DS ASSIGNMENT:

Write a comparison table of all time complexities of all sorting algorithms?

1. The efficiency of an algorithm depends on two parameters. They are:-

1. Time complexity
2. Space complexity.

1. Time Complexity:- Time complexity is defined as the number of times a particular instruction set is executed rather than the total time taken. It is because the total time took also depends on some external factors like the compiler used, Processor's speed etc.

2. Space Complexity:- Space complexity is the total memory space required by the program for its execution.

Both are calculated as the function of input size(n). One important thing here is that inspite of these parameters the efficiency of an algorithm also depends upon the nature and size of the input.

Types of Time Complexity:-

1. Best Time Complexity:- Define the input for which algorithm takes less time (or) minimum time. In the best case, calculate the lower bound of an algorithm.

Example:- In the linear search, when search data is present at the first location of large data then the best case occurs.

2. Average Time Complexity:- In the average case take all random inputs and calculate the computation time for all inputs.

And we divide it by the total number of inputs.

3. Worst Time Complexity:- Defines the input for which algorithm takes (place) a long time (or) maximum time. In the worst calculate the upper bound of an algorithm.

Example:- In the linear Search, when search data is present at the last location

Of large data then the worst case occurs.

The below table compares the time complexities and Space complexities of all algorithms.

S.NO.	Algorithm	Time Complexity			Space Complexity (Worst).
		Best	Average	Worst	
1.	Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$.
2.	Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$.
3.	Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$.
4.	Heap Sort	$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n\log(n))$	$O(1)$.
5.	Quick Sort	$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n^2)$	$O(n)$.
6.	Merge Sort	$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n\log(n))$	$O(n)$.
7.	Bucket Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$.
8.	Radix Sort	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$.
9.	Count Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	$O(k)$.
10.	Shell Sort	$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n^2)$	$O(1)$.
11.	Tim Sort	$\Omega(n)$	$\Theta(n\log(n))$	$O(n\log(n))$	$O(n)$.
12.	Tree Sort	$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n^2)$	$O(n)$.
13.	Cube Sort	$\Omega(n)$	$\Theta(n\log(n))$	$O(n\log(n))$	$O(n)$.

ite a 'c' program for multiplication of two polynomials using linked lists?

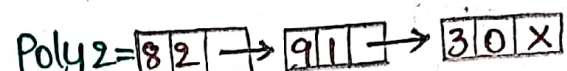
```
//vi.mulpoly.c
//17/4/23.
```

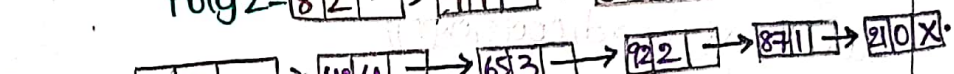
Example:- $Poly_1 = 2x^3+3x^2+4x+7=0$

$Poly_2 = 8x^2+9x+3=0$

$Poly_3 = 16x^5+18x^4+6x^3+24x^4+27x^3+12x+32x^3+36x^2+12x+56x^2+63x+21=0$

$= 16x^5+42x^4+65x^3+92x^2+87x+21=0$

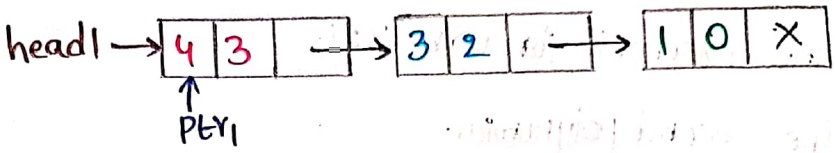




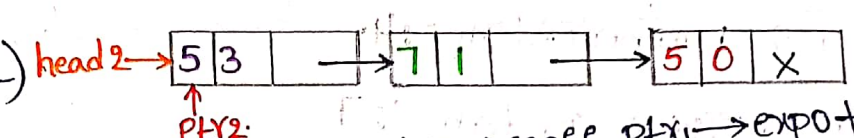

```
#include<stdio.h>
#include<stdlib.h>
Struct node{
        float coeff;
          int expo;
        Struct node *next;
 };
 Struct node * temp;
temp=(Struct node*)malloc(sizeof(Struct node));
      new->coeff=co;
      new->expo= ex;
      new->next=NULL;
if(head ==NULL || ex>head->expo) // If there is no node in the list.
 {
  new->next=head;        // Given exponent is greater than the first node exponent
   head=new;
 }
else
  {temp=head;
  while(temp->next !=NULL && temp->next->expo>=ex);
  temp=temp->next;
  new->next=temp->next;
  temp next=new;
 }
return head;
}
Struct node * create(struct node *head)
{int i,n;
```

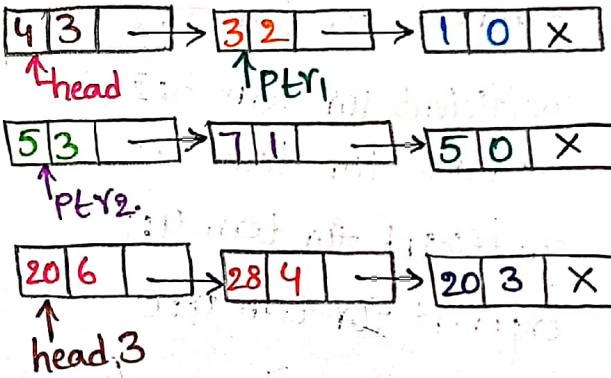
```
float coeff;
int expo;
Printf("Enter the number of terms of the polynomial: ");
Scanf("%d", &n);
for(i=0; i<n; i++)
{
Printf("Enter the co-efficient of %d:", i+1);
 Scanf("%f", &coeff);
Printf("Enter the exponent of %d : ", i+1);
Scanf("%d", &expo);
head=insert(head, coeff, expo);
}
return head;
}
Void print (struct node *head)
{
if(head==NULL)
 Printf("No polynomial exists");
 else
 {
 struct node *temp=head;
 while(temp!=NULL)
{
 Printf("(%.1f x^%d)", temp->coeff, temp->expo);
 temp=temp->next;
 if (temp!=NULL)
 {
  Printf("+");
 }
 else {
       Printf("\n");
      }
 }
}
}
Void mulpoly (struct node * head1, struct node * head2)
 {struct node * ptr1=head;
```

```
struct node *ptr2=head2;
struct node*head3=NULL;
if(head1==NULL||head2==NULL)// Check if first (or) Second polynomial is NULL.
{
 Printf("Zero polynomial\n");
  return;
}
```




```
while(ptr1!=NULL)
{
 while(ptr2!=NULL)
  {
   head3=insert(head3, ptr1->coeff * ptr2->coeff, ptr1->expo+ptr2->expo);
   ptr2=ptr2->next; // Multiplication of two polynomials.
  }
  ptr1=ptr1->next;
  ptr2=head2;
}
Print(head3);
}.
```






```
int main()
{ struct node *head1=NULL;
  struct node *head2=NULL;
Printf("Enter the first polynomial\n");
  head1=Create(head1);
Printf("Enter the second polynomial\n");
  head2=create(head2);
 mulpoly(head1,head2);
 return0;
}.
```

Output:- Enter the first polynomial.

Enter the number of terms:3

Enter the coefficient for term 1:4

Enter the exponent for term 1: 3
Enter the coefficient for term 2: 6
Enter the exponent for term 2: 6.
Enter the coefficient for term 3: 8
Enter the exponent for term 3: 1.

Enter the Second polynomial.

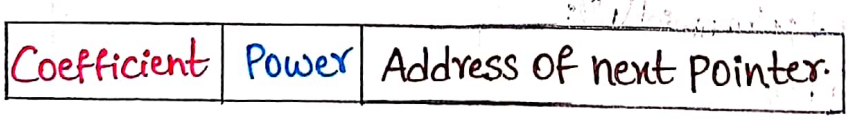
Enter the number of terms: 4.

Enter the coefficient for term 1: 7
Enter the exponent for term 1: 3
Enter the coefficient for term 2: 3
Enter the exponent for term 2: 2
Enter the coefficient for term 3: 2
Enter the exponent for term 3: 1
Enter the coefficient for term 4: 1
Enter the exponent for term 4: 0

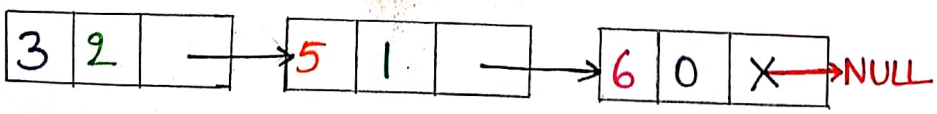
$42.0x^9+18.0x^8+12.0x^7+6.0x^6+28.0x^6+12.0x^5+8.0x^4+56.0x^4+4.0x^3+24.0x^3+16.0x^2+8.0x^1$

$=42.0x^9+18.0x^8+12.0x^7+34.0x^6+12.0x^5+64.0x^4+28.0x^3+16.0x^2+8.0x^1$

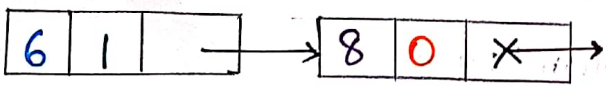
Node structure:-



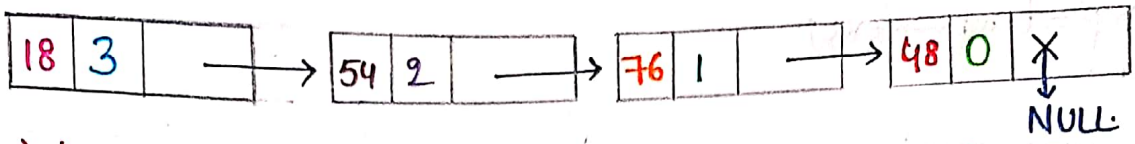
List-1:-



List-2:-



Resultant List:-



3. Write a 'C' Program to demonstrate priority queues using linked list?

A.

```
//vi. priorityqueue.c
//17/4/23.
#include<stdio.h>
#include<stdlib.h>
Struct node{
   int data;
   int info; Priority;
 Struct node *next;
};
Struct node * head = NULL;        //Function to push according to priority.
Void Enqueue (int data value, int priorityvalue)
{
 Struct node * new_node = NULL;
 new_node = (Struct node*) malloc (sizeof(struct node));
 new_node -> data = datavalue;
 new_node -> Priority = priority value;
 new_node -> next = NULL;
 if(head == NULL)                  //The head of list has lesser priority than new
 {                                   node. So, insert new node before head node
 head = new_node;                    and change head node.
 }
else if(head -> priority < new_node -> priority)
{ new_node -> next = head;          //Insert New node before head.
  head = new_node;
}
else                                //Traverse the list and find a position
 {                                  to insert the new node
  Struct node * temp = head;
  while(temp -> next != NULL && temp -> next -> priority > new_node -> Priority)
```

Priority-	4	0	2	1	3
Values-	54	96	87	69	72

```
{                    // Either at the ends of the list (or) at required position.
 temp = temp->next;
}
 new_node->next = temp->next;
 temp->next = new_node;
 }
}
Void Dequeue()
{struct node * temp=head;
 head=head->next;
 free (temp);
}
int peek()                    // Return the value at head.
{return head->data;
}
int main()
{
  Enqueue(5,1);               //Create a priority queue
  Enqueue(6,0);                  7->4->5->6.
  Enqueue(4,2);
  Enqueue(7,3);
While(head !=NULL)
{
  Printf("%d ", Peek());
   Dequeue();
 }
return 0;
}.
```

Enqueue.

Priority Value	3	0	2	1
DataValue	7	6	4	5

Linked list	Push()	POP()	Peek()
Time Complexity.	O(n)	O(1)	O(1).

Output:- 6 7 4 5 6.