#1

```
Struct Node {
        void* data
        Node* next
}

Node(void* d) {
        this->data = d;
        this->next = null;
}

Bool isPalindrome(Node* head)
        Stack s = newStack()

        Node* node = head
        while(node)
        {
                s.push(node->data)
                Node = node->next
        }

        node = head;
        while(node)
        {
                Int top = s.top();
                s.pop();
                if(top != node->data)
                {
                        Return false;
                }
                Node = node->next
        }
        Return true;
}
```

2.

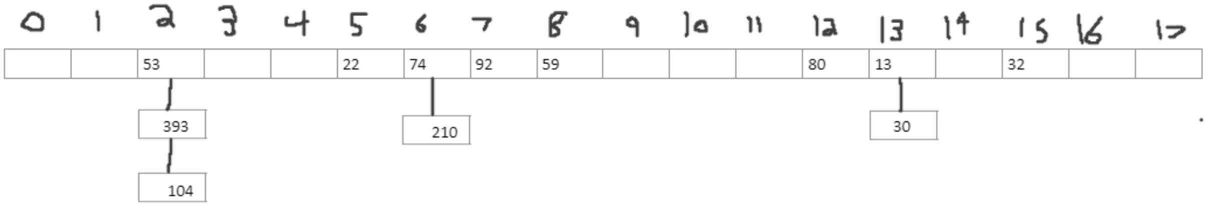
Problem 1: matching negative and positive values will come out as the same number after the hash function, causing a collision

Problem 2: the second problem is that the hash table to hold the values would have to be huge, as 25,000,000 slots is quite a few

A better hash function would be: f(x) = x + 1000

- This would resolve both problems as matching negative and positive numbers would come out as different values, and the second problem is resolved as the table would have 2000 slots, as opposed to 25 million slots

3.



4.
500 => 80
501 => 50
502 => 20
503 => 148
504 => 117
505 => 87

589 =>80

The value 589 comes out of the hash function with the same key as the value 500, causing a collision. This collision can be solved by creating the has table as an array of linked lists, and adding each collision onto the linked list at that key spot. So in this case, the value 589 would be stored at the same spot as 500, in the form of nodes in a linked list

5.

0	1	2	3	4	5	6	7	8	9	10	11	12
									35			

0	1	2	3	4	5	6	7	8	9	10	11	12
		41							35			

0	1	2	3	4	5	6	7	8	9	10	11	12
		41							35		50	

0	1	2	3	4	5	6	7	8	9	10	11	12
		41				9			35		50	

0	1	2	3	4	5	6	7	8	9	10	11	12
26		41				9			35		50	

0	1	2	3	4	5	6	7	8	9	10	11	12
26		41				9	54		35		50	

0	1	2	3	4	5	6	7	8	9	10	11	12
26	27	41				9	54		35		50	

0	1	2	3	4	5	6	7	8	9	10	11	12
26	27	41	0			9	54		35		50	

0	1	2	3	4	5	6	7	8	9	10	11	12
26	27	41	0	100		9	54		35		50	

0	1	2	3	4	5	6	7	8	9	10	11	12
26	27	41	0	100	200	9	54		35		50	