- Figure out what is the Sorting algo used by my prog. language
- How a hash map works internally
    - Hash colission
    - Chaining

In [8]:
```
def binary_search(data, val):
    s = 0
    e = len(data) - 1

    while s <= e:
        m = int((s+e)//2)

        if data[m] == val:
            return m

        if data[m] > val:
            e = m - 1
        else:
            s = m + 1

    return -1

print(func([1,3,5,6,7,8,9], 9))
print(func([1,3,5,6,7,8], 9))

"""
data 1,3,5,6,7,8,9
     0 1 2 3 4 5 6

s = 0 3 5
e   7 7 7
m   3 3 5 6

val 9

data 1,3,5,6,7,8
     0 1 2 3 4 5

s = 0 4 6
e = 6 6 6
m = 3 5

"""
```

```
6
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[8], line 19
     16     return -1
     18 print(func([1,3,5,6,7,8,9], 9))
---> 19 print(func([1,3,5,6,7,8], 9))
     21 """
     22 data 1,3,5,6,7,8,9
     23      0 1 2 3 4 5 6
   (...)
     37 
     38 """

Cell In[7], line 8, in func(data, val)
      5 while s <= e:
      6     m = int((s+e)//2)
----> 8     if data[m] == val:
      9         return m
     11     if data[m] > val:

IndexError: list index out of range
```

Stack

```
In [ ]:
```

LIFO: Last in First out.

Examples + terms:

- Stack of plates, Stack trace
- Stack overflow
- Stack undeflow

Operations:

- push(): Add Data at the end/top
- pop(): Remove data from the end/top
- peek(): Look at the element at the top without removing it
- empty()

```
In [ ]:
```

In [9]:
```
for c in '()[]{}':
    print(c, ord(c))
```

```
( 40
) 41
[ 91
] 93
{ 123
} 125
```

C++
Stack:

- push()
- pop()
- top()
- empty()

Stack using Vector:

- push: push_back()
- pop: pop_back()
- peek: back()
- empty: empty()

Java
Stack:

- push()
- pop()
- peek()
- empty()

In [ ]:
```
```

Time Complexity of Operations

```
template <type T>
class Stack {

    public:
        void push_back(); # O(1)
        void pop();       # O(1)
        T peek();         # O(1)
        bool empty();     # O(1)
}
```

Queue

- FIFO: First in First out

Queue

- Enqueue: Insert/ Add /Push: O(1)
- Dequeue: Remove/ Delete/ Pop: O(1)
- Peek(): O(1)
- Empty(): O(1)

Dequeue

- Circular Queue
- Double Ended Queue

In [ ]:

```
```

Question

[https://leetcode.com/problems/valid-parentheses/ (https://leetcode.com/problems/valid-parentheses/)](https://leetcode.com/problems/valid-parentheses/)

TC: O(n)
SC: O(n)

In [ ]:
```python
class Solution:
    def isValid(self, s: str) -> bool:
        brackets = {'(':')', '{':'}', '[':']'} # O(1)

        stack = [] # O(n)

        for i in s:
            if i in brackets:
                stack.append(i)
            else:
                if len(stack) == 0 or i != brackets[stack.pop()]:
                    return False

        return len(stack) == 0
```

```java
class Solution {
    public boolean isValid(String s) {

        char []arr = s.toCharArray();
        Stack<Character> stack = new Stack<>();

        for (char ch:arr){
            if(stack.isEmpty()){
                stack.push(ch);
            }else{
                char top = stack.peek();
                if(ch-top==1 || ch -top == 2){
                    stack.pop();
                }else{
                    stack.push(ch);
                }
            }
        }
        return stack.isEmpty();

    }
}
```

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> st = new Stack<>();
        // )

        for(int i = 0; i < s.length(); i++){
            if(s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt
(i) == '['){
                st.push(s.charAt(i));
            }else{
                if(st.isEmpty())
                    return false;
                else if((st.peek() == '(' && s.charAt(i) == ')')|| (s
t.peek() == '{' && s.charAt(i) == '}') || (st.peek() == '[' &&s.charA
t(i) == ']'))
                    st.pop();
                else{
                    return false;
                }
            }

        }

        return st.isEmpty();

    }
}
```

```
class Solution {
public:
    bool isValid(string s) {
      // ([)]

      stack<char> st;

      for(auto c: s) {
        if (c == '{' || c== '[' || c == '(') {
            st.push(c);
        } else {
            if (st.empty()) {
                return false;
            }

            if ((c == ')' && st.top() != '(') || (c == ']' && st.top
() != '[') || (c == '}' && st.top() != '{')) {
                return false;
            }
            st.pop();
        }

      }

      return st.empty();
    }
};
```

In [ ]:

Next greater element

In [ ]:

```
// Brute Force
//  for i=0; i < n; i++
//     temp = -1
//     for j = i+1; j < n; j++
//          if a[i] < a[j]
//               temp = a[j]
//               break
//      a[i] = temp
// TC: O(n^2)
// SC: O(1)
// [3 4 -1 -1]
```

In [16]:
```
# TC: O(n)
# SC: O(n)

def next_greater(data):
    stack = [] # use list  as a stack

    for i in range(len(data)-1,-1,-1): # for(i=0;i<n;i++)


        while len(stack) > 0 and stack[-1] < data[i]: # stack[-1] => top of st
            stack.pop()
        curr = data[i]

        if len(stack) == 0:
            data[i] = -1
        else:
            data[i] = stack[-1]
        stack.append(curr)

    return data

print(next_greater([3,2,1,4,5,4]))
print(next_greater([]))
print(next_greater([1,2,3,4]))
print(next_greater([4,3,2,1]))
```

```
[4, 4, 4, 5, -1, -1]
[]
[2, 3, 4, -1]
[-1, -1, -1, -1]
```

In [ ]:

https://leetcode.com/problems/daily-temperatures/ (https://leetcode.com/problems/daily-temperatures/)

In [ ]:
```

        [73,74,75,71,69,72,76,73]
         0   1  2  3  4  5  6  7
        [74,75,76,72,72,76,-1-1]
        [1, 2, 6, 5, 5, 6,-1,-1]
         0  1  2  3  4  5  6  7
        [1, 1, 4, 2, 1, 1, 0, 0]
```

In [ ]:
```
## brute force
# TC: O(n^2)
# SC: O(1)
class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        vector<int> answers;

        for(int i = 0; i < temperatures.size(); i++) {
            int ans =  0;
            for(int j = i+1; j < temperatures.size(); j++){
                if (temperatures[i] < temperatures[j]) {
                    ans =  j-i;
                    break;
                }
            }
            answers.push_back(ans);
        }
        return answers;
    }
};
```

In [ ]:
```
class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        vector<int> answers;
        stack<int> st;

        for(int i = temperatures.size()-1; i >= 0; i--) {
            int ans = 0;

            while (!st.empty() && temperatures[st.top()] <= temperatures[i]) {
                st.pop();
            }

            if (!st.empty()) {
                ans = st.top() - i;
            }
            answers.push_back(ans);
            st.push(i);
        }
        reverse(answers.begin(), answers.end());
        return answers;
    }
};
```

In [ ]:

In [ ]:

Question

https://leetcode.com/problems/next-greater-element-i/ (https://leetcode.com/problems/next-greater-element-i/)

Two approach:

- Value based (R->L)
- Index based (L->R)

In [ ]:

In [ ]:

Question
https://leetcode.com/problems/minimum-add-to-make-parentheses-valid/ (https://leetcode.com/problems/minimum-add-to-make-parentheses-valid/)

TC: O(n)

SC: O(1)

Using Stack

In [ ]:
```
class Solution {
    public int minAddToMakeValid(String s) {
        Stack<Character> st = new Stack<>();

        int invalidOpening = 0, invalidClosing = 0;
        for(int i = 0; i < s.length(); i++){
            if(s.charAt(i) == '(')
                st.push(')');
            else if(st.isEmpty())
                invalidClosing++;
            else if(s.charAt(i) == st.peek())
                st.pop();
        }
        invalidOpening = st.size();

        return invalidOpening + invalidClosing;
    }
}
```

In [ ]:
```
class Solution:
    def minAddToMakeValid(self, s: str) -> int:
        stack = []

        for i in s:
            if i == '(' or not stack or (stack and stack[-1] != '('):
                stack.append(i)
            else:
                stack.pop()

        return len(stack)
```

In [ ]:

TC: O(n)

SC: O(1)

Without Stack

In [ ]:
```
class Solution {
    public int minAddToMakeValid(String s) {
    int openParan = 0;
       int count = 0;
       for(char c : s.toCharArray()) {
           if(c == '(') {
               openParan++;
            }
           else if(c == ')') {
               if(openParan > 0) openParan--;
               else count++;
           }
        }
       count += openParan;
       return count;
    }
}
```

In [ ]:

In [ ]:

Question
[https://leetcode.com/problems/next-greater-element-ii/ (https://leetcode.com/problems/next-greater-element-ii/)](https://leetcode.com/problems/next-greater-element-ii/)

In [ ]:

In [ ]:

In [ ]: