] = x;
  }
}
void Display(struct Stack st)
{
  int i;
  for (i = st.top; i >= 0; i--)
    printf("%d ", st.S[i]);
  printf("\n");
}
int main()
{
  struct Stack st;
  create(&st);

  push(&st, 10);
  push(&st, 20);
  push(&st, 30);
  push(&st, 40);

  Display(st);

  return 0;
}
```

Output:



Expt no.-5

```
// Array implementation of Linear Queue.

#include <stdio.h>
#include <stdlib.h>
struct Queue
{
  int size;
  int front;
  int rear;
  int *Q;
};
void create(struct Queue *q, int size)
{
  q->size = size;
  q->front = q->rear = -1;
  q->Q = (int *)malloc(q->size * sizeof(int));
}
void enqueue(struct Queue *q, int x)
{
  if (q->rear == q->size - 1)
    printf("Queue is Full");
  else
  {
    q->rear++;
    q->Q[q->rear] = x;
  }
}
void Display(struct Queue q)
{
```

```
  int i;

  for (i = q.front + 1; i <= q.rear; i++)
    printf("%d \t", q.Q[i]);
  printf("\n");
}
int main()
{
  struct Queue q;
  create(&q, 5);

  enqueue(&q, 10);
  enqueue(&q, 20);
  enqueue(&q, 30);
  Display(q);

  return 0;
}
```

Output:

Expt no.-6

```
// Array implementation of Circular Queue.

#include <stdio.h>
#include <stdlib.h>
struct Queue
{
  int size;
  int front;
  int rear;
  int *Q;
};
void create(struct Queue *q, int size)
{
  q->size = size;
  q->front = q->rear = 0;
  q->Q = (int *)malloc(q->size * sizeof(int));
}
void enqueue(struct Queue *q, int x)
{
  if ((q->rear + 1) % q->size == q->front)
    printf("Queue is Full");
  else
  {
    q->rear = (q->rear + 1) % q->size;
    q->Q[q->rear] = x;
  }
}
void Display(struct Queue q)
{
  int i = q.front + 1;

  do
  {

    printf("%d ", q.Q[i]);
    i = (i + 1) % q.size;
  } while (i != (q.rear + 1) % q.size);
```

```
  printf("\n");
}
int main()
{
  struct Queue q;
  create(&q, 5);

  enqueue(&q, 10);
  enqueue(&q, 20);
  enqueue(&q, 30);
  enqueue(&q, 40);

  Display(q);

  return 0;
}
```

Output:

Expt no.-7

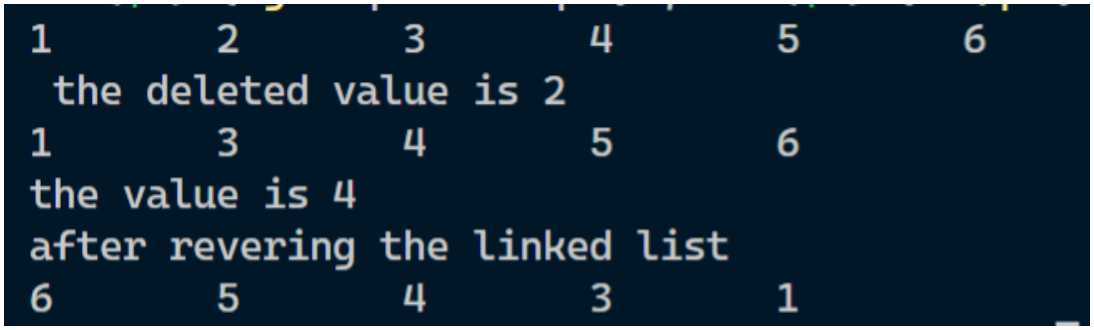
```
// To implement linear linked list and perform different
operation such as node insert and delete, temp of an item,
reverse the list.

#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *next;
} *first = NULL;

void insert(int pos, int value)
{
  struct Node *t, *p;
  if (pos == 0)
  {
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = value;
    t->next = first;
    first = t;
  }
  else
  {
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = value;
    p = first;
    for (int i = 0; i < pos - 1; i++)
    {
      p = p->next;
    }
    t->next = p->next;
    p->next = t;
```

```
  }
}

int delete(int pos)
{
  struct Node *p, *q;
  int x = -1;
  p = first;
  q = NULL;
  if (pos == 1)
  {
    x = q->data;
    first = first->next;
    free(p);
  }
  else
  {
    for (int i = 0; i < pos - 1; i++)
    {
      q = p;
      p = p->next;
    }
    if (pos != 0)
    {
      q->next = p->next;
      x = p->data;
      free(p);
    }
  }
  return x;
}

struct Node *Search(int key)
{
  struct Node *p = first;

  while (p != 0)
```

```
  {
    if (key == p->data)
    {
      return p;
    }
    p = p->next;
  }
  return NULL;
}

void Reverse()
{
  struct Node *p = first, *q = NULL, *r = NULL;

  while (p != 0)
  {
    r = q;
    q = p;
    p = p->next;
    q->next = r;
  }
  first = q;
}

void display()
{
  struct Node *p;
  p = first;
  while (p != 0)
  {
    printf("%d \t", p->data);
    p = p->next;
  }
}

int main()
{
```

```
  insert(0, 1);
  insert(1, 2);
  insert(2, 3);
  insert(3, 4);
  insert(4, 5);
  insert(5, 6);
  display();

  printf("\n the deleted value is %d \n", delete (2));
  display();

  struct Node *temp = Search(4);
  if (temp)
  {
    printf("\n the value is %d \n", temp->data);
  }
  else
  {
    printf("\n value is not found \n");
  }

  Reverse();
  printf("after revering the linked list \n");
  display();
  return 0;
}
```

Output:



Expt no.-8

```
// To implement circular linked list and perform different
operation such as node insert and delete.

#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *next;
} *head = NULL;

void insert(int pos, int value)
{
  struct Node *t, *p;
  if (pos == 0)
  {
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = value;
    if (head == 0)
    {
      head = t;
      t->next = head;
    }
    else
    {

      t->next = head;
      for (int i = 0; i < value - 1; i++)
      {
        p = p->next;
      }
      p->next = t;
    }
  }
  else
  {
```

```
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = value;
    p = head;
    for (int i = 0; i < pos - 1; i++)
    {
      p = p->next;
    }
    t->next = p->next;
    p->next = t;
  }
}

int delete(int pos)
{
  struct Node *p, *q;
  int x = -1;
  p = head;
  q = NULL;
  if (pos == 1)
  {
    while (p != head)
    {
      p = p->next;
    }
    x = head->data;
    if (head == p)
    {
      free(head);
      head = NULL;
    }
    else
    {
      p->next = head->next;
      free(head);
      head = p->next;
    }
```

```
    head = head->next;
    free(p);
  }
  else
  {
    for (int i = 0; i < pos - 1; i++)
    {
      q = p;
      p = p->next;
    }
    if (pos != 0)
    {
      q->next = p->next;
      x = p->data;
      free(p);
    }
  }
  return x;
}

void display()
{
  struct Node *p;
  p = head;

  do
  {
    printf("%d \t", p->data);
    p = p->next;
  } while (p != head);
}

int main()
{
  insert(0, 1);
  insert(1, 2);
  insert(2, 3);
```

```
  insert(3, 4);
  insert(4, 5);
  insert(5, 6);
  display();

  printf("\n the deleted value is %d \n", delete (5));
  display();

  return 0;
}
```

Output:

```
1       2       3       4       5       6
 the deleted value is 5
1       2       3       4       6
```

Expt no.-9

```
// To implement double linked list and perform different
operation such as node insert delete

#include <stdio.h>
#include <stdlib.h>
struct Node
{
  struct Node *prev;
  int data;
  struct Node *next;

} *first = NULL;

void creat(int A[], int n)
{
  int i;
  struct Node *t, *last;
  first = (struct Node *)malloc(sizeof(struct Node));
  first->data = A[0];
  first->next = first->prev = NULL;
  last = first;

  for (int i = 1; i < n; i++)
  {
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = A[i];
    t->next = last->next;
    t->prev = last;
    last->next = t;
    last = t;
  }
}
void insert(int pos, int val)
{
```

```
  struct Node *p, *t;

  p = first;
  if (pos == 0)
  {
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = val;
    t->prev = NULL;
    t->next = first;
    first->prev = t;
    first = t;
  }
  else
  {
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = val;
    for (int i = 0; i < pos - 1; i++)
    {
      p = p->next;
    }

    t->next = p->next;

    t->prev = p;
    if (p->next)
    {
      t->next->prev = t;
    }

    p->next = t;
  }
}
int deleteDoubly(int pos)
{
  int val;
  struct Node *p = first;
  p = first;
```

```
  if (pos == 1)
  {
    first = first->next;
    val = p->data;
    if (first)
    {
      first->prev = NULL;
    }
    free(p);
  }
  else
  {
    for (int i = 0; i < pos - 1; i++)
    {
      p = p->next;
    }

    val = p->data;
    p->prev->next = p->next;
    if (p->next)
    {
      p->next->prev = p->prev;
    }
    free(p);
  }
  return val;
}

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

```
int main()
{

  int A[] = {1, 2, 3, 4, 5};
  creat(A, 5);
  insert(0, 6);
  insert(1, 10);

  display(first);

  printf("\n deleted value is %d \n", deleteDoubly(4));
  display(first);
  return 0;
}
```

Output:

```
6       10      1       2       3       4       5
 deleted value is 2
6       10      1       3       4       5
```

Expt no.–10

```
//  Linked list implementation of Stack.

#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *next;
```

```
} *top = NULL;
void push(int x)
{
  struct Node *t;
  t = (struct Node *)malloc(sizeof(struct Node));

  if (t == NULL)
    printf("stack is full\n");
  else
  {
    t->data = x;
    t->next = top;
    top = t;
  }
}
void Display()
{
  struct Node *p;
  p = top;
  while (p != NULL)
  {
    printf("%d ", p->data);
    p = p->next;
  }
  printf("\n");
}
int main()
{
  push(10);
  push(20);
  push(30);

  Display();

  return 0;
}
```

Output:

Expt no.-11

```
// Linked list implementation of Queue.

#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *next;

} *front = NULL, *rear = NULL;
void enqueue(int x)
{
  struct Node *t;
  t = (struct Node *)malloc(sizeof(struct Node));
  if (t == NULL)
    printf("Queue is FUll\n");
  else
  {
    t->data = x;
    t->next = NULL;
    if (front == NULL)
      front = rear = t;
    else
```

```
    {
      rear->next = t;
      rear = t;
    }
  }
}
void Display()
{
  struct Node *p = front;
  while (p)
  {
    printf("%d \t", p->data);
    p = p->next;
  }
  printf("\n");
}
int main()
{
  enqueue(10);
  enqueue(20);
  enqueue(30);
  enqueue(40);
  enqueue(50);

  Display();

  return 0;
}
```

Output:

Expt no.-12

```
// Polynomial representation using linked list.

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

struct term
{
  int coefficient;
  int exponent;
  struct term *next;
};

struct polynomial
{
  struct term *head;
};

struct term *create_term(int coefficient, int exponent)
{
  struct term *new_term = malloc(sizeof(struct term));
  new_term->coefficient = coefficient;
  new_term->exponent = exponent;
  new_term->next = NULL;
  return new_term;
}

struct polynomial *create_polynomial()
{
  struct polynomial *new_poly = malloc(sizeof(struct
polynomial));
  new_poly->head = NULL;
  return new_poly;
}

void insert_term(struct polynomial *poly, int coefficient,
int exponent)
```

```
{
  struct term *new_term = create_term(coefficient,
exponent);

  if (poly->head == NULL)
  {
    poly->head = new_term;
    return;
  }

  struct term *cur = poly->head;
  struct term *prev = NULL;
  while (cur != NULL && exponent < cur->exponent)
  {
    prev = cur;
    cur = cur->next;
  }

  if (prev == NULL)
  {
    new_term->next = poly->head;
    poly->head = new_term;
  }
  else
  {
    new_term->next = cur;
    prev->next = new_term;
  }
}

void print_polynomial(struct polynomial *poly)
{
  struct term *cur = poly->head;
  while (cur != NULL)
  {
    printf("%dx^%d", cur->coefficient, cur->exponent);
    cur = cur->next;
```

```
    if (cur != NULL)
    {
      printf(" + ");
    }
  }
}

int main()
{
  struct polynomial *poly = create_polynomial();

  insert_term(poly, 3, 2);
  insert_term(poly, 2, 1);
  insert_term(poly, 1, 0);

  print_polynomial(poly);
}
```

Output:

```
3x^2 + 2x^1 + 1x^0
```

Expt no.–13

```
// To implement a Binary Search Tree

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

struct node
{
  int key;
  struct node *left, *right;
```

```
};


struct node *newNode(int item)
{
  struct node *temp = (struct node *)malloc(sizeof(struct
node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}

void inorder(struct node *root)
{
  if (root != NULL)
  {
    inorder(root->left);

    printf("%d -> ", root->key);

    inorder(root->right);
  }
}

struct node *insert(struct node *node, int key)
{
  if (node == NULL)
    return newNode(key);

  if (key < node->key)
    node->left = insert(node->left, key);
  else
    node->right = insert(node->right, key);

  return node;
}

struct node *minValueNode(struct node *node)
```

```
{
  struct node *current = node;

  while (current && current->left != NULL)
    current = current->left;

  return current;
}

struct node *deleteNode(struct node *root, int key)
{
  if (root == NULL)
    return root;

  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);

  else
  {
    if (root->left == NULL)
    {
      struct node *temp = root->right;
      free(root);
      return temp;
    }
    else if (root->right == NULL)
    {
      struct node *temp = root->left;
      free(root);
      return temp;
    }

    struct node *temp = minValueNode(root->right);

    root->key = temp->key;
```

```
    root->right = deleteNode(root->right, temp->key);
  }
  return root;
}

int main()
{
  struct node *root = NULL;
  root = insert(root, 8);
  root = insert(root, 3);
  root = insert(root, 1);
  root = insert(root, 6);
  root = insert(root, 7);
  root = insert(root, 10);
  root = insert(root, 14);
  root = insert(root, 4);

  printf("Inorder traversal: ");
  inorder(root);

  printf("\nAfter deleting 10\n");
  root = deleteNode(root, 10);
  printf("Inorder traversal: ");
  inorder(root);
}
```

Output:

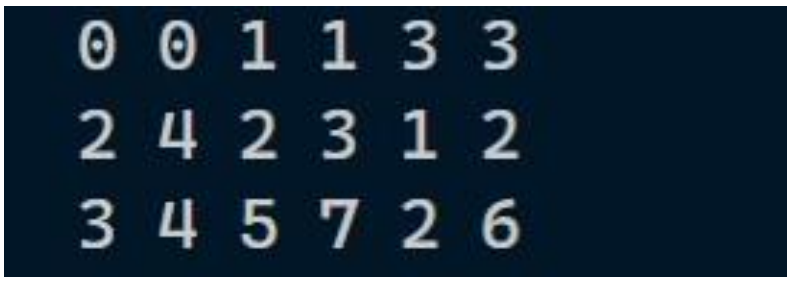
```
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->
After deleting 10
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 14 ->
```

Expt no.–14

```
// . To represent a Sparse Matrix.
#include <stdio.h>

int main()
{
  int sparseMatrix[4][5] =
      {
          {0, 0, 3, 0, 4},
          {0, 0, 5, 7, 0},
          {0, 0, 0, 0, 0},
          {0, 2, 6, 0, 0}};

  int size = 0;
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 5; j++)
      if (sparseMatrix[i][j] != 0)
        size++;

  int compactMatrix[3][size];

  int k = 0;
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 5; j++)
      if (sparseMatrix[i][j] != 0)
      {
        compactMatrix[0][k] = i;
        compactMatrix[1][k] = j;
        compactMatrix[2][k] = sparseMatrix[i][j];
        k++;
      }

  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < size; j++)
      printf("%d ", compactMatrix[i][j]);
```

```
    printf("\n");
  }
  return 0;
}
```

Output:



Expt no.–15

```
// To perform binary search operation.


#include <stdio.h>

int binarySearch(int array[], int x, int low, int high)
{

  while (low <= high)
  {
    int mid = low + (high - low) / 2;

    if (array[mid] == x)
      return mid;

    if (array[mid] < x)
      low = mid + 1;
```

```
    else
      high = mid - 1;
  }

  return -1;
}

int main(void)
{
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
  return 0;
}
```

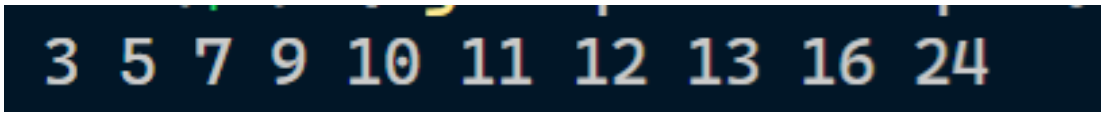
Output:

Expt no.-16

```
// . To perform Bubble sort.

#include <stdio.h>
#include <stdlib.h>
void swap(int *x, int *y)
```

```
{
  int temp = *x;
  *x = *y;
  *y = temp;
}
void Bubble(int A[], int n)
{
  int i, j, flag = 0;

  for (i = 0; i < n - 1; i++)
  {
    flag = 0;
    for (j = 0; j < n - i - 1; j++)
    {
      if (A[j] > A[j + 1])
      {
        swap(&A[j], &A[j + 1]);
        flag = 1;
      }
    }
    if (flag == 0)
      break;
  }
}
int main()
{
  int A[] = {11, 13, 7, 12, 16, 9, 24, 5, 10, 3}, n = 10,
i;

  Bubble(A, n);

  for (i = 0; i < 10; i++)
    printf("%d ", A[i]);
  printf("\n");

  return 0;
}
```

Output:

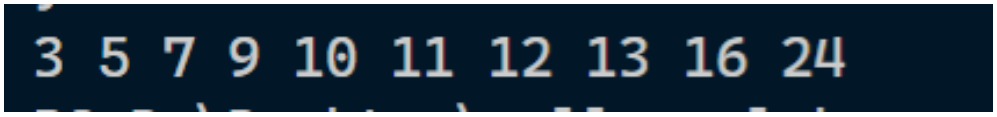


Expt no.-17

```
// To perform Selection sort.

#include <stdio.h>
#include <stdlib.h>
void swap(int *x, int *y)
{
  int temp = *x;
  *x = *y;
  *y = temp;
}
void SelectionSort(int A[], int n)
{
  int i, j, k;

  for (i = 0; i < n - 1; i++)
  {
    for (j = k = i; j < n; j++)
    {
      if (A[j] < A[k])
        k = j;
    }
    swap(&A[i], &A[k]);
  }
```

```
}
int main()
{
  int A[] = {11, 13, 7, 12, 16, 9, 24, 5, 10, 3}, n = 10,
i;

  SelectionSort(A, n);

  for (i = 0; i < 10; i++)
    printf("%d ", A[i]);
  printf("\n");

  return 0;
}
```

Output:



Expt no.-18

```
// To perform Insertion sort.

#include <stdio.h>
#include <stdlib.h>
void swap(int *x, int *y)
{
  int temp = *x;
  *x = *y;
```

```
  *y = temp;
}
void Insertion(int A[], int n)
{
  int i, j, x;

  for (i = 1; i < n; i++)
  {
    j = i - 1;
    x = A[i];
    while (j > -1 && A[j] > x)
    {
      A[j + 1] = A[j];
      j--;
    }
    A[j + 1] = x;
  }
}
int main()
{
  int A[] = {11, 13, 7, 12, 16, 9, 24, 5, 10, 3}, n = 10,
i;

  Insertion(A, n);

  for (i = 0; i < 10; i++)
    printf("%d ", A[i]);
  printf("\n");

  return 0;
}
```

Output:

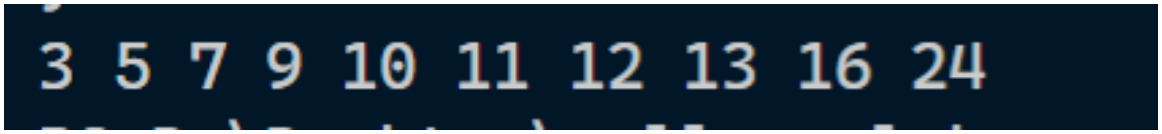
```
3 5 7 9 10 11 12 13 16 24
```

Expt no.-19

```
// Quick sort in C

#include <stdio.h>

void swap(int *a, int *b)
{
  int t = *a;
  *a = *b;
  *b = t;
}

int partition(int array[], int low, int high)
{

  int pivot = array[high];

  int i = (low - 1);

  for (int j = low; j < high; j++)
  {
    if (array[j] <= pivot)
    {

      i++;

      swap(&array[i], &array[j]);
    }
  }

  swap(&array[i + 1], &array[high]);
```

```
  return (i + 1);
}

void quickSort(int array[], int low, int high)
{
  if (low < high)
  {
    int pi = partition(array, low, high);

    quickSort(array, low, pi - 1);

    quickSort(array, pi + 1, high);
  }
}

void display(int array[], int size)
{
  for (int i = 0; i < size; ++i)
  {
    printf("%d\t ", array[i]);
  }
}

int main()
{
  int data[] = {11, 13, 7, 12, 16, 9, 24, 5, 10, 3};

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

  quickSort(data, 0, n - 1);

  display(data, n);
}
```

Output:



Expt no.-20

```
#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];

  i = 0;
  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 display(int A[], int size)
{
  int i;
  for (i = 0; i < size; i++)
    printf("%d ", A[i]);
  printf("\n");
}

int main()
{
  int arr[] = {11, 13, 7, 12, 16, 9, 24, 5, 10, 3};
  int arr_size = sizeof(arr) / sizeof(arr[0]);

  mergeSort(arr, 0, arr_size - 1);

  display(arr, arr_size);
  return 0;
}
```

Output: