Name - Chiradeep Banik

Enrol. No. - 20UCS176

Sec - A

Q.1

Implementation of stack data-structure

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

struct stack{
    int* array;
    unsigned int capacity;
    int top;
};
typedef struct stack Stack;

Stack* create_stack(unsigned int capacity){
    Stack* s=(Stack*) malloc(sizeof(Stack));
    s->array=(int*) malloc(capacity * sizeof(int));
    s->capacity = capacity;
    s->top=0;
    return s;
}
```

```
void push(stack* s, int data){
    if(s->top == s->capacity){
        printf("Stack OverFlow\n");
        return;
    }
    s->array[s->top] = data;
    s->top++;
    printf("%d pushed to stack\n", data);
}
int pop(stack* s){
    if(s->top == 0){
        printf("Stack UnderFlow\n");
        return INT_MIN;
    }
    s->top--;
    int temp = s->array[s->top];
    printf("%d popped from stack\n", temp);
    return temp;
}
void main(){
```

```
void main(){
    Stack* s = create_stack(2);
    push(s,10);
    push(s,20);
    push(s,30);
    pop(s);
    pop(s);
    pop(s);
}
```

Console

↳ → 10 pushed to stack
20 pushed to stack
Stack Overflow
20 popped from stack.
10 popped from stack
Stack Underflow.

→

Q.2

Implementation of QUEUE

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

struct queued
    int* array;
    int left;
    int right;
    unsigned int capacity;
};
typedef struct queue Queue;
Queue* create_queue(unsigned int capacity){
    Queue* q = (Queue*) malloc(sizeof(Queue));
    q->array = (int*)malloc(sizeof(int) * capacity);
    q->capacity = capacity;
    q->left = 0;
    q->right = 0;
    return q;
}
```

```
void enqueue(Queue* q, int data){
    if(q->right >= q->capacity){
        printf("Queue is full\n");
        return;
    }
    q->array[q->right] = data;
    q->right++;
    printf("%d enqueued to queue\n", data);
}
int dequeue(Queue* q){
    if(q->left >= q->right){
        printf("Queue is empty\n");
        return INT_MIN;
    }
    int temp = q->array[q->left];
    for(int i=0; i<q->right-1; i++){
        q->array[i] = q->array[i+1];
    }
    q->right--;
    printf("%d dequeued from queue\n", temp);
    return temp;
}
```

→

```
void main(){
    Queue* q = create_queue(2);
    enqueue(q, 10);
    enqueue(q, 20);
    enqueue(q, 30);
    dequeue(q);
    dequeue(q);
    dequeue(q);
}
```

Console

↳

```
10 enqueued to queue.
20 enqueued to queue
Queue is full.
10 dequeued from queue
20 dequeued from queue
Queue is empty.
```

Q3

Implementation of Bubble Soart

```
#include <stdio.h>

void swap(int* num1, int* num2){
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}

int* bubble_soart(int* arr, int len){
    for(int i=0; i<len; i++){
        int swapped = -1;
        for(int j=0; j<len-1; j++){
            if(arr[j] > arr[j+1]){
                swap(&arr[j], &arr[j+1]);
                swapped = 1;
            }
        }
        if(swapped == -1){
            break;
        }
    }
    return arr;
}
```

```
void main (){
    int arr[] = {3, 44, 38, 5, 44};
    int len = sizeof(arr) / sizeof(arr[0]);
    int* sorted_arr = bubble_sort(arr, len);
    for (int i=0 ; i<len ; i++){
        printf(" %d", sorted_arr[i]);
    }
    printf("\n");
```

Console

→ 3 5 38 44 44

→

Q.9

Implementation of Selection Sort

```
#include <stdio.h>
void swap (int* num1, int* num2){
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
int* selection_sort (int* arr, int len){
    for (int i=0; i<len; i++){
        int lowest_index = i;
        for (int j = i+1; j<len; j++){
            if (arr[j]<arr[lowest_index]){
                swap (&arr[j], &arr[lowest_index]);
            }
        }
    }
    return arr;
}
```

```
void main(){
    int arr[] = {3, 44, 38, 5, 44};
    int len = sizeof(arr)/sizeof(arr[0]);
    int* sorted_arr = selection_sort(arr, len);
    for(int i=0; i<len; i++){
        printf("%d", sorted_arr[i]);
    }
    printf("\n");
}
```

Console

→ 3 5 38 44 44

Q.5

Implementation of Insertion Sort

```
#include <stdio.h>
void swap(int* num1, int* num2);
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}

int* insertion_sort(int* arr, int len){
    for(int i=1 ; i<len; i++){
        int key = arr[i];
        for(int j=0 ; j<=i; j++){
            if(arr[j] > arr[i]){
                swap(&arr[i], &arr[j]);
            }
        }
    }
    return arr;
}
```

```
void main(){
    int arr[] = {3, 44, 38, 5, 44};
    int len = sizeof(arr) / sizeof(arr[0]);
    int* sorted_ar = insertion_sort(arr, len);
    for(int i=0 ; i<len; i++){
        printf(" %d", sorted_ar[i]);
    }
    printf("\n");
}
```

Console

↳ 3 5 38 44 44