4. (a) There a number is stored in the linked list. We know, if we want to add some number we have to start from ~~MSB~~ LSB and here LSB is in the end of list. It is hard to access the ~~t~~ one way list from Last.

So the approch is we will reverse the list, then we add 1 and calculated the result with carry through link list and Then again reverse the Link list to get our resultant number.

Code:

```
# include < bits/ stdc++.h>

using namespace std;

class node
{ public:
    int data;
    node* nxt;
};
node* head = NULL;
```

```
void rev()                    // code for reversing List
{
    node * prev = NULL;
    node * current = head;
    node * next;
   while (current != NULL)
   {
        next = current -> nxt;
        current -> nxt = prev;
        prev = current;
        current = next;
   }
   head = prev;
}
Void traverse()
{   node * ptr = head;
     while (ptr != NULL)
     {     cout << ptr->data;
     }  prev ptr = ptr -> nxt
     }
    cout << endl;
}
```

```
Void of add()
{
    rev() ;     // calling reverse function to reverse the
                   link list
    node* ptr = head -> nxt;
    int sum, carry = 0;
    sum = head -> data;
    head -> data >= (sum + 1) % 10.
    carry = (sum + 1) / 10;
    while (ptr != NULL) && carry != 0)
    {
        sum = ptr -> data + carry;
        ptr -> data = sum % 10;
        carry = sum/10;
        ptr = ptr -> nxt;
    }
    rev();
    if (carry != 0)
    { ptr = new node();
      ptr => data = carry;
      ptr -> nxt = head
      head = ptr
    }
}
```

```
int main()
{
    string s;
    cin >> s;          // taking a number into string
    head = new node();
    head -> data = s[0] - '0';

    node * prev = head;
   for(int i = 1 ; i < s.size() ; i++)
    { node * ptr = new node()    // storing the number
        ptr -> data = s[i] - '0';  // in link list.
        prev -> nxt = ptr;
        prev = ptr;
    }
   add();      // adding 1 to the list
   traverse(); // getting the result
}
```

4. (b) Here we apply insertion sort to the link list.

As we can not apply in back iterate from back to front in link list, we will find iterate from front. for every pivot element, we will find a right spot to insert from iterating from the front and will continue the insertion sort for next node

here is the code for it,

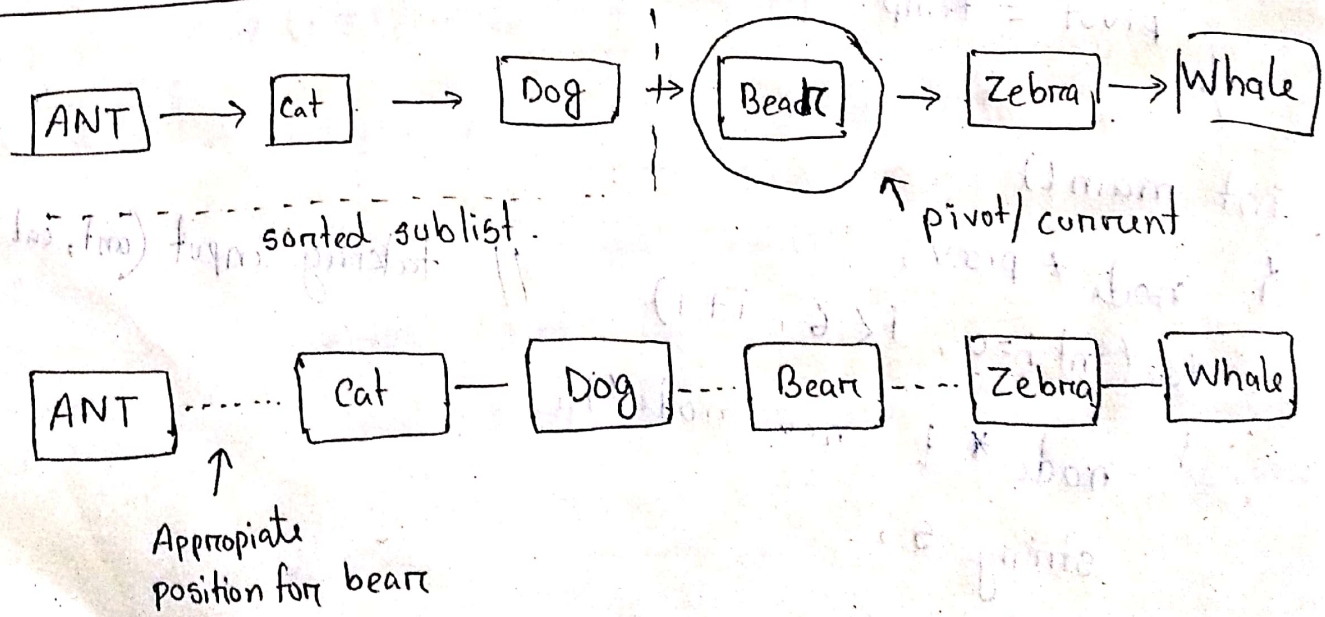
```
#include <bit/stdc++.h>
using namespace std;
class node
{ public:
      string data;
      node* nxt = NULL;
};
node* head = NULL;
void traverse ()
{  node* prev = head;
   while (prev != NULL)
   {  cout << prev -> data << " " << " ";
      prev = prev -> nxt;
   }
```

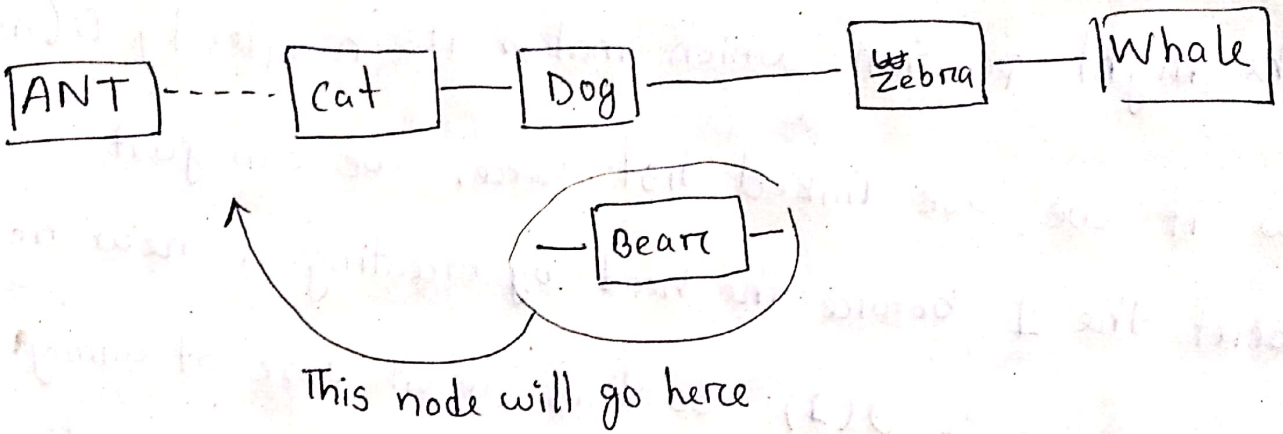
```
    cout << endl;
    }
    void insertion_sort()
{   if (head -> nxt == NULL || head == NULL)
        return;
    node * prev = head;
    node * pivot = head -> nxt
    while (pivot != NULL)    // traversing the whole List.
    {
        node * temp = pivot -> nxt;
        node * ptr = head;
        while (ptr != pivot)   // Loop for inserting pivot
        {
            if (ptr -> data >= pivot -> data)
            {
                prev -> nxt = pivot -> nxt;
                pivot -> nxt = head;
                head = pivot; // when data is less than first
                              element, the pivot is made the
                              first
                break;
            }
```

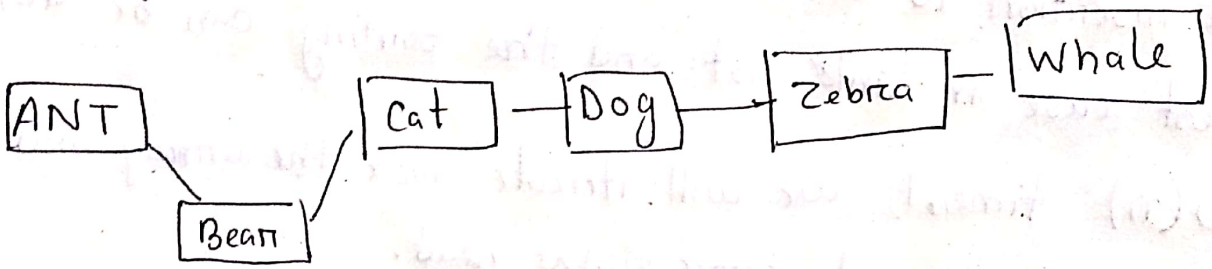
```
else if ( (ptr -> nxt -> data) > (pivot -> data))
{
    prev -> nxt = temp;          // code for middle insert.
    pivot -> nxt = ptr -> nxt;
    ptr -> nxt = pivot
    break;
}
ptr = ptr -> nxt;
}
if ( ptr == pivot) // this would be true when the sublist
{ prev = pivot;      is already sorted.
}
pivot = temp
}
int main()
{ node * prev;
for (int i=0, i< 6, i++)      // taking input (ant, cat .....)
{ node * t = new node();
  string s;
  cin>> s;
  t -> data = s;
  if (head == NULL)
    { head = t;}
```

```
else
  {
    prev -> next = t;
  }
  prev = t;
}
cout << "Before :" << endl;
traverse();
insertion_sort();
cout << "After :" << endl; // before and after sorting link list are printed.
traverse();
}
```

Illustration of the node 'Bear':







4. (b) (ii) When we apply insertion sort in array, we need n times to iterate over the array and another n times to ~~for~~ insert the data in array as data is insertion in array requires n swaps in worst case. This makes the worst case comlexity of insertion sort $O(n^2)$. But in link list we can insert in $O(1)$. So ~~th~~ in one sense it can be useful.

Let us consider the worst case of array when the array is totally reversed,

5 4 3 2 1.

in this array we need (n-1) number shift to insert is 1

in the right position. which makes its complexity $O(n^2)$

Now if we use linked list there, we can just insert the 1 before the head by creating a new node. Before insertion is $O(1)$. So this worst case of array become the best case in Link list and the sorting can be done in $O(n)$ times! we will iterate over the array and just insert a the node before the head.

However when the array is already sorted the Link list insertion sort will take $O(n^2)$ which is the worst case

But from the above explaination we could make a worst case, a best case by link list representation.