Name: Aryan Kumar Singh **UID: 22BCS13577**

Branch: BE-CSE **Section:IOT_615-B**

Semester: 05 **DOP:23/12/2024**

DAY- 4th(Monday)

1. Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return the next greater number for every element in nums.

The next greater number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number.

Example 1:

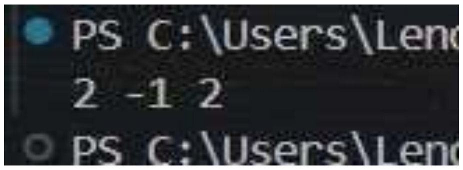
Input: nums = [1,2,1]

Output: [2,-1,2] **Code:**

```
#include <vector>

#include <stack> #include <iostream> using namespace

std; vector<int> nextGreaterElements(vector<int>

&nums)

{     int n = nums.size();     vector<int>

result(n, -1);      stack<int> st;

for

(int i = 0; i < 2 * n; ++i)

   {

       while (!st.empty() && nums[st.top()] < nums[i % n])
```

```
        {               result[st.top()] =
nums[i % n];              st.pop();

        }         if (i <
n)          {             st.push(i);
        }     }     return result; } int main() {     vector<int>
nums = {1, 2, 1};     vector<int> result
= nextGreaterElements(nums);     for (int num :
result)
    {
        cout << num << " ";
    }
return 0;  }
```

Output:



```
PS C:\Users\Len
2 -1 2
PS C:\Users\Len
```

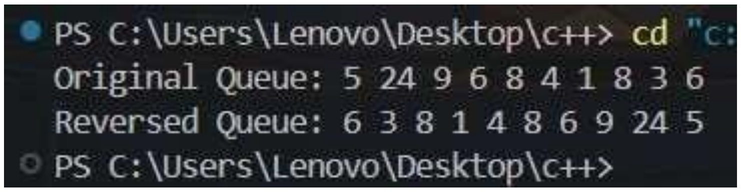
2. Given a queue, write a recursive function to reverse it.

Standard operations allowed :
enqueue(x) : Add an item x to rear of queue. dequeue() : Remove an item from front of queue. empty() : Checks if a queue is empty or not.
Examples 1:
Input : Q = [5, 24, 9, 6, 8, 4, 1, 8, 3, 6]
Output : Q = [6, 3, 8, 1, 4, 8, 6, 9, 24, 5]
Explanation : Output queue is the reverse of the input queue.

Code:

```
#include <iostream> #include <queue>
using namespace std; void
reverseQueue(queue<int> &q)
{      if
(q.empty())
    {         return;
    }
    int frontElement = q.front();     q.pop();
reverseQueue(q);
    q.push(frontElement);
}

int main()
 {     queue<int> q;
q.push(5);
    q.push(24);
    q.push(9);
    q.push(6);
    q.push(8);
    q.push(4);
    q.push(1);
    q.push(8);
    q.push(3);
    q.push(6);      cout <<
"Original Queue: ";
queue<int> temp = q;
while (!temp.empty())
    {
        cout << temp.front() << " ";

        temp.pop();
    }      cout << endl;
reverseQueue(q);      cout <<
"Reversed Queue: ";     while
(!q.empty())
    {         cout << q.front() <<
" ";         q.pop();     }
    cout << endl;

return 0; }
```

Output:



```
PS C:\Users\Lenovo\Desktop\c++> cd "c:
Original Queue: 5 24 9 6 8 4 1 8 3 6
Reversed Queue: 6 3 8 1 4 8 6 9 24 5
PS C:\Users\Lenovo\Desktop\c++>
```

3. Given a balanced parentheses string s, return the score of the string.

The score of a balanced parentheses string is based on the following rule:

"()" has score 1.

AB has score A + B, where A and B are balanced parentheses strings.

(A) has score 2 * A, where A is a balanced parentheses string.

Example 1:

Input: s = "()"

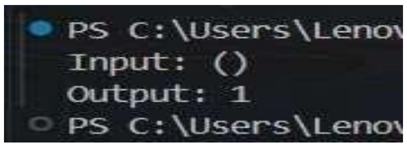
Output: 1

Code:

```
#include <iostream>

#include <stack>  #include

<string>  using namespace

std; int

scoreOfParentheses(string s)

{     stack<int> st;

for (char c

: s)
```

```
    {     if (c == '(')

{

st.push(0);

        }         else          {              int

top = st.top();              st.pop();

if (top

== 0)

             {

st.push(1);

             }

else              {                  int innerScore = top * 2;
st.pop();                  st.push(st.empty() ? innerScore :
innerScore + st.top());              }

        }     }     int

score = 0;     while

(!st.empty())

    {         score += st.top();         st.pop();     }     return

score; } int main() {     string s = "()";     cout << "Input:

" << s << endl;     cout << "Output: " <<

scoreOfParentheses(s) << endl;     return 0;

}
```

Output:



```
PS C:\Users\Lenov
Input: ()
Output: 1
PS C:\Users\Lenov
```

4. You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.
Return the max sliding window.

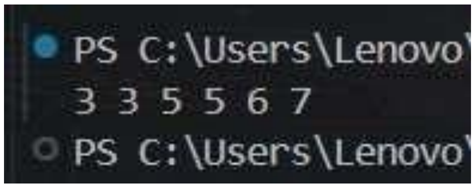
Example 1:
Input: nums = [1,3,-1,-3,5,3,6,7],

k = 3 **Output:** [3,3,5,5,6,7]

Code:

```
#include <iostream>

#include <vector> #include <deque> using namespace std;

vector<int> maxSlidingWindow(vector<int> &nums, int k)

{     deque<int> dq;      vector<int>

result;     for (int i = 0; i < nums.size();

++i)

    {

        if (!dq.empty() && dq.front() == i - k)

        {

dq.pop_front();

        }

        while (!dq.empty() && nums[dq.back()] < nums[i])

        {

            dq.pop_back();

        }
        dq.push_back(i);          if
(i >= k - 1)
        {
            result.push_back(nums[dq.front()]);           }
}     return result; } int main() {     vector<int> nums
= {1, 3, -1, -3, 5, 3, 6, 7};     int k = 3;      vector<int>
result = maxSlidingWindow(nums, k);
    for (int x : result)
    {         cout << x
<< " ";
    }      cout <<
endl;     return
0;  }
```

Output:



```
PS C:\Users\Lenovo
3 3 5 5 6 7
PS C:\Users\Lenovo
```

5. You have an infinite number of stacks arranged in a row and numbered (left to right) from 0, each of the stacks has the same maximum capacity.

Implement the DinnerPlates class:

DinnerPlates(int capacity) Initializes the object with the maximum capacity of the stacks capacity.

void push(int val) Pushes the given integer val into the leftmost stack with a size less than capacity.

int pop() Returns the value at the top of the rightmost non-empty stack and removes it from that stack, and returns -1 if all the stacks are empty.

int popAtStack(int index) Returns the value at the top of the stack with the given index index and removes it from that stack or returns -1 if the stack with that given index is empty.

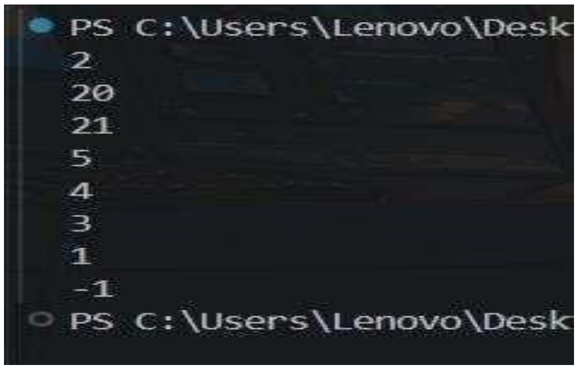
Code:

```
#include <iostream>

#include <vector>

#include <stack> #include

<set> using namespace

std; class

DinnerPlates

{ private:

   int                    capacity;

vector<stack<int>> stacks;

set<int> available;  public:

   DinnerPlates(int cap) : capacity(cap) {}

void push(int val)

   {        if

(available.empty())

       {

           stacks.push_back(stack<int>());

available.insert(stacks.size() - 1);

       }
```

```
        int idx = *available.begin();          stacks[idx].push(val);
if
(stacks[idx].size() == capacity)
        {
available.erase(idx);
        }     }
int pop()
    {
        while (!stacks.empty() && stacks.back().empty())
        {
            stacks.pop_back();          }
if (stacks.empty())               return
-1;                              int    val    =
stacks.back().top();
stacks.back().pop();
available.insert(stacks.size() - 1);
return val;
    }
    int popAtStack(int index)
    {          if (index >= stacks.size() ||
stacks[index].empty())              return -1;         int val =
stacks[index].top();          stacks[index].pop();
available.insert(index);          return val;
```

```
    } }; int main()
{
    DinnerPlates D(2);
    D.push(1);
    D.push(2);
    D.push(3);
    D.push(4);
    D.push(5);      cout <<
D.popAtStack(0) << endl;

    D.push(20);
    D.push(21);    cout <<
D.popAtStack(0)  <<  endl;   cout  <<
D.popAtStack(2)  <<  endl;   cout  <<
D.pop() << endl;             cout <<
D.pop() << endl;              cout <<
D.pop() << endl;              cout <<
D.pop() << endl;              cout <<
D.pop() << endl;              return 0;
}
```

Output:



```
PS C:\Users\Lenovo\Desk
2
20
21
5
4
3
1
-1
PS C:\Users\Lenovo\Desk
```

6. Suppose there is a circle. There are N petrol pumps on that circle. Petrol pumps are numbered 0 to (N-1) (both inclusive). You have two pieces of information corresponding to each of the petrol pump: (1) the amount of petrol that particular petrol pump will give, and (2) the distance from that petrol pump to the next petrol pump.

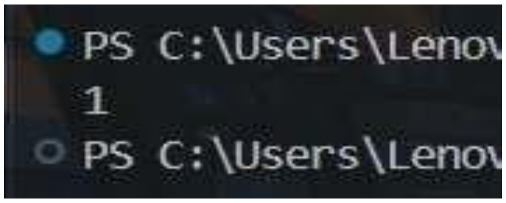
Initially, you have a tank of infinite capacity carrying no petrol. You can start the tour at any of the petrol pumps. Calculate the first point from where the truck will be able to complete the circle. Consider that the truck will stop at each of the petrol pumps. The truck will move one kilometer for each litre of the petrol.

Code:

```
#include <iostream> #include
<vector> using namespace
std;
int findStartingPoint(int N, vector<pair<int, int>> &pumps)
{
   long long total_petrol = 0, total_distance = 0;
long long current_petrol = 0;     int start = 0;     for
(int i = 0; i < N; ++i)
    {        total_petrol += pumps[i].first;         total_distance
 += pumps[i].second;         current_petrol += pumps[i].first
- pumps[i].second;        if
(current_petrol < 0)
```

```
        {              start = i +
1;              current_petrol =
0;
        }
    }
    if (total_petrol >= total_distance)
    {          return
start;      }
return -1;
} int main() {     int N = 3;     vector<pair<int, int>> pumps =
{{1, 5}, {10, 3}, {3, 4}};     cout << findStartingPoint(N,
pumps) << endl;     return 0;
}
```

Output:



```
PS C:\Users\Lenov
1
PS C:\Users\Lenov
```

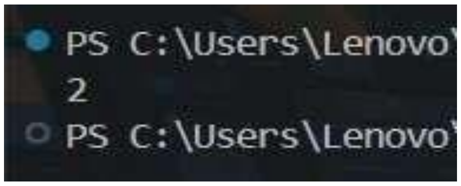
7. There are a number of plants in a garden. Each of the plants has been treated with some amount of pesticide. After each day, if any plant has more pesticide than the plant on its left, being weaker than the left one, it dies.

You are given the initial values of the pesticide in each of the plants. Determine the number of days after which no plant dies, i.e. the time after which there is no plant with more pesticide content than the plant to its left.

Code:

```
#include <iostream>

#include <vector>

#include <stack> #include

<algorithm> using namespace std; int

poisonousPlants(vector<int> &p)

{     int n = p.size();

stack<int> st;     vector<int>

days(n, 0);     int maxDays

= 0;     for

(int i = 0; i < n; ++i)

    {

        while (!st.empty() && p[i] <= p[st.top()])

        {

st.pop();         }         if

(!st.empty())

        {             days[i] = days[st.top()]

+ 1;

        }

st.push(i);          maxDays =

max(maxDays, days[i]);

    }
    return maxDays; } int main() {      int n
= 7;      vector<int> p = {6, 5, 8, 4, 7,
10, 9};      cout << poisonousPlants(p)
<< endl;      return 0;
}
```

Output:



```
PS C:\Users\Lenovo
2
PS C:\Users\Lenovo
```

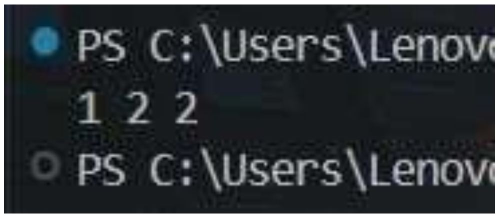
8. You are given an integer array nums of length n and an integer array queries.

Let gcdPairs denote an array obtained by calculating the GCD of all possible pairs (nums[i], nums[j]), where 0 <= i < j < n, and then sorting these values in ascending order.
For each query queries[i], you need to find the element at index queries[i] in gcdPairs. Return an integer array answer, where answer[i] is the value at gcdPairs[queries[i]] for each query.
The term gcd(a, b) denotes the greatest common divisor of a and b.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric> using
namespace std; int gcd(int
a, int b)
{      return __gcd(a, b);
}
vector<int> gcdQueries(vector<int> &nums, vector<int> &queries)
{       vector<int>
gcdPairs;
int n = nums.size();      for
(int i = 0; i < n; ++i)
    {
        for (int j = i + 1; j < n; ++j)
        {
            gcdPairs.push_back(gcd(nums[i], nums[j]));
        }
    }
    sort(gcdPairs.begin(), gcdPairs.end());
vector<int> answer;
    for (int query : queries)
    {
        answer.push_back(gcdPairs[query]);
    }
    return answer;
} int main()
{
     vector<int> nums = {2, 3, 4};
vector<int> queries = {0, 2, 2};
    vector<int> result = gcdQueries(nums, queries);  for
    (int res : result)
    {          cout << res
<< " ";
    }
    cout << endl;
return 0;
}
```

Output:



```
PS C:\Users\Lenov
1 2 2
PS C:\Users\Lenov
```

9. Given a string containing just the characters '(' and ')', return the length of the longest valid (well-formed) parentheses substring.

Example 1:

Input: s = "(()"

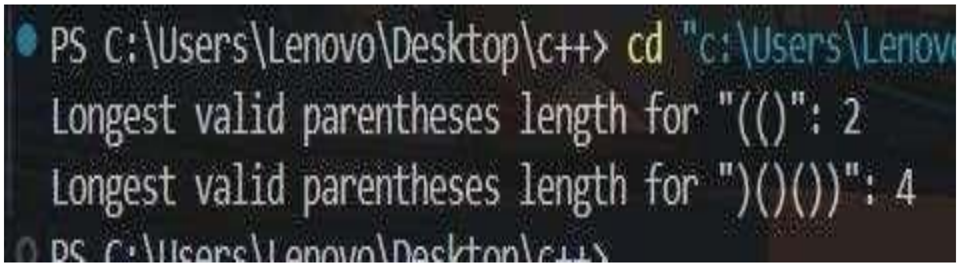
Output: 2

Code:

```
#include <iostream>

#include <stack> #include <string>

using namespace std; int

longestValidParentheses(string s) {

stack<int> st;    st.push(-1);      int

maxLength = 0;    for (int i = 0; i <

s.length(); ++i) {


      if (s[i] == '(')

{           st.push(i);

}         else          {

st.pop();             if

(!st.empty())

         {

             maxLength = max(maxLength, i - st.top());

         }           else

{
```

```
st.push(i);
            }
        }
    }
    return maxLength;
} int main() {     string
s1 = "(()"; string s2 =
")()())";      cout <<
"Longest valid
parentheses length for
\"" << s1 << "\": " <<
longestValidParenthes
es(s1) << endl;

    cout << "Longest valid parentheses length for \"" << s2 << "\": " <<
longestValidParentheses(s2) << endl;
    return 0;
}
```

Output:



```
PS C:\Users\Lenovo\Desktop\c++> cd "c:\Users\Lenov
Longest valid parentheses length for "(()": 2
Longest valid parentheses length for ")()())": 4
PS C:\Users\Lenovo\Desktop\c++>
```

10. You are given an integer array nums and an integer k.

Find the longest subsequence of nums that meets the following requirements:

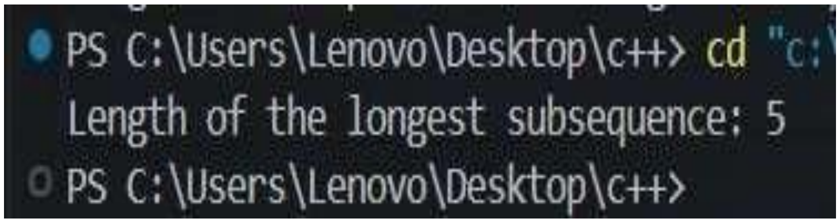
- **The subsequence is strictly increasing and**
- **The difference between adjacent elements in the subsequence is at most k.**

Return the length of the longest subsequence that meets the requirements.
A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Code:

```
#include <iostream>
#include <vector> #include
<algorithm> using namespace
std;
int longestSubsequence(vector<int> &nums, int k)
{     int n =
nums.size();
vector<int> dp(n, 1);     for
(int i = 1; i < n; ++i)
    {         for (int j = 0; j < i;
++j)
        {
            if (nums[i] > nums[j] && nums[i] - nums[j] <= k)
            {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
    }
    return *max_element(dp.begin(), dp.end());
} int main()
{
    vector<int> nums = {4, 2, 1, 4, 3, 4, 5, 8, 15};
int k = 3;
    cout << "Length of the longest subsequence: " << longestSubsequence(nums, k) << endl;
return 0;
}
```

Output:



```
PS C:\Users\Lenovo\Desktop\c++> cd "c:
Length of the longest subsequence: 5
PS C:\Users\Lenovo\Desktop\c++>
```