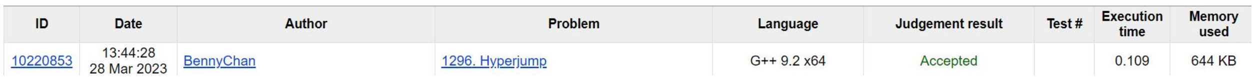
Laboratory work # 1

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

Mail: 1824362950@qq.com

Problem # 1296 *Hyperjump*

Screenshot from Timus:



ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
10220853	13:44:28 28 Mar 2023	BennyChan	1296. Hyperjump	G++ 9.2 x64	Accepted		0.109	644 KB

Explanation of algorithm:

Dynamic – *Kadane's* algorithm:

Go through all elements, sum them, and save maximum value of sum. Negative sum is useless, which will be replaced with 0. The final output is the maximum of sum.

Computational complexity of algorithm:

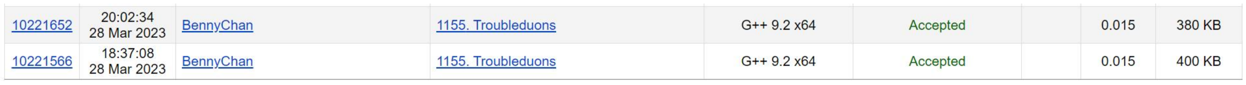
$$O(N)$$

Source code:

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main(){

    int N;
    int a[60001];

    int ThisSum = 0;
    int MaxSum = 0;

    cin >> N;
    for(int i = 0; i < N; i++){
        cin >> a[i];
    }

    for(int i = 0; i < N; i++){
        ThisSum += a[i];
        if(ThisSum > MaxSum){
            MaxSum = ThisSum;
        }else if(ThisSum < 0){
            ThisSum = 0;
        }
    }
    cout << MaxSum << endl;

    return 0;
}
```

Problem # 1155 *Troubleduons*

Screenshot from Timus:



10221652	20:02:34 28 Mar 2023	BennyChan	1155. Troubleduons	G++ 9.2 x64	Accepted		0.015	380 KB
10221566	18:37:08 28 Mar 2023	BennyChan	1155. Troubleduons	G++ 9.2 x64	Accepted		0.015	400 KB

Explanation of algorithm:

Categorize 8 cameras into two parts, namely *ACFH* and *BDEG*, the four cameras inside each section are not adjacent. If $A + C + F + H \,!= B + D + E + G$, return impossible due to no solution. If the sum of two parts is equal, then we follow a principle, namely for adjacent destroys while for non-adjacent moving troubleduons between vertices of the same group, finally we can successfully move other cameras (*A, C, H*) in the same part to *F* (for instance), move other cameras (*D, E, G*) in the same part to *B* (For instance). Then the numbers of cameras *F* and *B* must be equal, so we can destroy two of them gradually. At last, we make it!

Computational complexity of algorithm:

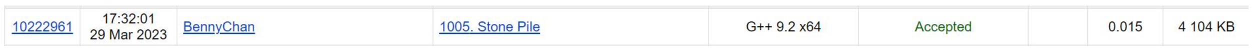
$$O(N)$$

Source code:

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

int a[8];

void move(int x,int y,int t)
{
          while(a[x]){
            if(a[t]==0){
            a[t]++;a[y]++;
            printf("%c%c+\n",t+'A',y+'A');
        }
        a[t]--;
        a[x]--;
        printf("%c%c-\n",t+'A',x+'A');
    }
}

int main()
{
    int i;
    for(i=0;i<8;i++)
        scanf("%d",&a[i]);
    int s1=a[0]+a[2]+a[5]+a[7],s2=a[1]+a[3]+a[4]+a[6];
    if(s1!=s2)
    {
        printf("IMPOSSIBLE\n");
        return 0;
    }

    move(0,5,4);
    move(2,5,6);
    move(7,5,4);
    move(3,1,0);
    move(4,1,0);
    move(6,1,2);
    while(a[1]--)
        printf("FB-\n");

    return 0;
}
```

Problem # 1005 *Stone Pile*

Screenshot from Timus:



10222961	17:32:01 29 Mar 2023	BennyChan	1005. Stone Pile	G++ 9.2 x64	Accepted		0.015	4 104 KB

Explanation of algorithm:

The problem can be viewed as a 0-1 backpack problem by using dynamic programming to solve it. There are up to 20 stones, stones of varying weight, now is to take some stones as full as possible half of the total weight of the backpack, the rest is not full of another pile, so that the difference between the two piles can be minimized!

Computational complexity of algorithm:

$$O(N \times M)$$

Source code:

```
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

int main() {
    int n;
    cin >> n;
    int w[n];
    for (int i = 0; i < n; i++) {
        cin >> w[i];
    }
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += w[i];
    }
    int half = sum / 2;
    int backpack[half+1];  //include  zero volume
// One-dimensional form given that two-dimentional form will exceed the space
    for (int i = 0; i <= half; i++) {
        backpack[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        for (int j = half; j >= w[i]; j--){
            backpack[j] = max(backpack[j], backpack[j-w[i]] + w[i]);
        }
    }
/* Prove:
    A + B = Sum
    A - B = Min
    A = (Sum + Min) / 2
    B = (Sum - Min) / 2
    Min = abs(Sum - 2 * min)
*/

    cout << abs(sum - 2 * backpack[half]) << endl;
    return 0;
}
```