Laboratory work # 4

Student: *Zhanghao Chen*
Student ID: 21321205
Timus Name: *BennyChan*
Mail: 1824362950@qq.com

Problem # 1322 *Spy*

Screenshot from Timus:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 10251520 | 11:38:12<br>22 Apr 2023 | BennyChan | 1322. Spy | G++ 9.2 x64 | Accepted | | 0.015 | 1 792 KB |

Explanation of algorithm:

Overall:

We use the Burrows-Wheeler inverse transformation algorithm

Concretely:

First, we define a function called getInd() that maps characters to their corresponding positions in the English alphabet. Then, we use scanf() to read an integer and a string from the input. Next, we define an array called characterCount[] with a size of 54, which is used to store the frequency of each character. We initialize this array to 0 using the memset() function. Then, we traverse the input string and store the frequency of each character in the characterCount[] array. Next, we traverse the characterCount[] array and calculate the cumulative sum of the frequency of each character. Based on this, we define an array called nextPositions[] with a size of 100000, which is used to store the position of each character in the new rearranged string. Then, we traverse the input string again and store the position of each character in the nextPositions[] array. Finally, we traverse the nextPositions[] array, output the new string, and add a line break.

Computational complexity of algorithm:

$$O(N)$$

Source code:

```cpp
#include <cstdio>
#include <cstring>

using namespace std;

char inputString[100001];
int nextPositions[100000];

int getInd(char c){
    if(c >= 'A' && c <= 'Z') return c-'A';
    if(c == '_') return 26;
    return c-'a'+27;
}

int main(){
    int x,stringLength;

    scanf("%d %s",&x,inputString);
    stringLength = strlen(inputString);
    --x;

    int characterCount[54];
    memset(characterCount,0,sizeof(characterCount));
    for(int i = 0;i < stringLength;++i){
        ++characterCount[getInd(inputString[i]) + 1];
    }
    for(int i = 1;i < 53;++i){
        characterCount[i] += characterCount[i-1];
    }

    for(int i = 0;i < stringLength;++i){
        nextPositions[characterCount[getInd(inputString[i])]++] = i;
    }
    for(int i = 0;i < stringLength;++i){
        putchar(inputString[x = nextPositions[x]]);
    }

    putchar('\n');

    return 0;
}
```

Problem # 1726 *Visits*

Screenshot from Timus:

| ID | Date | Author | Problem | Language | Judgement result | Test # | Execution time | Memory used |
|---|---|---|---|---|---|---|---|---|
| 10264943 | 12:36:30 3 May 2023 | BennyChan | 1726. Visits | G++ 9.2 x64 | Accepted | | 0.343 | 1 228 KB |

Explanation of algorithm:

In our case of the solution we sort the coordinates $X$ and $Y$ in ascending order, then we find the distance between point $i$ with the next point in the array, then we use this distance to multiply it with the number of time we have to cross this path which is $2 \times i \times (n - i)$, where $i$ would

be the number of points that are already visited before and $(n - i)$ would be the number of remaining points to be visited. The whole result $i \times (n - i)$ is then multiplied by 2 because we must again go back through the same path. Once we have the total distance from each point to the others, we find the average distance between all points. We can achieve this by dividing the total distance with the total number of paths from each point to the others by using $n \times (n - 1)$ formulas, where $N$ is the total number of points.

**The algorithm is fast enough** because it uses the sorting technique to sort the coordinates of each point. Sorting is a very efficient algorithm that has a time complexity of $O(n \times log(n))$. The algorithm also uses a relatively simple formula to calculate the distance between each pair of points.

Computational complexity of algorithm:

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

Source code:

```
#include<iostream>
#include<algorithm>

using namespace std;

long x[100001], y[100001];

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; i++) {
            cin >> x[i] >> y[i];
        }

        sort(x + 1, x + 1 + n);
        sort(y + 1, y + 1 + n);

        long long dis = 0;
        for (int i = 1; i < n; i++) {
            long long tmp = (long long) (x[i + 1] - x[i] + y[i + 1] - y[i]) * i * (n - i) * 2;
            dis += tmp;
        }

        dis /= ((long long) n * (n - 1));

        cout << dis << endl;
    }
}
```