Assaignment – 4

Question 1

Write a function "insert_any()" for inserting a node at any given position of the linked list. Assume position starts at 0.

```
void insert_any(Node** current, int pos, int data)
{
if (pos < 1 || pos > size + 1)
cout << "Invalid position!" << endl;
else {
while (pos--) {
if (pos == 0) {
Node* temp = getNode(data);
temp->next = *current;
*current = temp;
}
else
current = &(*current)->next;
}
size++;
}
}
```

Question 2

Write a function "delete_beg()" for deleting a node from the beginning of the linked list.

```
Node* delete_beg(struct Node* head)
{
if (head == NULL)
return NULL;
Node* temp = head;
head = head->next;
delete temp;
return head;
}
```

Question 3

Write a function “delete_end()” for deleting a node from the end of the linked list.

```
Node* delete_end(struct Node* head)
{
if (head == NULL)
return NULL;
if (head->next == NULL) {
delete head;
return NULL;
}
Node* second_last = head;
while (second_last->next->next != NULL)
second_last = second_last->next;
delete (second_last->next);
second_last->next = NULL;
return head;
}
```

Question 4

In the Binary Search algorithm, it is suggested to calculate the mid as beg + (end - beg) / 2

instead of (beg + end) / 2. Why is it so?

Sol:

Because `beg + end` may overflow. Which then means you get a result that is less than `beg`. or far into the negative if you are using signed integers.

So instead they take the distance between `beg` and `end` and add half of that to beg. This is only a single extra operation to make the algorithm more robust.

Question 5

Write the algorithm/function for Ternary Search.

Sol:

In this algorithm, the given array is divided into three parts and the key (element to be searched) is compared to find the part in which it lies and that part is further divided into three parts.

Steps to perform Ternary Search:

- First, we compare the key with the element at mid1. If found equal, we return mid1.
- If not, then we compare the key with the element at mid2. If found equal, we return mid2.
- If not, then we check whether the key is less than the element at mid1. If yes, then recur to the first part.

- If not, then we check whether the key is greater than the element at mid2. If yes, then recur to the third part.
- If not, then we recur to the second (middle) part.