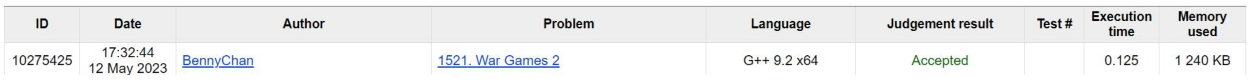
Laboratory work # 5

Student: *Zhanghao Chen*
Student ID: 21321205
Timus Name: *BennyChan*

Mail: 1824362950@qq.com

Problem # 1521 *War Games 2*

Screenshot from Timus:



ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
10275425	17:32:44 12 May 2023	BennyChan	1521. War Games 2	G++ 9.2 x64	Accepted		0.125	1 240 KB

Explanation of algorithm:

The solveArrayCopy () function takes two inputs: *cnt* and *k*. It creates a vector '*arr*' of size *cnt* and fills it with values ranging from 1 to *cnt*. The algorithm then proceeds to iterate *cnt* times. In each iteration, it selects the *k*th element from the start index and prints it. It then removes this element from the vector by shifting the remaining elements left or right and adjusting the indices accordingly. The algorithm continues this process until there is only one element left in the vector.

Computational complexity of algorithm:

$$O(N^2)$$

Source code:

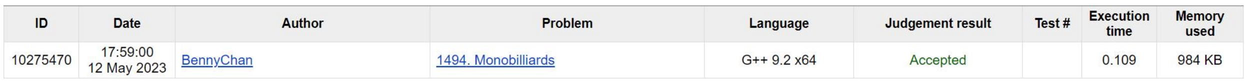
```
#include <bits/stdc++.h>
using namespace std;

void solveArrayCopy() {
    int cnt, k;
    cin >> cnt >> k;
    vector<int> arr(cnt);
    for (int i = 1; i <= cnt; i++) arr[i - 1] = i;
    int left = 0, right = cnt - 1, start = 0, tmp1 = 0;
    for (int leaveCnt = 0; leaveCnt < cnt; leaveCnt++) {
        if (right == left) {
            cout << arr[left] << " ";
            break;
        }
        tmp1 = start + (k - 1);
        if (tmp1 > right) {
            tmp1 = (tmp1 - right) % (right - left + 1) + left - 1;
            if (tmp1 < left) tmp1 = right;
        }
        cout << arr[tmp1] << " ";
        if (tmp1 - left > right - tmp1) {
            for (int i = tmp1 + 1; i <= right; i++) arr[i - 1] = arr[i];
            right--;
            start = tmp1;
        } else {
            for (int i = tmp1 - 1; i >= left; i--) arr[i + 1] = arr[i];
            left++;
            start = tmp1 + 1;
        }
    }
    cout << endl;
}

int main() {
    solveArrayCopy();
    return 0;
}
```

Problem # 1494 *Monobilliards*

Screenshot from Timus:



ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
10275470	17:59:00 12 May 2023	BennyChan	1494. Monobilliards	G++ 9.2 x64	Accepted		0.109	984 KB

Explanation of algorithm:

To address the problem at hand, we can implement a solution that involves creating two arrays. Array A will be used to store the input value (i.e., the number of the ball taken out), while array B will store the ball into the bag in order. Next, we will compare the values in arrays A and B. If a value in array A is equal to that in array B, we will update the corresponding value in array B and retrieve the next value in array A. This process will continue until all values in array A have been processed. By following this approach, we can ensure that the balls are bagged in the required order, and all the numbers in the final array B will be retrieved as expected.

Computational complexity of algorithm:

$$O(N)$$

Source code:

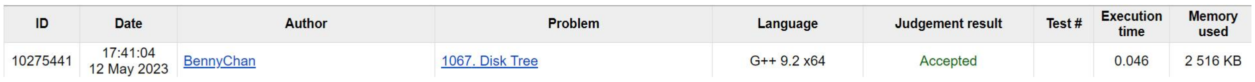
```
#include <cstdio>
using namespace std;

int a[100002], stack[100002];

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
    {
        scanf("%d", &a[i]);
    }
    int top = 0, index = 1;
    for (int i = 1; i <= n; ++i)
    {
        stack[++top] = i;
        while (top && stack[top] == a[index])
        {
            top--;
            index++;
        }
    }
    if (top)
    {
        printf("Cheater");
    }
    else
    {
        printf("Not a proof");
    }
    return 0;
}
```

Problem # 1067 *Disk Tree*

Screenshot from Timus:



ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
10275441	17:41:04 12 May 2023	BennyChan	1067. Disk Tree	G++ 9.2 x64	Accepted		0.046	2 516 KB

Explanation of algorithm:

We use a **tree structure** using a linked list and a node class to represent the directory structure, where each node represents a directory or file. Each node object is stored in a list which acts as a container for the entire directory tree. The program builds a tree structure by adding nodes step by step and uses a recursive method to print out all nodes and their children. The put () function adds nodes level by level by splitting the path into different levels of directories. The putSorted () function is responsible for inserting new nodes into the node list at each level in alphabetical order. The print Directory () function traverses the tree in a depth-first manner and prints out the names of each directory, with each level indented by a certain number of spaces. The main () function initializes an empty directory tree, reads in a number '*N*' of directory paths to add, and calls the put function for each path. It then calls printDirectory () function to output the entire directory tree.

Computational complexity of algorithm:

$$O(N \times \log_2 N)$$

Source code:

```
#include <iostream>
#include <list>
#include <string>
using namespace std;

class Node {
public:
    string date;
    list<Node*> next;
    Node(string date) {
        this->date = date;
    }
    string toString() {
        return date;
    }
};

Node* putSorted(list<Node*>& tree, string name) {
    list<Node*>::iterator it = tree.begin();
    Node* newDir = new Node(name);
    bool inserted = false;
    while (it != tree.end()) {
        Node* dir = *it;
        int compare = dir->date.compare(name);
        if (compare == 0) {
            inserted = true;
            newDir = dir;
            break;
        }
        else if (compare > 0) {
            inserted = true;
            tree.insert(it, newDir);
            break;
        }
        it++;
    }
    if (!inserted) {
        tree.push_back(newDir);
    }
    return newDir;
}

void put(list<Node*>& tree, string directory) {
    int slashId = directory.find('\\\\');
    if (slashId == -1) {
        putSorted(tree, directory);
    }
    else {
        string currentDirectory = directory.substr(0, slashId);
        string nextDirectory = directory.substr(slashId + 1);
        Node* dir = putSorted(tree, currentDirectory);
        if (dir->next.empty())
            dir->next = list<Node*>();
        put(dir->next, nextDirectory);
    }
}

void printDirectory(list<Node*>& tree, int level) {
    if (!tree.empty()) {
        list<Node*>::iterator it = tree.begin();
        while (it != tree.end()) {
```

```cpp
            Node* dir = *it;
            for (int j = 0; j < level; j++) {
                cout << ' ';
            }
            cout << dir->date << endl;
            printDirectory(dir->next, level + 1);
            it++;
        }
    }
}

int main() {
    list<Node*> tree = list<Node*>();
    int N;
    cin >> N;
    cin.ignore();
    for (int i = 0; i < N; i++) {
        string directory;
        getline(cin, directory);
        put(tree, directory);
    }
    printDirectory(tree, 0);
    return 0;
}
```