#DSUC

13/12/24

#Last min. Revision

①. Linear Search Algorithm & (Program)

```
#include <stdio.h>
int linear_search (int arr[], int N, int x)
  { for (int i=0; i<N; i++)
      if (arr[i] == x)
        return i;
    return -1; }

// Driver code
int main (void) {
  int arr[] = {2, 3, 4, 10, 40};
  int x = 10;
  int N = sizeof(arr) / sizeof(arr[0]);

  // Function call
  int result = linear_search(arr, N, x);
  (result == -1) ? printf("Element is not present in Array")
                 : printf("Element is present at index %d", result);
  return 0; }
```

---

②. Binary Search Algorithm & (Iterative Approach) Program

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

```
if (arr[mid] < x)
    low = mid + 1;
else
    high = mid - 1; }
return -1; }

int main (void) {
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 10;
    int result = binary_search(arr, 0, n-1, x);
if (result == -1)
    printf ("Element is not present in array");
else printf("Element is present at index %d", result); }
```

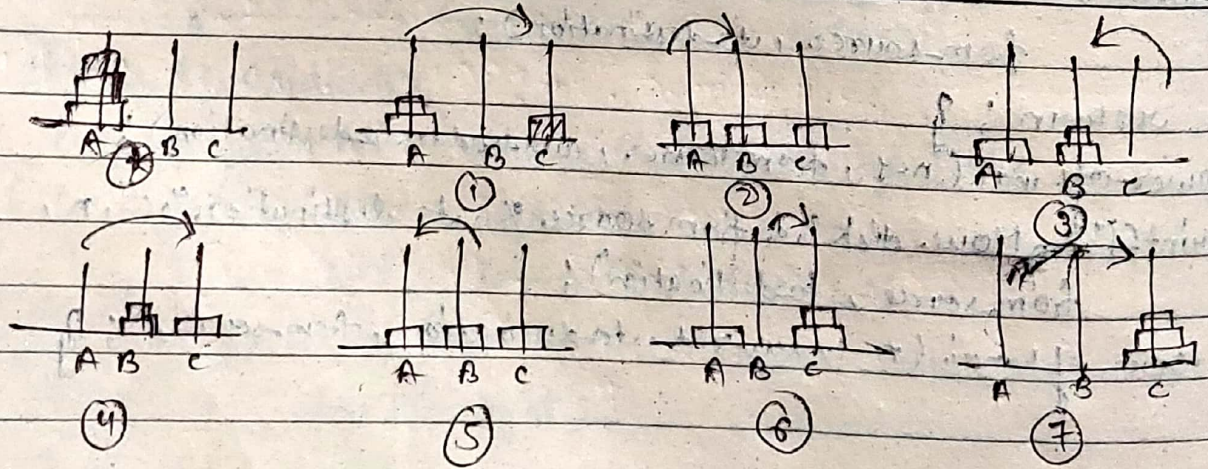
* Tower of Hanoi Algorithm (Program)

```
void tower_of_hanoi (int n, char from_source, char to_destination,
                     char aux_rod) {
    if (n == 1) {
printf ("\n Move disk 1 from source %c to destination %c",
        from_source, to_destination);
    return; }
tower_of_hanoi (n-1, from_source, aux_rod, to_destination);
printf("\n Move disk %d from source %c to destination %c", n,
        from_source, to_destination);
tower_of_hanoi (n-1, aux_rod, to_destination, from_source); }
```

```
int main() {
    int n = 4;          // No. of disks.
    tower_of_hanoi(n, 'A', 'C', 'B');
    return 0; }         // A, B, C are the names of rods.
```

Output:

Move disk 1 from source A to destination B
" " 2 " " A " " C
" " 1 " " B " " C
" " 3 " " A " " B
" " 1 " " C " " A
" " 2 " " C " " B
" " 1 " " A " " B
" " 4 " " A " " C
" " 1 " " B " " C
" " 2 " " B " " A
" " 1 " " C " " A
" " 3 " " B " " C
" " 1 " " A " " B
" " 2 " " A " " C
" " 1 " " B " " C



Dijkstra's Algorithm (Program)

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

#define V 9

int minDistance(int dist[], bool sptSet[])
{   int min = INT_MAX, min_index;
    for(int v=0; v<V; v++)
        if(sptSet[v]==false && dist[v]<=min)
            min = dist[v], min_index = v;
    return min_index;
}
void printSolution(int dist[])
{ printf("Vertex \t\t Distance from source.\n");
    for(int i=0; i<V; i++)
        printf("%d \t\t\t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src)
{ int dist[V];
    bool sptSet[V];
    for(int i=0; i<V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
    for(int count=0; count<V-1; count++){
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
```

```
for (int v=0; v<V; v++)
    if (!sptSet[v] && graph[u][v]
        && dist[u] != INT_MAX
        && dist[u] + graph[u][v] < dist[v])
    dist[v] = dist[u] + graph[u][v]; }

printSolution(dist); }

int main() {
    int graph[V][V] = { {0,4,0,0,0,0,0,8,0},
                        {4,0,8,0,0,0,0,11,0},
                        {0,8,0,7,0,4,0,0,2},
                        {0,0,7,0,9,14,0,0,0},
                        {0,0,0,9,0,10,0,0,0},
                        {0,0,4,14,10,0,2,0,0},
                        {0,0,0,0,0,2,0,1,6},
                        {8,11,0,0,0,0,1,0,7},
                        {0,0,2,0,0,0,6,7,0} };

    dijkstra(graph, 0);
    return 0; }
```

Output:

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

Bubble Sort Implementation

```
#include <stdio.h>
void swap (int *arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp; }

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

int main() {
    int arr[] = {6, 0, 3, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort (arr, n);
    for (int i = 0; i < n; i++)
        printf ("%d", arr[i]);
    return 0; }
```

* Implementing Stack Operation Using Array *

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

                                    // A structure to represent a stack
struct stack { int top;
               unsigned capacity;
               int * array; };

struct stack * createStack (unsigned capacity)
{ struct stack* stack = (struct stack*) malloc (sizeof (struct
  stack));
  stack->capacity = capacity;
  stack->top = -1;
  stack->array = (int*) malloc (stack->capacity *
    sizeof(int));
  return stack; }

int isFull (struct stack * stack) {
  return stack->top == stack->capacity - 1; }

int isEmpty (struct stack * stack) {
  return stack->top == -1; }

void push (struct stack * stack, int item) {
  if (isFull (stack))
    return;
  stack->array[++stack->top] = item;
  printf ("%d pushed to stack\n", item); }
```

```
int pop(struct stack* stack) {
        if (isEmpty(stack))
            return INT_MIN;
        return stack->array[stack->top--]; }
int peek(struct stack* stack) {
        if (isEmpty(stack))
            return INT_MIN;
        return stack->array[stack->top]; }

int main() {
    struct stack* stack = createStack(100);
    push(stack, 10);
    push(stack, 20);
    push(stack, 30);

    printf("%d popped from stack\n", pop(stack));
    return 0; }
```

Output:

```
10 pushed into stack
20 "    "    "
30 "    "    "
30 popped from "
Top Element is : 20
Element present in stack : 20 10.
```

* Insertion Sort

```
#include <math.h>
#include <stdio.h>
void insertionSort (int arr[], int N) {
    for (int i=1; i<N; i++) {
        int key = arr[i];
        int j = i-1;
    while (j>=0 && arr[j] > key) {
        arr[j+1] = arr[j]
            j = j-1; }

        arr[j+1] = key; }}
int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int N = sizeof(arr) / sizeof(arr[0]);
    printf("Unsorted Array: ");
    for (int i=0; i<N; i++) {
        printf("%d", arr[i]); }
    printf("\n");

    insertionSort (arr, N);
    printf("Sorted Array: ");
    for (int i=0; i<N; i++) {
        printf("%d", arr[i]); }
    printf("\n");
    return 0; }
```

output:

Unsorted Array:- 12 11 13 5 6

Sorted Array:- 5 6 11 12 13

* Selection Sort :-

```
#include <stdio.h>
void selectionSort (int arr[], int n) {
    for (int i=0; i<n-1; i++) {
        int min_idx = i;
        for (int j=i+1; j<n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j; } }

    int temp = arr[i];
    arr[i] = arr[min_idx];
    arr[min_idx] = temp; } }

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

int main () {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original Array: ");
    printArray(arr, n);
    selectionSort(arr, n);
    printf("Sorted Array: ");
    printArray(arr, n);
    return 0; }
```

Output

Original Vector:- 64 25 12 22 11

Sorted Vector:- 11 12 22 25 64