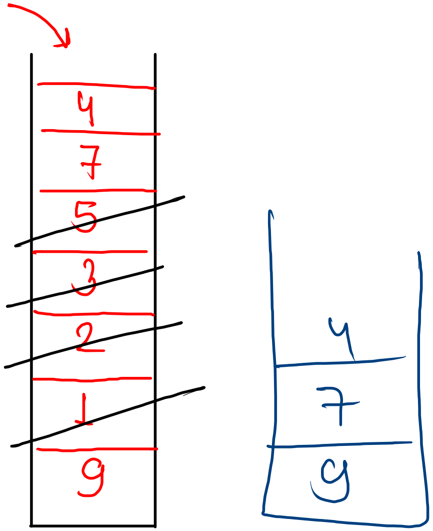
⇒ Nearest larger element on left side

	0	1	2	3	4	5	6
arr :-	9	1	2	3	5	7	4
	-1	9	9	9	9	9	7

	0	1	2	3	4
arr :-	7	2	8	6	9
	-1	7	-1	8	-1

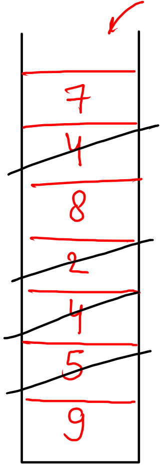
	0	1	2	3	4	5	6
arr :-	9	1	2	3	5	7	4
	-1	9	9	9	9	9	7

stack will store values



arr :- [9 5 4 2 8 4 7]

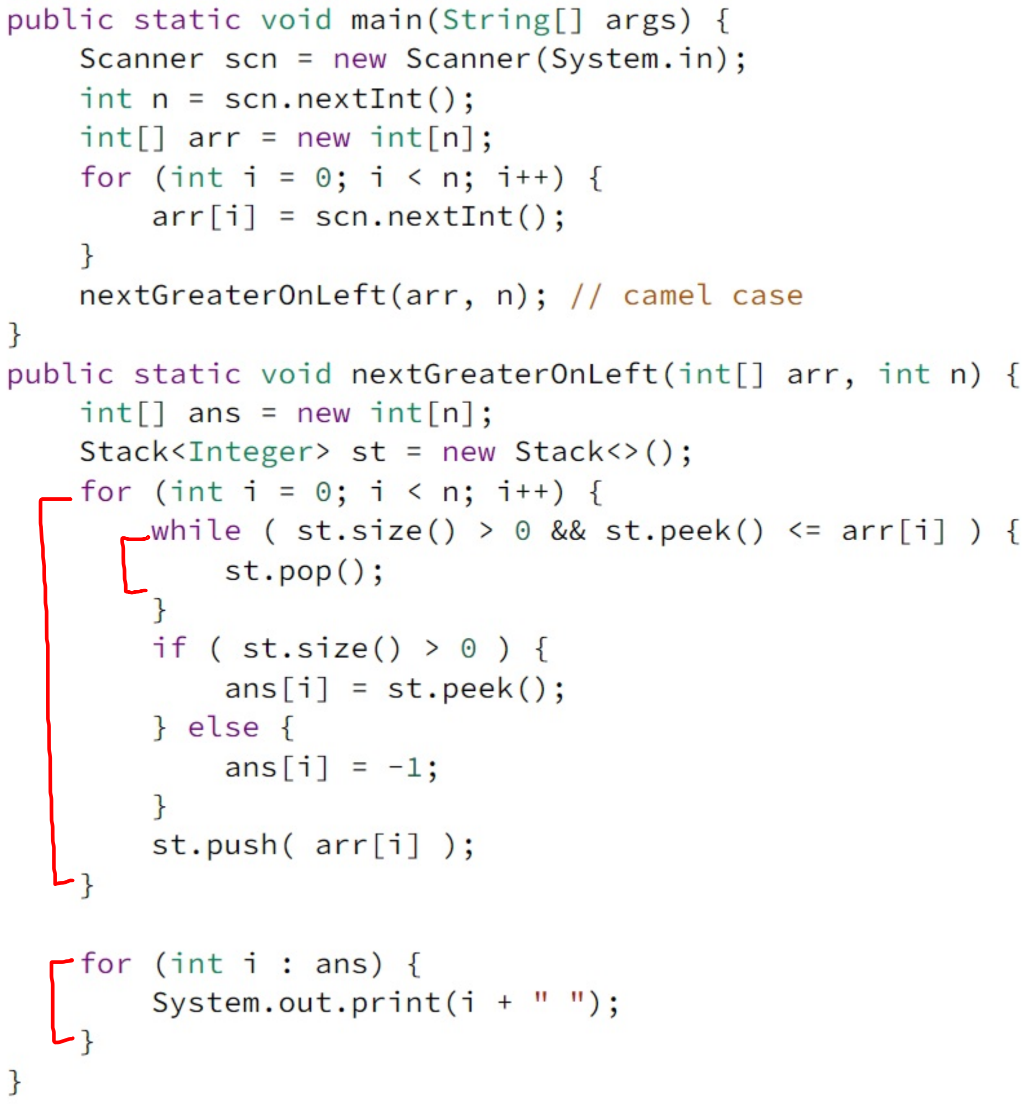
9	5	4	2	8	4	7
↑	↑	↑	↑	↑	↑	↑
-1	9	5	4	9	8	8



deside and push

psudo code

```
1) create stack
2) loop from 0 to n-1
   2.1) while ( peek < arr[i])
              pop()
   2.2) if   peek > arr[i]
              ans[i] = peek
   2.3 push (arr[i])
```



```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    nextGreaterOnLeft(arr, n); // camel case
}
public static void nextGreaterOnLeft(int[] arr, int n) {
    int[] ans = new int[n];
    Stack<Integer> st = new Stack<>();
    for (int i = 0; i < n; i++) {
        while ( st.size() > 0 && st.peek() <= arr[i] ) {
            st.pop();
        }
        if ( st.size() > 0 ) {
            ans[i] = st.peek();
        } else {
            ans[i] = -1;
        }
        st.push( arr[i] );
    }

    for (int i : ans) {
        System.out.print(i + " ");
    }
}
```

T.C = O(N)

→ Variation

- Nearest greater element on left (sol)
- Nearest smaller element on left (just change sign in while loop)
- Nearest greater element on right (only run loop in reverse)
- Nearest smaller element on right (change loop as well as sign)

Next Smaller Element To The Right

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    nextGreaterOnLeft(arr, n); // camel case
}
public static void nextGreaterOnLeft(int[] arr, int n) {
    int[] ans = new int[n];
    Stack<Integer> st = new Stack<>();
    for (int i = n - 1; i >= 0; i--) {
        while ( st.size() > 0 && st.peek() >= arr[i] ) {
            st.pop();
        }
        if ( st.size() > 0 ) {
            ans[i] = st.peek();
        } else {
            ans[i] = -1;
        }
        st.push( arr[i] );
    }

    for (int i : ans) {
        System.out.print(i + " ");
    }
}
```

T.C = O(N)

S.C = O(N)

⇒ Hashmap

[all operations in HM are having constant TC]

(Note :- amotised O(1))

Hashmap

key → value
(String) (Integer)

"Banglore" → 257

"Delhi" → 265

"Mumbai" → 0

"Chennai" → 500

"Kolkata" → 357

Key	value
Integer	Integer
Boolean	Boolean
Character	Character
Double	Double
String	String
⋮	array
	Arraylist
	Stack
	Queue
	PQ . . . —

Syntex

key Data type

value data type

```
HashMap< String, Integer> map = new HashMap<>();
```

Inbuilt f^n

key

value

- `map.put("Banglore", 257);` // to insert a pair
- `map.get("Chennai");` // to access value mapped with key

  (return type :- value data type)
- `map.remove("Mumbai");`

(Evergreen f^n :- size() & isEmpty())

↳ map.containsKey("banglore"); // false

(map is case sensitive)

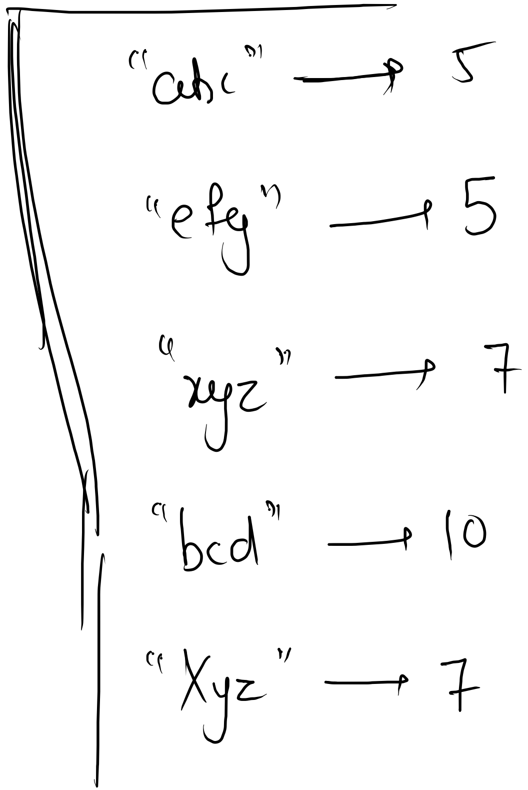
↳ map.containsValue(500); // true

Note:- 1) all keys are always unique

2) values can be same

3) if duplicate key is added, then value will be updated

4) data is unorganised



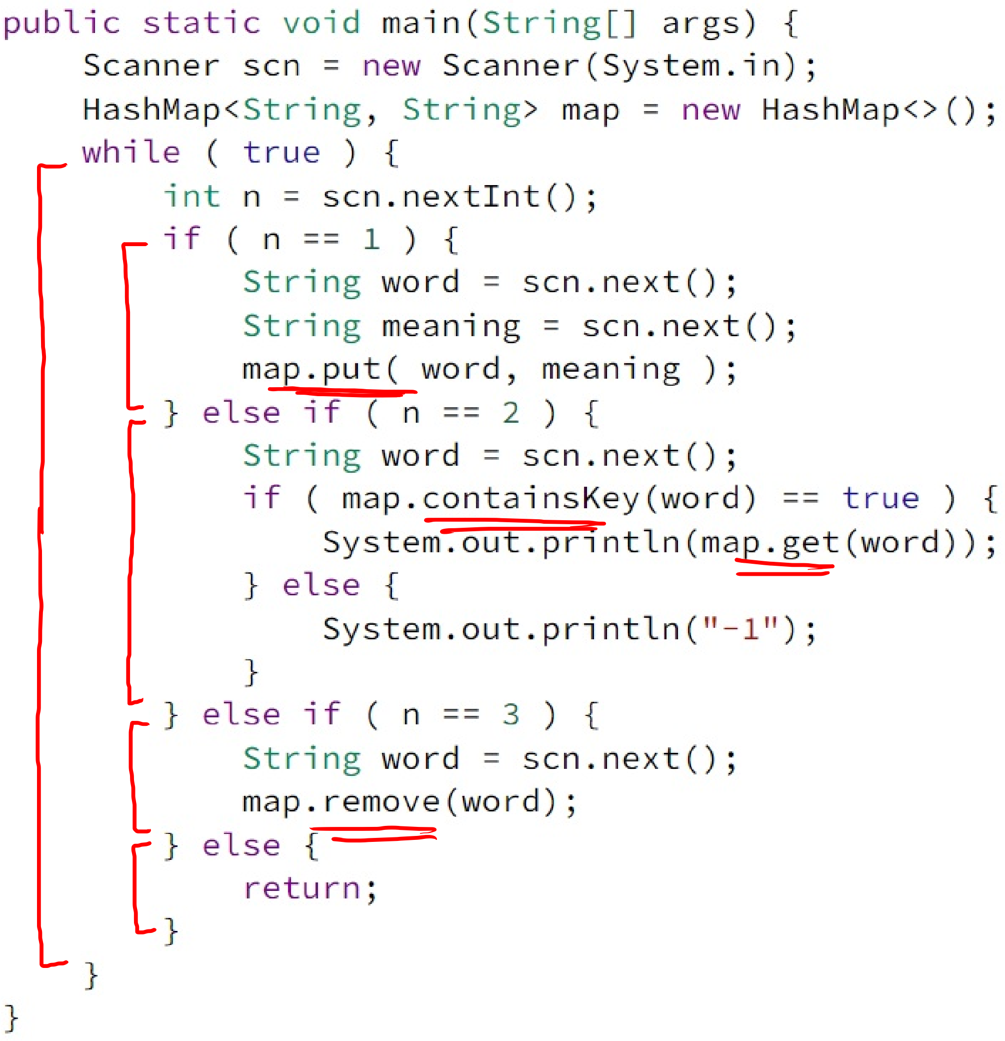
map.put("bcd", 10)

map. put("Xyz", 7)

map. put("efg", 5)

Word Meaning

Code



```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    HashMap<String, String> map = new HashMap<>();
    while ( true ) {
        int n = scn.nextInt();
        if ( n == 1 ) {
            String word = scn.next();
            String meaning = scn.next();
            map.put( word, meaning );
        } else if ( n == 2 ) {
            String word = scn.next();
            if ( map.containsKey(word) == true ) {
                System.out.println(map.get(word));
            } else {
                System.out.println("-1");
            }
        } else if ( n == 3 ) {
            String word = scn.next();
            map.remove(word);
        } else {
            return;
        }
    }
}
```