Data structures and Algorithms

Assignment-4

K. Sai Abhinav
CSE-F
AP19110010339

1)

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node * next;
};
struct Node * insert(struct Node *head, int value,
                     size_t position);
struct Node* deleteNode(struct Node* head,
                        int position);
void print_list(struct Node * head);
int main(int argc, char *argv[]){
{ int num, n1, d1;
    struct Node *head = NULL;
    head = insert(head, 1, 0);
    head = insert(head, 100, 1);
    head = insert(head, 21, 2);
    head = insert(head, 2, 3);
    head = insert(head, 5, 4);
    printf("Enter the nth position to insert: ");
    scanf("%d", &num);
```

```
printf ("Enter the data: ");
scanf ("%d", &n1);
head = insert (head, n1, num-1);
print_list (head);
printf("Enter the Kth position you need to delete:");
scanf ("%d", &d1);
deleteNode(head, d1);
print_list (head);
return 0;
}
struct Node* insert (struct Node* head, int value,
size_t position) {
    size_t i = 0;
    struct Node *currentNode;
    currentNode = malloc (sizeof *currentNode);
    currentNode -> data = value;
    struct Node ** nextForPosition = &head;
    for (i=0; i< position && *nextforposition != NULL
                                        ; i++)
{
    nextforposition = &(*nextforposition) -> next;
    currentNode -> next = *nextforposition;
    *nextforposition = currentNode;
    return head;
}
```

```
struct Node* deleteNode(struct Node *head,
                        int position) {
    struct Node *temp;
    struct Node* p = head;
    int count = 0;
    if (p == NULL) {
        printf("Linked list is empty");
    }
    if (position == 0){
        head = p->next;
        free(p);
    }
    else {
        while (count < position-1 && p != NULL) {
            p = p->next;
        }
        temp = p->next;
        p->next = temp->next;
        free(temp);
        head = p;
    }
}
void printlist(struct Node *head) {
    struct Node* i = head;
    while (i != NULL) {
        printf("%d\n", i->data);
        i = i->next;
    }
}
```

2)

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{ int data;
  struct Node *next;
};
void push(struct Node **head_ref,
                    int new_data)
{
  struct Node* new_node =
      (struct Node*) malloc (sizeof(struct Node));
  new_node->data = new_date;
  new_node->next = (*head_ref);
  (*head_ref) = new_node;
}
void printList(struct Node *head)
{
  struct node *temp = head;
  while(temp != NULL)
  { printf("%d", temp->data);
    temp = temp->next;
  }
  printf("\n");
}
```

```
void merge(struct Node *p, struct Node **q)
{
    struct Node *p_curr = p, *q_curr = *q;
    struct Node *p_next = p, *q_next;
    while (p_curr != NULL && q_curr != NULL)
    {
        p_next = p_curr->next;
        q_next = q_curr->next;
        q_curr->next = p_next;
        p_curr->next = q_curr;
        p_curr = p_next;
        q_curr = q_next;
    }
    *q = q_curr;
}
int main()
{
    struct Node *p = NULL, *q = NULL;
    push(&p, 3);
    push(&p, 2);
    push(&p, 1);
    printf("First linked list is:\n");
    printList(p);
    push(&q, 6);
    push(&q, 5);
    push(&q, 4);
    printf("second linked list is:\n");
    printList(p);
    merge(p, &q);
    printList(p);
    return 0;
}
```

3) #include<stdio.h>
void print(int arr[], int i, int j)
{
printf("[%d...%d]--{", i, j);
for(int k=i; k<=j; k++) {
printf("%d", arr[k]);
}
printf("}\n");
}
void findsubarrays(int arr[], int n, int sum)
{ for(int i=0; i<n; i++)
{ int sum_so_far=0;
for(int j=i; j<n; j++)
{
sum_so_far += arr[j];
if(sum_so_far == sum) {
print(arr, i, j);
}
}
}
}
int main()
{ int arr[] = {3, 4, -7, 1, 3, 3, 1, -4};
int sum;
printf("Enter K value: ");
scanf("%d", &sum);
int n = sizeof(arr)/sizeof(arr[0]);
findsubarrays(arr, n, sum);
return 0;
}

4) #include<stdio.h>
#include<stdlib.h>
#define SIZE 5
int queue[SIZE];
int front, rear = 0;
int insert(int);
void display();
void main()
{
while(1) {
int ch, val;
printf("1. Insert\n 2. display\n 3. exit\n");
printf("Enter the choice");
scanf("%d", &ch);
switch(ch)
{
case 1: printf("Enter the value to insert:");
scanf("%d", &val);
insert(val);
break;
case 2: display();
break;
case 3: exit(0);
break;
default: printf("wrong selection. Try again");
}}
}

```
int insert(int val)
{
   if(rear==SIZE)
   {
      printf("Queue is full");
   }
   else {
      queue[rear]=val;
          rear++;
   }
}
void display()
{  int i;
   if(rear==front)
   {  printf("Queue is empty");
   }
   else
   {
      printf("Reverse order is : ");
      for(i=rear-1;i>=front;i--)
      {  printf("%d", queue[i]);
      }
      printf("\n");
   }
   printf("Alternate order is : ");
   for(i=front;i<=rear-1;i++)
   {  if(i%2==0)
      {
         printf("%d",queue[i]);
      }
   }
```

```
for (i = front; i <= rear-1; i++)
{
    if (i != 0)
    {
        printf("%d", queue[i]);
    }
}
printf("\n");
}
```

5) (1)

a) An array is the data structure that contains a collection of similar data type elements whereas the linked list is considered as non primitive data structures contains a collection of unordered linked elements known as nodes.

b) In an array, memory is assigned during compile time while in a linked list it is allocated during run time.

c) In addition memory utilization is inefficient in the array, memory utilization is efficient in linked list.

(ii)

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

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

```
void merge(struct Node **p, struct Node **q)
{
    struct Node *p_curr = *p, *q_curr = *q;
    struct Node *p_next, *q_next;
    struct Node *p_curr1;
    p_curr1 = q_curr;
    p_curr = q_curr->next;
    p_curr1->next = p_curr;
    p_curr = p_curr1;
    *p = p_curr;
    *q = q_curr;
}
int main()
{
    struct Node *p = NULL, *q = NULL;
    push(&p, 3);
    push(&p, 2);
    push(&p, 1);
    printf("First linked list is \n");
    printList(p);
    push(&q, 6);
    push(&q, 5);
    push(&q, 4);
    printf("Second linked list is \n");
    printList(q);
```

```
merge(&p, &q);
printf("List 1 is : \n");
printList(p);
printf("\nList 2 is : \n");
printList(q);
getchar();
return 0;
}
```