Binary Search

```
1. Start
2. Read the Array size
3. Loop (i<n)
4. Read the array elements
5. Loop ends
6. Loop (i<n)
    a. Loop (j<n-1)
        i. If (a[i]>a[j+1])
        ii. Set temp as a[i]
        iii. Set a[i] as a[j+1]
        iv. set a[j+1] as temp
    b. Loop ends
7. Loop ends
8. Read the element(key) to be searched
9. Loop (begin <end)
10.Compare key with the middle element(begin+end/2).
11.If key matches with middle element, print "Element found"
12.Else If key is greater than the mid element, then x can only lie in right
   half subarray after the mid element. So we search in  right half.
13.Else (x is smaller) search in left half.
14.If key not found print "Element not present"
15.If ends
16.Stop
```

BUBBLE SORT

```
1. Start
2. Read array size(n)
3. Read array elements by  loop(i<n)
4. Loop ends
5. Loop (i<n)
    a. loop(j<n-1)
        i. If (a[i]>a[j+1])
        ii. Set temp as a[i]
        iii. Set a[j] as a[j+1]
        iv. Set a[j+1] as temp
        v. If ends
    b. Loop end
6. Loop ends
7. Print sorted array(i<n)
8. Loop ends
9. Stop
```

SELECTION SORT

```
1. Start
2. Read the size of array(n)
3. Read the array elements (i<n)
4. Set i = 0
5. loop(i< n-1)
      a. loop(j<n)

            i.   temp=a[i]
           ii.   a[i]=a[j]
          iii.   a[j]=temp
      b. Loop ends
6. Loop ends
7. Print sorted array(i<n)
8. stop
```

INSERTION SORT

```
1. Start
2. Read array size(n)
3. Read array elements(i<n)
4. Set i = 1
5. loop(i<n)
      a. temp=a[i];
      b. j=i-1;
      c. while(temp<a[j]&&j>=0)
               a[j+1]=a[j];
               J--;
      d. While loop ends
6. A[j+1] = temp
7. Loop ends
8. Print sorted array(i<n)
9. Stop
```

STACK

```
1. Start
2. Read size of stack
3. Set top = -1
4. if(push)
```

```
        a.  if(top = size -1
            Print overflow
        b.  else
            Read element
            Top++;
            Stack[top] = element
            Print the entered element
5.  if(pop)
        a.  if(top == -1)
             Print underflow
        b.  else
            Print deleted element
            Top --
6.  if(display)
        a.  if(top = -1)
            Print underflow
        b.  Else
            Print stack [ for(i=top;i>=0;i--) ]
7.  Stop
```

QUEUE

```
1.  Start
2.  Set front and rear to -1
3.  Read size of queue
4.  if (insertion)
        a.  if(rear = size -1)
             Print overflow
        b.  Else
            Read the entered element
            if (front == -1 and rear == -1)
            Front = 0
            Rear ++
            queue [rear] = element
      5.if (deletion)
        a.  if (front = -1)
             Print underflow
        b.  Else
            Delete queue[front]
        c.  if(front= =rear)
             Front = rear =-1
        d.  Else front ++
    6. If(Display)
                a.  if (front == -1 and rear == -1
                    Print underflow
                b.  Else
                    loop(i<rear)
                    Print queue[i]
            7. Stop
```

Circular queue

```
1.  Start
2.  Set front , rear = 0
3.  Read size of queue
4.  Display menu to to select from insert/remove/display/exit
5.  If (insertion)
        a.  if(rear-front = size)
            Print queue is full
        b.  Else
            Print enter the element
            f(front== 1 and rear ==1)
            Front = rear = 0
            Scan queue[rear]
        c.  Else if(front>0 and rear == size)
            Front --
            Scan queue[front]
        d.  Else
            Scan queue[rear]
            Rear ++
    5. If (deletion)
        a.  if(front == -1 and rear == -1)
            Print empty queue
        b.  else
        c.  Print deleted element queue[front++]
                    If (front == rear)
                    Print queue is empty
                    Front = rear = -1
                    Else
                    display()
    6. If (display)
        a.  loop(i=front;i<rear;i++)
        b.  Print queue[i]
    7. Stop
```

Stack using linked list

```
1. Start
2. Set NODE *start = NULL and *top = NULL
3. Display menu to choose from insert , delete or display
4. If(display)
        Ptr = (NODE*)malloc(sizeof(NODE))
        Read element , item
        Ptr->data=item
        ptr->link=NULL
        if(start == NULL)
        Start = ptr
        Top =ptr
        Else
        ptr->link=top
        Top = ptr
5. If (delete)
        if(top == null)
        Print stack is empty
        Else
        S = top
        Print deleted element (top->data)
        top = top->link
        free(s)
        if(top = NULL)
        Start =(NULL)
6. if(display)
        if(top ==null)
        Print empty stack
        Else
        print stack elements are
        Display = top
        Loop while(disply != NULL)
        Item = display->data
        Print item
        Display = display-> link
7. Stop
```

##

For queue using linked list replace all start as rear && top as front

##

Polynomial Addition

1. Start
2. Set *p1 =NULL , *p2 = NULL, *p = NULL
3. Read coefficients and exponents of 2 polynomials
4. Compare exponents from first node

+++