Name ≠ Farhan Shah

ID ≠ 13180

Semester ≠ 8th

Subject ≠ Data Structue Alogorithm

Teacher ≠ Dr. Naeem Jan

Date ≠ 25/9/2020

Scanned with CamScanner

Q1:- Explain the following operation in a single link list.

1) Insert an Element
2) Delete an Element

Answer:-

Now that you have got an understanding of the basic concepts behind linked list and their types its time to dive into the common operations that can be performed.

Two important points to remember.

⇒ (head) points to the first node of the linked list

⇒ pointer of the last node is (Null) so if the next current node is (Null) we have reached the end of the linked list.

In all of the example. we will assume that the linked list has three node 1 -->2 --3 with node structure as below.

```
struct node
{
  int data;
  struct node * next
};
```

Scanned with CamScanner

⇒ How To Traverse a linked list :-

When temp (Null) we know that we have reached the end of the linked list so we get out of the while loop.

```
Struct node * temp = head;
printf ("\n\nList Element are -\n");
 while (tem ! = Null)
 {
    printf ("%d -->, temp ->data);
    temp = temp -> next;
    }
```

Output:

List Element are

1 -->2 -->3 -->

⇒ How to Add Element to a linked list

⇒ Add to the begining:

① Allocate memory for new node
② store data
③ change next of new to point to head
④ change head to point to recently created node

```
Struct node * newNode;
NewNode = malloc (size of ( Struct node));
Newnode -> data = 4;
newnode -> next = head;
head = newNode;
```

Scanned with CamScanner

⇒ Add to the End:-

① Allocate memory for new node
② Store data
③ Traverse to last node
④ Change next of last node to recently created node.

```
Struct nod * newnode;
new node = malloc (Size of (Struct node));
newnode -> data = 4;
newnode -> next = Null;
Struct node * temp = head;
while (tem -> next != Null) {
  temp = temp -> Next;
  }
tem -> Next = new node;
```

→ Add to the Middle

① Allcate memory and store data for new node
② Transverse to node just before the required position of new node
③ change next pointer to include new in b/w.

```
Struct node * newNode;
newNode = malloc (Size of (Struct nod));
new node -> data = 4;
   Structure node * temp = head;
for (int i=2; i < position; i++) {
   If (tem -> next != Null) {
      temp = temp -> next
   }
}
```

Scanned with CamScanner

```
newnode -> next = temp -> next
temp -> next = newNode;
```

⇒ How to Delete:-

⇒ Delete from beginning

② Point head to the second node

```
head = head -> next;
```

⇒ Delete from end:

① Traverse to second last element

② change its next pointer to null.

```
struct node* temp = head;
while (temp -> next -> next != Null) {
    temp = temp -> next;
}
temp -> next = Null
```

⇒ Delete from middle:-

① Traverse to element before the element to be deleted

② change next pointer to exclude the node from the chain.

```
for (int i=2; i<position; i++) {
    if (temp -> next != Null) {
        temp = temp -> next
    }
}
temp -> next = temp -> next -> next;
```

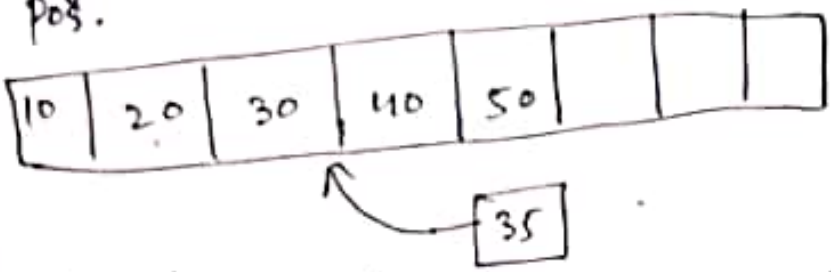
Scanned with CamScanner

Q2: Write a programe to Insert a new element in the given unsorted array at Kth position

Answer:

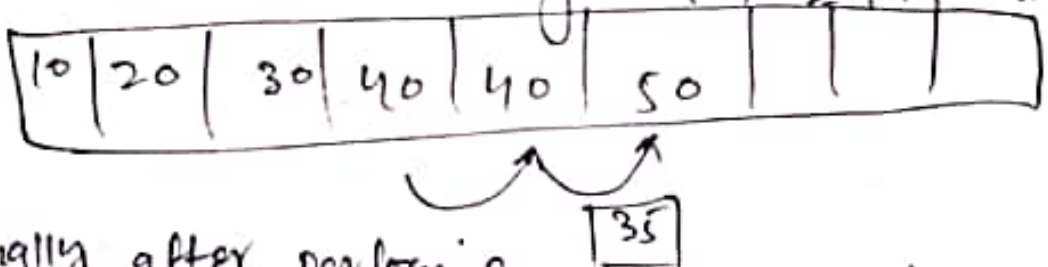
(1) Input size and elements in array. Store it in some variable say size and arr.

(2) Input new element and position to Insert in array. Store it in some variable say num and pos.

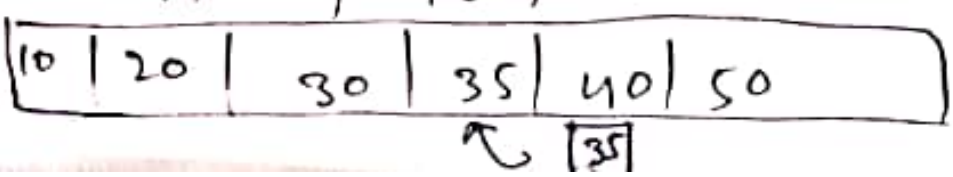


(3) To Insert new element in array shift element from the given position to one position right Hence run a loop in desending order from size to pos to insert. The loop structre should look like for (i=size; i>=pos; i--)

Inside the loop copy presvious element to current element by arr[i] = arr[i-1];.



(4) finally after performing shift operation. copy the new element at its specified position i.e arr[pos-1] = num;



Scanned with CamScanner

⇒ program to insert element in array:

```
/**
 * c program
*/
#include <stdio.h>
# define Max_size 100

int main()
{
   int arr[Max_size];
   int i, size, num, pos;
   /* input size of the array*/
   printf("Enter size of the array: ");
      scanf("%d", &size);

   /* input element in array*/
   printf("Enter element in array: ");
      for(i=0; i<size ; i++)
      { scanf("%d", &arr[i]);
   /* input new elent and position to insert*/
   printf("Enter element to insert: ");
      scanf("%d", &num);
   printf("Enter the element position: ");
     scanf("%d", &pos);
   /* if position of element is not valid*/
   if(pos > size+1 || pos <= 0)
   {
     printf("Invalide position. Please enter
   position b/w 1 to %d"; siz ); }
      else
```

Scanned with CamScanner

```
/* Make room for new array element
   by shifting to right */
   for (i= size ; i>= pos; i--)
    {
      arr[i] =arr [i-1];
    }
/* Insert new element at given position and
   increment siz */ arr [pos-1] =num;
     size++;
   /* print array after insert aperation */
    print ("Array element after Insertion:");
      { for (i=0; i< size ; i++)
        printf("% d/t", arr[i]);
      }
    } return 0;
  }
```

Q3:- Answer:

Quick Sort:- Quick Sort is a fast Sorting algorithm used to sort a list of elements. Quick sort algorithm is Invented C.A.R Hoare.

The quick sort algorithm attempts to separate the list of element into two parts and then sort each part recursively. that means

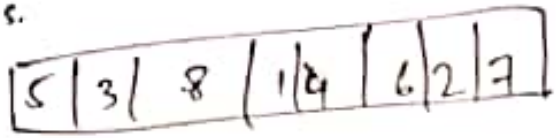
Scanned with CamScanner

it use divid and conquer strategy. In quick sort the position partition of the lis is performed based on the element called pivot. Here pivot element. is one of the elements in the list.

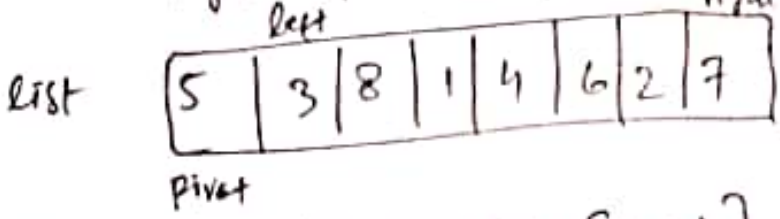
The list is divided into two partition such that "all element to the left are smaller than the pivot and all elements the to the right of pivot are greater than or equal to the pivot.

Example

consider the following unsorted list elements.



Define pivot, left and right. Set pivot = 0 left = 1 and right = 7 Here 7 indicate size - 1



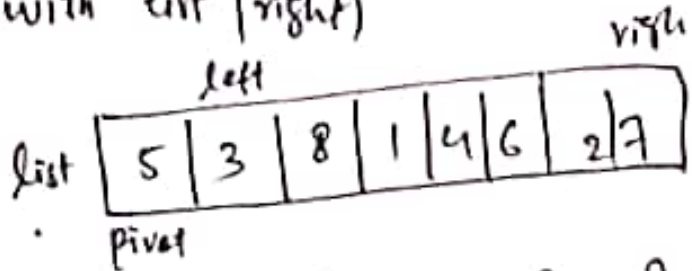
compare list [left] with list [pivot]. If list left is greater than list [pivot] then stop. left otherwise move left to the next

compare list [right] with list [pivot]. If list [right] is smaller than list [pivot] Then stop right otherwise move right to the previous.

Scanned with CamScanner

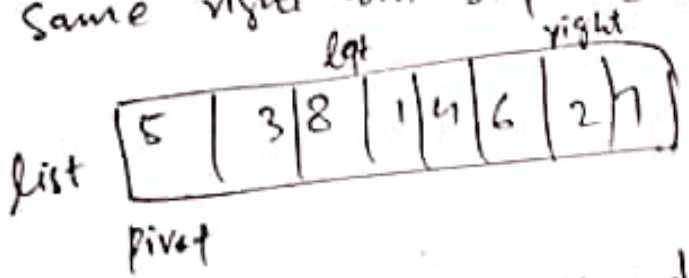
Repeat the same until left >= right

If both left & right are stoped but left < right then swap list[left] with list(right) and continue the process if left >= right the swap list[pivot] with list[right]

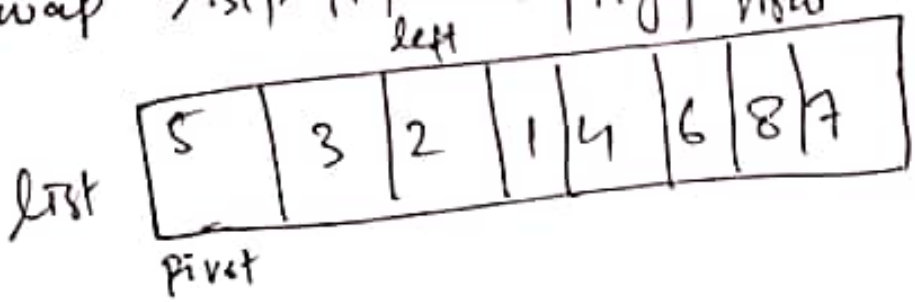


compare list[left] < list[pivot] as it is true increment left by one and repeat the same left will stop at 8.

compare list[right] > list[pivot] as it is true decrement right by one and repeat the same right will stop at 2

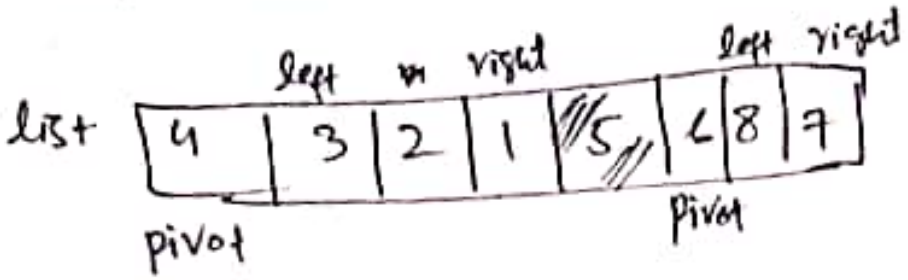


Here left & right both are stoped and left is not greater than right. So we need to swap list[left] and [right]

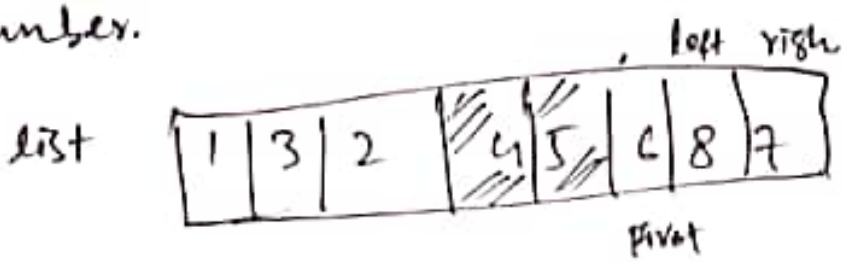


Scanned with CamScanner

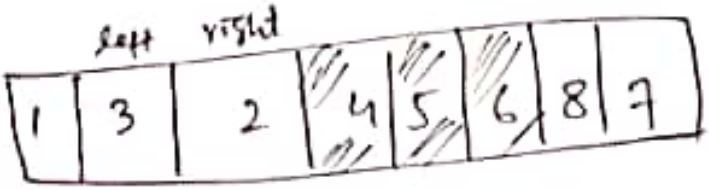
and then



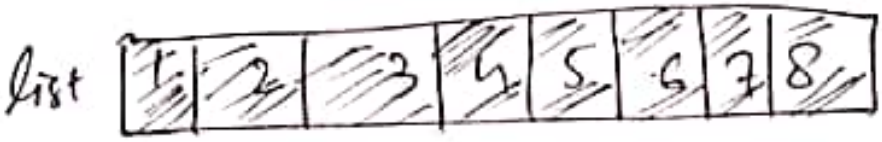
In the left sublist as there are no smaller number than the pivot left will keep on moving to the neat and stope at last number.



and then



Repeat the same recursively on both left and right sublists until all the number are stored The final Sorted list will be as follows.



Scanned with CamScanner

⇒ Insertion Sort Algorithm:

a: array of items
n: size of list

Start
1. declare variable - i, key, j
2. loop: i = 2 to n-1 // outer loop
   2.1 key = a[i];
   2.2 j = i - 1;
   2.3 loop: (j >= 0 && arr[j] > key) // inner loop
   2.3.1 arr[j+1] = arr[j];
   2.3.2 j = j - 2;
   end loop // inner loop
   2.4 arr[j+1] = key;
   end loop // outer loop
   Stop.

Q4:-

Answers-

⇒ part(A) Adjecency matrix:-

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	0	0
C	0	1	0	1	1
D	1	0	1	0	1
E	0	0	1	1	0

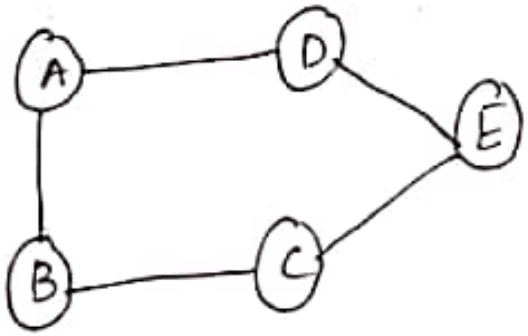
1 = true, 0 = false

Scanned with CamScanner

Need of $O(n^2)$ space to store the graph

with sparse graphs (graphs that do not have too many edges), there may be "too many" ~~size~~ zeros in the matrix Dense graphs have many more edges.

⇒ Part (B)



Scanned with CamScanner