Chandra Mahesh
AP19110010149

1) Write a program insert and delete an element at the nth and kth position in a linked list where n and k is taken as user.

A)

```
# include <stdio.h>
# include <stdlib.h>
struct node {
int data ;
Struct Node* next;
};
Struct Node* next;
};
Struct Node* next;
void insert (int data, int n) {
node* temp = new node;
temp -> data = data;
temp -> next = Null ;
if (n==1) {
temp -> next = head;
return ;
}
Void delete (int k) {
Struct node* temp = head
```

```
if (k==1) {
  head = temp->next;
  free (temp);
  return;
}
Node* temp=head;
 for(int i=0, i<n-2, i++) {
 temp = temp -> next
  temp ->next =temp;
 }
   void print( );
 for(int i=0; i<k-2; i++)
     temp= temp->next;
    tree (temp);
}
int main ( ){
int n,x,t.
 head=Null;
Printf("Enter the position for inserting:");
Scan f("%d",&n);
 Scanf ("%d",&x);
Insert (x,n);
```

```
Printf ("Enter the Position to delete")
Scanf ("%d", &k)
 delete (k);
 Printf(x);
 return;
}
```

2) Construct a new linked list by merging alternate nodes of two lists for example in list 1 {1 2 3} and list2 {4,5,6} in new list we should have {1 4 2 5 3 6}

```
# include <stdio.h>
# include <stdlib.h>
 Struct node {
 int data;
 Struct node* next;
};
Void Print List (Struct node* head)
 {
 Printf(" %d → ", (ptr → data));
    ptr = ptr → next;
    Printf ("Null \n");
}
void push (Struct node* head ; int data)
```

```
{
Struct node* new = (struct node* )malloc
                   (size of (struct node));
    new->data = data;
    new->next=*head;
    *head = new;
    }
Struct node* merge (struct node*a, struct node*b)
    {
    Struct node fake;
    Struct node* tail= fake;
      fake. next=null;
      while (1) {
      if (a==null)
      {
        tail->next=b;
        break;
      }
      else if (b=null)
      }
        else
        {
        tail->next =a ;
        tail=a
        a: a—> next;
        tail -> next=b;
        }
    }
```

```
return
}
void main()
{
int keys[] = {1,2,3,4,5,6,7}
int n = size of (key) / sizeof keys [0]
struct node * a = null ; *b = null;
for (int i = n-i ; i>0, i=i-a)
  Push (&a, keys[i]),
for (int i = n-2, i>=0, i = i-2)
    Push (&b; keys [j ];
  struct node * head = merge (a,b);
  print list (head);
}
```

3) Find all the elements in the stack whose sum is equal to k.

A)
```
# include <stdio.h>
void find (int arr[], int a, int k) {
    int total=0
    int x=0, y=0;
  for (x=0; x<a, x++) {
  while (total<k, && y<a)
    total = arr[y]
      y++;
    if (total==0)
    {
     Print ("find");
    return;
     total = -arr[x];
  }
}
int main (void){
int arr[ ]={9, 10, 12, 4, 1, 2, 3}
int k= 565;
int a= size of (arr) / size of (arr[0]);
find (arr, a, k);
return 0;
}
```

4

A)
```
# include <stdio.h>
# define size 20
void insert (int);
void delete ( );
int queue [20], a=-1, b=-1;
void ( ) {
int num ; choice;
while (1) {
Printf ("\n Nau\n");
Printf ("1. insert \n 2. delete\n. 3. Print \n,
        4. Reverse\n 5 Alternate \n 6. Exit");
Printf ("\n Enter your choice");
Scanf ("%od", &choice);
switch (choice) {
case 1 : Printf ("Enter num to insert:");
         Scanf ("%od", &num);
         insert (num);
         break;
case 2 : Printf ("Reverse queue");
         for (int i=size, i>0> i--)
          if (queue [i]==0)
```

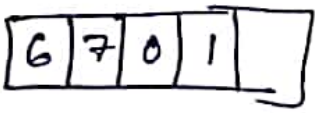
```
    continue;
Printf ("%d", queue[i]);
}
break;
case 3:
  Printf ("Alternate elements");
      for (int i=0, i<size, i>0, i+=2)
  {
    if [queue[i]==0;)
    continue
  Printf ("%d", queue[i]);
}
  break;
  return 0;
};
```

5) i) Array vs linked lists

1) Both are the data structure .. Both are used to store the data.

2) Cost of accessing the elements

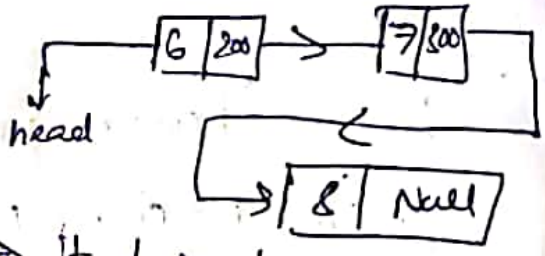
Arrays



⇒ it takes at constant time

O(1)

linked list



⇒ It depends on number of nodes in the linked list

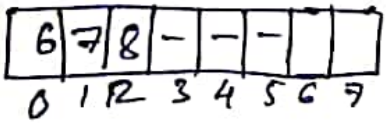
O(n)

3) Memory requirment and utilization.

Array

⇒ Ineffective in memory utilization

Ex:



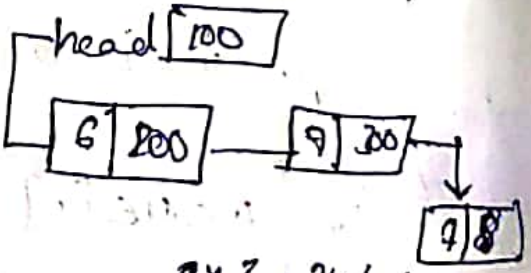
8 × 4 = 32 bytes

Used = 12

⇒ Require memory in less

linked list

⇒ it is in dynamic byte



8 × 3 = 24 byte

⇒ More requirements

4) Cost of insertion and cost of deletion

	Array	Linked list
Beginning	O(n)	O(1)
At end	O(1)	O(n)
ith position	O(n)	O(n)

5. Easy use and operations

Array	Linked list
→ easier to use	→ less easier
→ linear and binary	→ linear

(ii)
```
#include <stdio.h>
#include <stdlib.h>
int len[int fa()]
{
int i=0, x, y=0
  while(1)
  {
    if(x[i])
    {
      xy++, i++;
    }
  else
    {
      break;
    }
```


Scanned with CamScanner

```
}
return xy;
}
void changelist (int x[], int a[])
{
    for (int i=len(x)-1, i>=0, i--)
    {
        x[i+1] = x[i]
    }
    x[0] = a[0];
    printf("\n Elements of old array: \n");
for (int i=0; i<len(x); i++)
    { printf("%d", x[i]);
    }
    for (int i=0, i<len(y); i++)
    {
        y[i] = y[i+1];
    }
    printf("\n Elements of newarray: \n")
    for (int i=0; i<len(a); i++)
{
printf("%d", a[i]);
}
int main()
```

```
{ int x[10] = {1,2,3}, a[10] = {4,5,6};
  Change list = (a,b);
}
```


Scanned with CamScanner