Homework #1 Solution

Solution Magenta Correction Time: 10/10/2022 23:30
Solution Released Time: 10/10/2022 18:30
Blue Correction Date: 09/20/2022 14:20
Red Correction Date: 09/16/2022 21:00
Due Time: 2022/10/06 14:20
Contact TAs: `ada-ta@csie.ntu.edu.tw`

Instructions and Announcements

- There are **four programming problems** and **two hand-written problems**
- **Programming.** The judge system is located at `https://ada-judge.csie.ntu.edu.tw`. Please login and submit your code for the programming problems (i.e., those containing "Programming" in the problem title) by the deadline. NO LATE SUBMISSION IS ALLOWED.
- **Hand-written.** For other problems (also known as the "hand-written problems"), you should upload your answer to **Gradescope** as demonstrated in class. For each sub-problem, please label (on Gradescope) the corresponding pages where your work shows up. NO LATE SUBMISSION IS ALLOWED.
- **Collaboration policy.** Discussions with others are strongly encouraged. However, you should write down your solutions **in your own words**. In addition, for **each and every** problem you have to specify the references (e.g., the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem. You may get zero point due to the lack of references.
- **Tips for programming problems.** Since the input files for some programming problems may be large, please add
  - `std::ios_base::sync_with_stdio(false);`
  - `std::cin.tie(nullptr);`

  to the beginning of the main function if you are using **`std::cin`**.

Problem 1 - Omelet and the LEGO® Tower (Programming) (10 points)

Problem Description

Omelet loves playing with LEGO® bricks. His best friend Miku gave him N LEGO bricks as his birthday gift. These N bricks are not ordinary LEGO bricks. They are legendary Miku bricks! Each legendary Miku brick has a cute and unique photo of Miku on top of it. For convenience, we label these bricks from 1 to N.

Now, Omelet wants to take photos of the towers he built and send them to Miku so that he can prove his love to Miku.

To be more precise, Omelet wants to take **exactly** N photos, **the i-th of them being an *i-perfect Miku tower***, defined as follows.

- A *perfect Miku tower* consists of exactly N legendary Miku bricks stacking up one above another.
- An *i-perfect Miku tower* is a *perfect Miku tower* with the legendary Miku brick labeled i on the top of the tower.

Initially, all N bricks are on the playground in Omelet's house, and the Miku tower is empty. There are three types of operation Omelet can do.

1. Pick one brick labeled with X and place it on top of the current tower(which could be empty).
   Notice that he can only do this when the brick labeled with X is not on the Miku tower.
2. Put the brick on top of the Miku tower back to the playground.
   Notice that he can only do this when the Miku tower is not empty.
3. Take a photo of the current Miku tower.
   Notice that he can only do this when the current Miku tower is a *perfect Miku tower*.

Hating to move his fingers, Omelet only does these operations at most $4 \cdot 10^6$ times. Can you help him by telling him a sequence of operations that achieves his goal?

Input

One positive integer $1 \leq N \leq 10^5$ in a line.

Output

Print an integer M on the first line, indicating the total number of operations you use.

M lines follow, each line is in one of the forms

1. `PLACE` X, which corresponds to the first type of operation, and X is the label of the brick.
2. `POP`, which corresponds to the second type of operation.
3. `PHOTO`, which corresponds to the third type of operation.

You will get `Accepted` if and only if

- $1 \leq M \leq 4 \cdot 10^6$.

- Your operations are valid.
- Omelet achieves his goal by doing these operations in order.

Notice that you **don't need to minimize** the number of operations.

Test Group 0 (0 %)

- Sample Input.

Test Group 1 (5 %)

- $1 \leq N \leq 1000$

Test Group 2 (10 %)

- $1 \leq N \leq 10000$

Test Group 3 (65 %)

- $1 \leq N \leq 50000$

Test Group 4 (20 %)

- No additional constraints.

Sample Input 1

```
2
```

Sample Output 1

```
8
PLACE 2
PLACE 1
PHOTO
POP
POP
PLACE 1
PLACE 2
PHOTO
```

Sample Input 2

```
1
```

Sample Output 2

```
2
PLACE 1
PHOTO
```

Sample Input 3

```
1
```

Sample Output 3

```
7
PLACE 1
PHOTO
POP
PLACE 1
POP
PLACE 1
POP
```

Hint

- In Sample data 2, 3, both output are acceptable since they are valid.
- When using 'cout' to print a new line, **use `'\n'` instead of endl** since `endl` might be too slow, which leads to `Time Limit Exceed` verdict.

Solution

First we can put all the bricks in the stack such that the bricks are labeled from 1 to N from top to bottom.

Then define `solve(l, r)` as follows:

1. Now, the i-th brick from top to bottom is labeled $l + i - 1$.
2. first `solve(l, (l+r)/2)`
3. then take $r - l + 1$ bricks from the stack.
4. put all the bricks back such that bricks labeled with $(l + r)/2$ to r are on the top.
5. `solve((l+r)/2+1, r)`

and $T(N) = 2T(N/2) + 2N, T(1) = 1$ so $T(N) = O(N \log N)$, which is good enough for this problem.

Subtask 1

You can just brute force to take all bricks away from the tower.

Subtask 2

You can put bricks into K groups. For each group use brute force to solve inner group. After finish one group, take all N bricks out and then put this group to the bottom of the tower while the order of other bricks are preserverd.

Subtask 3

This is for some unexpected $N \log N$ solution.

Sample Code

```cpp

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

vector<string> op;
void place(int x) {
  op.push_back("PLACE " + to_string(x));
}
void pop() {
  op.push_back("POP");
}
```

```cpp
void photo() {
  op.push_back("PHOTO");
}
// now [l, r] are on the top of the stack
void solve(int l, int r) {
  if (l == r) {
    photo();
    return;
  }
  int m = l + r >> 1;
  solve(l, m);
  for (int i = l;i <= r;++i)
    pop();
  for (int i = l;i <= m;++i)
    place(i);
  for (int i = r;i > m;--i)
    place(i);
  solve(m+1, r);
}
int32_t main() {
  ios_base::sync_with_stdio(0), cin.tie(0);
  int N;
  cin >> N;
  for (int i = N;i >= 1;--i)
    place(i);

  solve(1, N);

  cout << op.size() << '\n';
  for (auto i : op)
    cout << i << '\n';
}
```

Test Data

You may download the test data for this problem here.

Problem 2 - Cash (Programming) (10 points)

Problem Description

Zisk is a wise man that knows everything. Little Z, on the other hand, is a normal guy.

Little Z is paid for his daily work by **cash**, whose amount is an integer between 1 and W (inclusive). After hearing an investment advice that "Do not hold too much **cash**," he hopes to minimize the amount of his cash by purchasing **gem**. There are N types of **gem**, and each has an unlimited supply. The price of the i-th types of **gem** is denoted by a_i.

Little Z uses a simple strategy to choose what types of **gem** to buy, one at a time: Given the amount of **cash** he currently holds, he buys the most expensive types of **gem** he can afford. He repeats the above action until he cannot buy anymore.

Little Z found that sometimes this simple strategy might be suboptimal. A strategy is optimal if and only if the remaining **cash** is the smallest among all possible strategies.

Hence, he wants to know, for every possible initial amount of **cash** (each integer from 1 to W), how much more **cash** he could have invested if he uses the optimal strategy.

Input

The first line of input contains two integers N, W, separated by a space. Each of the following N lines contains an integer represents a_i, the price of the i-th type of gem.

- $1 \leq N \leq 5000$
- $1 \leq W \leq 5000$
- $1 \leq a_i \leq 5000$

Output

Output W lines of non-negative integers, the i-th of which represents the difference between Little Z's plan and the optimal one when the initial amount of **cash** is i.

Test Group 0 (0 %)

- Sample Input.

Test Group 1 (30 %)

- $W \leq 50$

Test Group 2 (70 %)

- No additional constraints.

Sample Input 1

```
2 6
3
5
```

Sample Output 1

```
0
0
0
0
0
1
```

Sample Input 2

```
4 20
6
8
10
12
```

Sample Output 2

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
2
2
4
4
0
0
0
```

Hint

- In the first sample test case, $N = 2; W = 6; a = [3, 5]$. When the initial amount of **cash** is 6, the optimal plan is different from the plan which Little Z's strategy gives. Little Z's strategy will buy only the gem with price 5, while the optimal strategy is to buy the gem with price 3 twice.
- By the way, **cash** is a currency unit in the Grand ADA Kingdom.
- The prices of **gems** are not guaranteed to be sorted. The a_i in the input might be given in arbitrary order.

Solution

For the optimal solution, it is similar to the stamp problem.

Let dp_w represent the minimum amount of remaining **cash** when the initial amount is w, then we have a recurrence relation

$$dp_w = \begin{cases} \min\limits_{1 \le j \le n, w \ge a_j} (dp_{w-a_j}) & \text{if } \exists j, w \ge a_j \\ w & \text{if } \nexists j, w \ge a_j \end{cases}$$

To calculate Little Z's solution, we can use many methods to implement "Take the biggest". One can use a lookup table and binary search to speed up such process. The first subtask is for slow implementation of this part.

Sample Code

```cpp
#include<bits/stdc++.h>

using namespace std;

int main() {
    int N, W;
    cin >> N >> W;

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

    vector<int> dp(W + 1);

    // optimal solution (stamp problem)
    for (int i = 0; i <= W; i++) {
        dp[i] = i;
        for (int j = 0; j < N; j++)
            if (i >= a[j])
                dp[i] = min(dp[i], dp[i - a[j]]);
    }

    sort(a.begin(), a.end());

    // Little Z's solution (use binary search and a lookup table to calculate fast)
    vector<int> greedy(W + 1);
    for (int i = 0; i <= W; i++) {
        auto it = upper_bound(a.begin(), a.end(), i);
        if (it == a.begin())
            greedy[i] = i;
        else
            greedy[i] = greedy[i - *prev(it)];
    }

    for (int j = 1; j <= W; j++)
        cout << greedy[j] - dp[j] << '\n';

    return 0;
}
```

Test Data

You may download the test data for this problem here.

Problem 3 - Monsoon in Kingdom (Programming) (15 points)

Problem Description

As a clever student, ZCK exchanges to the ADA kingdom this semester. There are n cities in the kingdom, and there is a flight connecting each pair of cities. Due to the special geographical location, monsoon often affects flights in the fall. To ensure flight safety, an airplane cannot fly in some range of directions for a long time.

Specifically, for a pair of cities A and B, if the slope of $\overleftrightarrow{AB}$ is in a specific range $[l, r]$, then the flight connecting them cannot fly in the fall.

Professor Chen finds that ZCK is very smart, so she gives ZCK homework to calculate the number of flights that cannot fly in the fall. Can you help him to calculate the answer?

Input

The first line of the input contains an integer N. The second line contains four space-separated integers l_1, l_2, r_1, r_2, where $l = \frac{l_1}{l_2}$, $r = \frac{r_1}{r_2}$. Each of the following n lines has two space-separated integers x_i, y_i indicating the position of the airport in the i-th city.

- $1 \leq N \leq 10^6$
- $-10^9 \leq l_1, r_1 \leq 10^9$
- $1 \leq l_2, r_2 \leq 10^9$
- $0 \leq x_i, y_i \leq 10^9$
- $\frac{l_1}{l_2} \leq \frac{r_1}{r_2}$
- $(x_i, y_i) \neq (x_j, y_j), \forall i \neq j$

Output

Output an integer m denoting the number of flights that cannot fly in fall.

Test Group 0 (0 %)

- Sample Input

Test Group 1 (15 %)

- $N \leq 3000$

Test Group 2 (25 %)

- $l = -10^9,\ r = 0$
- $x_i \neq x_j, \forall i \neq j$

Test Group 3 (40 %)

- $l = -10^9$
- $x_i \neq x_j, \forall i \neq j$

Test Group 4 (20 %)

- No additional constraints

Sample Input 1

```
5
-1 1 1 1
1 1
0 0
2 0
2 2
0 2
```

Sample Output 1

```
8
```

Sample Input 2

```
5
-1000000000 1 0 1
3 9
0 0
12 0
6 3
5 3
```

Sample Output 2

```
7
```

Sample Input 2

```
5
-1000000000 1 2 3
0 2
1 0
2 3
3 0
4 2
```

Sample Output 2

```
8
```

Hint

- The slope of two cities located at $(x_A, y_A), (x_B, y_B)$ is $\dfrac{y_A - y_B}{x_A - x_B}$.
- It is undefined for the slope of two cities having the same x-coordinate. Hence, the flight that connects them will always not be in the range $[l, r]$.
- It seems like **Task Group 2** is similar to the **counting inversions problem**. You may want to search for it to get further hints.

Solution

First, let's take a look at this problem:

Inversion Number

Given an array a with length n, find the number of pairs (i, j) such that $i < j$ and $a_i > a_j$. This kind of pairs is called **inversion**, and the total number of inversions is called the **inversion number** of the array.

Let's use "Divide and Conquer" to count the inversion number.

For each subproblem $[l, r)$, we count the inversion number of the array $a[l], a[l+1], \ldots, a[r-1]$ and sort it.

Let $m = \lfloor \frac{l+r}{2} \rfloor$, and solve the subproblems $[l, m)$, $[m, r)$ first.

There are 3 kinds of inversion (i, j):
(1) $i < j < m$, which has been counted in the subproblem $[l, m)$
(2) $i < m \leq j$
(3) $m \leq i < j$, which has been counted in the subproblem $[m, r)$

Remember when we're doing merge sort, we merge two sorted arrays $a[l..m-1], a[m..r-1]$ by comparing the smallest elements that are not put into the result array b from each array, and put the smaller one into b. When putting $a[x]$ into b, if $x < m$, we count the number of elements in b that come from $a[m..r-1]$. This number is the number of inversion (x, j) of kind (2) since it is just the number of j such that $j \geq m$ and $a[j] < a[x]$. Therefore, when we finish merging the arrays, we count the number of inversion among all x, which is the number of inversion of kind (2).

Subtask 3

When $l = -10^9$, it is equivalent to $l = -\infty$. We only need to count the number of pairs of points A, B such that the slope of $\overleftrightarrow{AB}$ is less than or equal to r. The inversion number of the array whose i-th element is the y-intercept of the line with slope r passing through the i-th point after sorting the points by their x-coordinates is just the number of pairs of such (A, B).

Prefix Sum

The answer of this problem with $l = l', r = r'$ is the answer of subtask 3 with $r = r'$ minus that with $r = l'$ plus the number of pairs of points with slope l', which can be count easily.

Sample Code

```
#include <bits/stdc++.h>
#define int long long
#define kN 1000006
using namespace std;

struct P{
    int x, y;
    bool operator<(const P r)const{return x<r.x||x==r.x&&y<r.y;}
}a[kN];

int ans=0, n, b[kN], l1, l2, r1, r2, buf[kN];

void solve(int *l, int *r){ // return the number of ordered pairs (non equal) in [l, r)
    if(r-l<2)return;
    int b=buf-l, *m=l+(r-l>>1);
    solve(l, m), solve(m, r);
    for(int *i=l, *j=m, *k=l+b; k<r+b; ++k)*k=j==r||(i<m&&*i<*j)?*(i++):(ans+=i-l, *(j++)
    );
    for(int *i=l; i<r; ++i)*i=i[b];
}

void solve2(int *l, int *r){ // return the number of equal pairs in [l, r)
    int cnt=1;
    for(int *i=l+1; i<r; ++i)*i==i[-1]?++cnt:(ans+=cnt*(cnt-1)>>1ll, cnt=1);
```

```
    ans+=cnt*(cnt-1)>>1ll;
}

signed main(){
    scanf("%lld%lld%lld%lld%lld", &n, &l1, &l2, &r1, &r2);
    for(int i=0; i<n; ++i)scanf("%lld%lld", &a[i].x, &a[i].y);
    sort(a, a+n);
    for(int i=0; i<n; ++i)b[i]=r2*a[i].y-r1*a[i].x;
    solve(b, b+n), ans=-ans;
    for(int i=0; i<n; ++i)b[i]=l2*a[i].y-l1*a[i].x;
    solve(b, b+n), solve2(b, b+n);
    printf("%lld\n", ans);
}
```

Test Data

You may download the test data for this problem here.

Problem 4 - ZCK Loves Apex (Programming) (15 points)

Problem Description

ZCK likes to play Apex, but for some special reasons, he doesn't want others to know he is playing. To prevent from being discovered, ZCK used to lie about where he was and what he was doing every time he played Apex, but that didn't work since he usually played Apex in public places and would be discovered by others.

As the man on top of the world, ZCK has now found a solution for this problem - teleportation! Whenever ZCK is caught not staying at the place he claimed, he can teleport there and claim that he has just come out from the bathroom.

Though ZCK can control the power of teleportation, there are still some restrictions on teleportation. Following the magic rule in this world, there is a mysterious value for each location. For every two locations x and y, you can teleport from x to y if and only if the difference between the mysterious value for that two locations is the same as the minimum mysterious value you'll pass when you're walking from x to y.

To simplify this problem, assume that ZCK lives on a straight street with N locations where he can play Apex. For the i-th location on the street, there is a mysterious value a_i. ZCK can teleport between location x and y ($x < y$) if and only if $|a_x - a_y| = \min_{x \le i \le y} a_i$. ZCK needs your help to find out how many pairs of locations are there on the street that he can teleport between.

Input

For each testcase, there would be two lines: The first line contains an integer N representing the length of the straight street. The second line contains N integers $a_1, a_2, \cdots, a_n$ representing the mysterious value for each location.

- $1 \le N \le 5 \cdot 10^5$
- $0 < a_i < 100000$

Output

For each testcase, print a single integer representing the number of pairs that ZCK can teleport between.

Test Group 0 (0 %)

- Sample Input.

Test Group 1 (40 %)

- $1 \le N \le 2000$

Test Group 2 (60 %)

- No additional constraints.

Sample Input 1

```
5
5 2 3 1 4
```

Sample Output 1

```
4
```

Sample Input 2

```
10
1 5 3 4 3 5 9 6 3 6
```

Sample Output 2

```
7
```

Sample Input 3

```
20
3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4
```

Sample Output 3

```
31
```

Solution

Subtask 1

This can be done by $O(N^2)$ counting number of legal pairs

Subtask 2

We can use "Divide Conquer" to solve this problem.

For each subproblem $[l, r]$, we try to calculate the number of pairs (l', r') with $l \leq l' \leq r' \leq r$ that satisfied the condition provided. Let `solve(l, r)` be the answer of $[l, r]$, we know that `solve(l, r) = solve(l, mid) + solve(mid+1, r) + number of pairs that cross through the midpoint`

To calculate the number of pairs that cross through the midpoint, we can first consider the case that $a_l > a_r$, then reverse the array and do the same thing. We can use two counting array to maintain stuff in the right side, one for a_i and one for $\min_{mid+1 \leq j \leq i} a_i + a_i$. Then we can scan the array from mid to l and query the right counting array while scanning.

Notes that you should use vector as counting array instead of `unordered`$_m aptoprevent TLE$.

Sample Code

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

const int MAXN = 500000;
const int MAXC = 100000;
const int INF = 100000;
int N;
int a[MAXN+1];
long long ans;
vector <int> aR(MAXC*2), amR(MAXC*2);

void solve(int L, int R) {
  if (L == R) return;
  int mid = (L + R) / 2;
```

```
  solve(L, mid);
  solve(mid+1, R);
  int minL, minR;
  // Solve a_l > a_r
  minR = INF;
  vector <int> undo;
  // use vector with undo instead of map
  for (int i = mid+1; i <= R; i++) {
    minR = min(minR, a[i]);
    amR[a[i] + minR]++; undo.push_back(a[i] + minR);
  }
  int curR = mid+1;
  minL = minR = INF;
  for (int i = mid; i >= L; i--) {
    minL = min(minL, a[i]);
    while (curR <= R && minL <= a[curR]) {
      minR = min(minR, a[curR]);
      amR[a[curR] + minR]--; undo.push_back(a[curR] + minR);
      aR[a[curR]]++; undo.push_back(a[curR]);
      curR++;
    }
    ans += aR[a[i] - minL] + amR[a[i]];
  }
  for (int i : undo) aR[i] = amR[i] = 0;
  undo.clear();
}

signed main(){
  cin >> N;
  for (int i = 1; i <= N; i++) {
    cin >> a[i];
  }
  // assume a_l > a_r
  solve(1, N);
  // change it to a_r > a_l
  for (int i = 1; i <= N/2; i++) {
    swap(a[i], a[N+1-i]);
  }
  solve(1, N);
  cout << ans << '\n';
}
```

Test Data

You may download the test data for this problem here.

Problem 5 Asymptotic Notations (Hand-Written) (25 points)

Welcome to ADA2022! Wish you a nice trip in the world of algorithm.

For problem 5, please assume that:

$$n \in \mathbb{R}^+,$$

and also:

$$\log n \equiv \log_2 n,\ \lg n \equiv \log_2 n$$

You can also write $\log n$ and lg in your answer, which take 2 as base by default for convenience.

(a) Complexity (10 points)

Prove or disprove each of the following statements:

1. $\ln n = O(\sqrt{n})$ (1pt)
2. $928 \log_{830} n = \Theta(\ln n)$ (1pt)
3. $\Sigma_{i=1}^{n} i^3 = \Theta(n^4),\ n \in \mathbb{Z}^+$(1pt)
4. $n^{\pi}(\log n)^{112} = O(n^3\sqrt{n})$ (2pt)
5. $\lfloor \log \log n \rfloor ! = O(n)$ (2pt)

6(a). $\ln(n!) = \Theta(n \ln n),\ n \in \mathbb{Z}^+$(2pt)

6(b). prove $\ln(n!) - n \ln n + n = \Theta(\ln n)$ (1pt)

Please note that you can't use 6(b) to prove 6(a) without your proof of 6(b).

Solution

1. $\ln n = O(\sqrt{n})$

   Prove

   Let $n = x^2$, you get $2 \ln x = O(x)$

   Exist $c = 2,\ x_0 = 1$ s.t. $2 \ln x < cx,\ \forall x > x_0$

   Also exist $n_0 = x_0^2 = 1$ s.t. $\ln n < c\sqrt{n},\ \forall n > n_0$

   Let $f(x) = 2\sqrt{n} - \ln n,\ f'(x) = \frac{\sqrt{n}-1}{n}$

   So $f(x)$ is strictly increasing $\forall n > 1$

2. $928 \log_{830} n = \Theta(\ln n)$

   Prove

   $928 \log_{830} n = 928 \frac{\ln n}{\ln 830}$

   Exist $c_1 = \frac{927}{\ln 830},\ c_2 = \frac{929}{\ln 830},\ n_0 = 1$ s.t. $c_1 \ln n \leq 928 \log_{830} n \leq c_2 \ln n,\ \forall\ n > n0$

3. $\Sigma_{i=1}^{n} i^3 = \Theta(n^4)$

Prove

Lemma 1:$\Sigma_{i=1}^{n} i^3 = \frac{1}{4}n^2(n+1)^2$

Prove of Lemma 1

n = 1, $\Sigma_{i=1}^{1} i^3 = 1\frac{1}{4}1^2(1+1)^2 = 1$

Suppose it holds on n=k,

$\Sigma_{i=1}^{k} i^3 = \frac{1}{4}k^2(k+1)^2$

Then, when n=k+1,

$\Sigma_{i=1}^{k+1} i^3 = \Sigma_{i=1}^{k} i^3 + (k+1)^3 = \frac{1}{4}k^2(k+1)^2 + (k+1)^3 = \frac{k^2+4k+4}{4}(k+1)^2 = \frac{1}{4}(k+1)^2(k+2)^2$

Lemma 1 Proved by Induction.

Exist $c_1 = \frac{1}{8},\ c_2 = \frac{1}{2}, n_0 = 3\ s.t.\ c_1 n^4 < \frac{1}{4}n^2(n+1)^2 < c_2 n^4, \forall n > n_0$

$\text{Lemma}_1 - ref_1 : https://www.quora.com/How-can-you-prove-1-3-2-3-3-3-n-3-n-2-n-1-2-4$

4. $n^{\pi}(\log n)^{112} = O(n^3\sqrt{n})$

Prove

Lemma 2 : $(\log n)^k = O(n)$, for every constant $k \in R^+$

Proof of Lemma 2:

Recall L'hospital's Rule:Let f and g be differentiable on (a, b) and $g(x) \neq 0$ except at $c \in (a, b)$. Assume both $\lim_{x\to c} f(x), \lim_{x\to c} g(x) = \infty$(or both equals to 0).

We have $\lim_{x\to c} \frac{f'(x)}{g'(x)} = \lim_{x\to c} \frac{f(x)}{g(x)}$

$\lim_{n\to\infty} \frac{(\log n)^k}{n}$

Use L'Hospital $\lim_{n\to\infty} \frac{\frac{(\log n)^{k-1}}{n}}{1}$

Use L'Hospital for $\lceil k \rceil$ times $\lim_{n\to\infty} \frac{(\log n)^{\epsilon}}{n} = 0, \epsilon \in (-1, 0]$

thus proved $(\log n)^k = O(n)$

Divide both side by n^{π} and we get $(\log n)^{112}, n^{3.5-\pi}$

Use Lemma 2, we know $(\log n)^{\frac{112}{3.5-\pi}} = O(n)$

So proved $n^{\pi}(\log n)^{112} = O(n^3\sqrt{n})$

5. $\lfloor \log\log n \rfloor ! = O(n)$

Prove

Let $n = 2^{2^x}$, we get $x! = O(2^{2^x})$

Prove $\lfloor x \rfloor ! = O(x!)$ and $x! = O(2^{2^x})$ to get $\lfloor x \rfloor ! = O(2^{2^x})$

(Big-O relation is transitive so is "greater relation")

$\lfloor x \rfloor ! = O(x!)$ Let $c = 1, n_0 = 0$, obviously we have $\lfloor x \rfloor ! \leq c \times x!, \forall n > n_0$

proved $\lfloor x \rfloor ! = O(x!)$

prove $x! = O(2^{2^x})$

$x! = \Pi_{i=1}^{x} i$

$2^{2^x} \geq \Pi_{i=1}^{x} 2^i = 2^{\frac{(1+x)x}{2}}, \forall x > 0$
$i < 2^i, \forall i \in Z^+ \rightarrow \Pi_{i=1}^{x} i < \Pi_{i=1}^{x} 2^i < \Pi_{i=1}^{x} 2^i \rightarrow x! < 2^{2^x}$
proved $x! = O(2^{2^x})$
Also proved $\lfloor \log \log n \rfloor ! = O(n)$

6(a). $\ln(n!) = \Theta(n \ln n),\ n \in \mathbb{Z}^+$

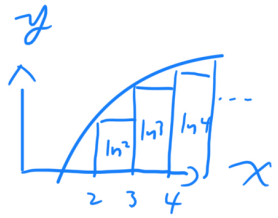
$\ln(n!) = \Sigma_{i=1}^{n} \ln i$
$\Sigma_{i=1}^{n} \ln i < \Sigma_{i=1}^{n} \ln n = n \ln n$
$\ln(n!) = O(n \ln n)$ ... (1.)
$\Sigma_{i=1}^{n} \ln i = \ln 1 + \ln 2 ... \ln n > \Sigma_{i=\lfloor \frac{n}{2} \rfloor}^{n} \ln i > \Sigma_{i=\lfloor \frac{n}{2} \rfloor}^{n} \ln \lfloor \frac{n}{2} \rfloor =$
$\lfloor \frac{n}{2} \rfloor \ln \lfloor \frac{n}{2} \rfloor = \Omega(n \ln n)$ ... (2.)
By (1.) (2.), we get $\ln(n!) = \Sigma_{i=1}^{n} \ln i$

6(b). $\ln(n!) - n \ln n + n = \Theta(\ln n)$

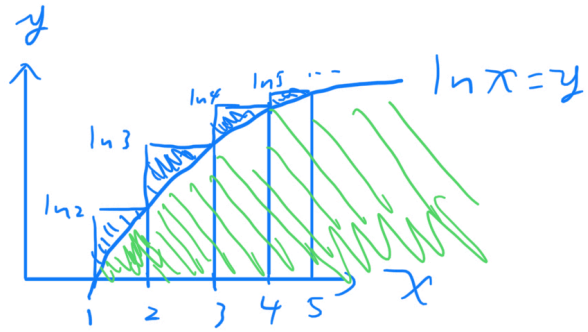
Use (3.20) in Textbook
$n! = \sqrt{2\pi n}(\frac{n}{e})^n e^{\alpha_n}$
$\ln n! = \frac{1}{2} \ln 2\pi n + n \ln n - n \ln e + \alpha_n \ln e$
$= \frac{1}{2} \ln n + n \ln n - n + f(n)$
$c_1 = \frac{1}{2} \ln 2\pi < f(n) = \frac{1}{2} \ln 2\pi + \alpha_n < \ln 2\pi + 1 = c_2$
$\ln n! - n \ln n + n = \frac{1}{2} \ln n + c = \Theta(\ln n)$

Solution

Sol 2 for 5(a.)6(b.)
Without using Stirling's approximation



$\ln(n!) - n \ln n + n = \Theta(\ln n)$
$\ln(n!) = \Sigma_{i=1}^{n} \ln i = \Sigma_{i=1}^{n-1} \ln i + \ln n < \int_1^n \ln x \ dx + \ln n$
$= x \ln x - x + C|_{x=1}^{n} + \ln n = n \ln n - n + 1 + \ln n$
$\ln(n!) - n \ln n + n < n \ln n - n + 1 + \ln n - n \ln n + n = 1 + \ln n$
$\exists c = 2, n_0 = e, \text{s.t.} 1 + \ln n < c \ln n, \forall n > n_0$
$\rightarrow 1 + \ln n = O(\ln n)$
By Big-O relation is transitive, $\ln(n!) - n \ln n + n = O(\ln n)$...(1)



Consider this plot, sum of the n rectangles' area $R_a = \ln(n!) = \Sigma_{i=1}^{n} \ln i$
the green area under the curve $G_a = \int_1^n \ln x dx = n \ln n - n + 1$
By $\frac{d}{dx} \ln x = \frac{1}{x}$, y th blue shaded area is at least $\frac{1}{2(y+1)}$
Sum of the blue shaded area is at least $\Sigma_{i=2}^{n} \frac{1}{2i}$
$R_a - G_a > \Sigma_{i=2}^{n} \frac{1}{2i}$
$R_a = \ln(n!) > \Sigma_{i=2}^{n} \frac{1}{2i} + G_a = \frac{\ln(n+1)}{2} + n \ln n - n + 1$
$R_a - n \ln n + n > \frac{\ln(n+1)}{2} + 1$
$\exists c = \frac{1}{2}, n_0 = 2, \text{s.t.} \frac{\ln(n+1)}{2} + 1 > c \ln n, \forall n > n_0$
$\rightarrow \frac{\ln(n+1)}{2} + 1 = \Omega(\ln n)$
By Big-Omega relation is transitive, $R_a - n \ln n + n = \ln(n!) n \ln n + n = \Omega(\ln n)$ ... (2)
By (1),(2). We proved $\ln(n!) - n \ln n + n = \Theta(\ln n)$

(b) Fill-in (9 points)

Please fill in the **numbers** according to their complexity in correct order.
Every pair $f(n) < g(n)$ in your answer means $f(n) = O(g(n))$.
You don't need to prove your answer in (b)!

- Points $= 9 - 0.6 \times \{(i, j) \mid i < j, answer[i] > answer[j]\}$
- **If your answer is not a permutation of [1,6], you would get 0 point from (b)!**

1. $\sqrt{n^{2.048}}$
2. $n \log(\lfloor n \rfloor !)$
3. 2^n
4. $n \log n$
5. n^k, for a large constant $k \to \infty,\ k \in \mathbb{R}^+$
6. $(\log n)^{1.0001^n}$

___ < ___ < ___ < ___ < ___ < ___

Fun fact: Every choice that is smaller than 5. is said to be "polynomially bounded".

Solution

$$n \log n < \sqrt{n^{2.048}} < n \log(n!) < n^k (k \in \mathbb{R}^+) < 2^n < (\log n)^{1.0001^n}$$

Answer is "412536"

Note: $log(n!) = \Theta(n \log n)$

(c) Recursion (6 points)

In this section, we assume $T(1) = 1$.

Please give the tightest upper bound for each subproblem and justify your answer with a brief proof.

1. $T(n) = 26T\left(\frac{n}{9}\right) + n^{1.5}$ (3pt)
2. $T(n) = 3n + T\left(\frac{2n}{3}\right) + T\left(\frac{2n}{9}\right)$ (3pt)

Solution

1. $T(n) = 26T\left(\frac{n}{9}\right) + n^{1.5}$

   Use Master Theorem

   $26T(n/9) + n^{1.5}$

   $1.5 = log_9 27 > log_9 26$

   $T(n) = O(n^{1.5})$

2. $T(n) = 3n + T\left(\frac{2n}{3}\right) + T\left(\frac{2n}{9}\right)$

   Use Recursion-Tree Method

   1st expansion: $3n$

   2nd expansion: $\frac{8}{3}n$

   3rd expansion: $\frac{64}{27}n$

   ...

   x th expansion is $\frac{8}{9}^{x-1} * 3n$

   Upper bound of total cost is $S = \Sigma_{i=0}^{\infty} 3n * \frac{8}{9}^{i} = 27n$

   $T(n) = O(n)$

Problem 6 (Hand-Written) (25 points)

Note: In this problem, if you use any data structures or algorithms that haven't been taught in class, you need to explain what they are and prove their complexity, if needed. This problem can be solved with the knowledge taught in class. Therefore, we recommend you answer it with the knowledge taught in class.

In the United Kingdom of ADA and DSA, a pandemic called **Covy-19** spreads rapidly. To protect the residents, the king of ADA collects some **gene** from **Covy-19**. Formally, one **gene** is represented by a string with length k. Furthermore, we say two genes r, s are **similar** if and only if there exists at most one i $(1 \leq i \leq k)$ such that $r[i] \neq s[i]$.

To recruit the best researchers in the kingdom, the king of ADA held a competition with up to 100,000,000 ADA (1 ADA = 0.4626 USD as I wrote this) worth of rewards. You, as an intelligent engineer, need the reward to repair your liver, so you decide to participate in this competition. Due to the lack of biological knowledge, the only tool you can rely on is a **gene comparator** that can compare two characters (you can assume that the characters have some ordering) in constant time.

As you arrive at the battlefield, the king started to announce the goal: Given n **gene**s, your **final task** is to find the number of pairs that are **similar**. Along with the words, you suddenly realized that you are teleported to a dark room.

(a) (5 points)

For the following statements, you only need to answer **T**(True) or **F**(False). No proofs or reasons need to be provided.

Assume that r, s are **similar** strings with length k and $1 \leq i(\in \mathbb{N}) < k$. Define $r_1 = r[1..i], r_2 = r[(i+1)..k]$, and define s_1, s_2 accordingly.

1. If $r_1 = s_1$, then $r_2 = s_2$
2. If $r_1 \neq s_1$, then $r_2 = s_2$
3. If $r_1 = s_1$, then $r_2 \neq s_2$
4. If $r_1 = s_1$, then r_2, s_2 are **similar**
5. If $r_1 \neq s_1$, then r_2, s_2 are **similar**

Solution

1. F
2. T
3. F
4. T
5. T

These problems are too easy for you. After you answered the questions, some **gene**s and a screen with "separation test" appeared in front of you.

(b) (5 points)

Note: according to policy, you need to prove the correctness and time complexity for (b).

Given n strings with length k, please design an $O(nk \log n)$ algorithm to divide them into groups, where

- each string in the same group are identical
- if two strings are in different group, then they are different.

Solution

Sort them in $O(nk \log n)$. Then, scan from the first string. Make s_i and s_{i+1} the same group if and only if $s_i = s_{i+1}$ (using naive comparison), which takes $O(nk)$. The total complexity is $O(nk \log n)$. The correctness comes from the fact that, same strings have the same lexicographical order, and vice versa.

You finally separated them and took some **gene** samples. For the following problems, you can assume that the n **gene**s are distinct.

(c) (5 points)

Design a **divide-and-conquer** algorithm to solve the **final task**. No correctness or complexity analysis needed. (Hint: **Problem (a) & (b)**)

This algorithm needs to run in $O(nk \log n \log k)$, although you don't need to prove it in this subproblem. Also, we recommend you to explain the algorithm in words. If you want to answer it in pseudo code, make sure it is correct, readable (comment or explain if needed), and **less than 35 lines**.

Solution

We apply divide and conquer on k. The basic idea is statement (a)-4. (For formal proof, see (d)) Suppose the strings are stored in `arr`, and we use the convention that `arr[1, 3, 5][2..6]` means the subarray formed by the substring `s[2..6]` for s in the 1st, 3rd, 5th substring. Now we briefly explain the functions in the pseudo code: (index style: 1-based and inclusive)

1. `GroupUp(index_arr, l, r)` group the strings `arr[index_arr][l..r]` using the algorithm in **(b)**.
2. `Solve(index_arr, l, r)` return the answer to the subproblem (i.e. how many similar pairs) `arr[index_arr][l..r]`. It uses divide and conquer and solve recusively for each group.

```
Func GroupUp(index_arr, l, r):
    Use the algorithm mentioned in (b);
    ans ← the groups ;          // vector of vector, or linked list of linked list
return ans
Func Solve(index_arr, l, r):
    if l == r then
        size ← length of index_arr;
        ans ← ans + size · (size - 1) / 2;
    else
        m ← ⌊ (l + r) / 2 ⌋;
        ans ← 0;
        groups ← GroupUp(index_arr, l, m);
        for group in groups do
            ans ← ans + Solve( group, m + 1, r );
        end
        groups ← GroupUp(index_arr, m + 1, r);
        for group in groups do
            ans ← ans + Solve( group, l, m );
        end
    end
return ans
```

The answer is given by `Solve(`$\{1, 2, \ldots n\}, 1, k$`)`.

(d) (3 points)

Prove the correctness for **(c)**.

Solution

We use the following facts: (define $r, r_1, r_2, s, s_1, s_2, k$ as in **(b)**)

1. r, s (with length k) are **similar** but distinct if and only if one of the following holds:
   - $r_1 = s_1$ and r_2, s_2 are **similar** but distinct
   - $r_2 = s_2$ and r_1, s_1 are **similar** but distinct
2. any string with length 1 are **similar**.

We now prove the correctness for `Solve()` by (strong) induction on $k = l - r$.
Base case: by fact 2, since each strings are similar, the answer is $n(n-1)/2$.
Inductive step: Suppose `Solve()` returns the correct answer for $k < k'$, then for $k = k'$, observe that:

1. each similar pair is counted at least once: suppose r, s are similar. WLOG let $r_1 = s_1$, then they will be in the same group in `GroupUp(index_arr, l, m)` and, since r_2, s_2 are similar, it will be counted in `Solve(group, m + 1, r)`.
2. each similar pair is counted at most once: suppose r, s are similar. Notice that they can belongs the same group for only one half. However, since `Solve()` only count each similar pair once, the pair (r, s) cannot be count twice.

Therefore, each similar pair will be counted exactly once, hence `Solve()` returns the correct answer for $k = k'$, as desired.

However, since you are very busy, you are not sure about how much time it will take. Therefore, you decide to analyze it first.

(e) (2 points)

Let $f(x) = x \log x$, and $a, b > 1$, prove that:

$$f(a) + f(b) \leq f(a + b)$$

Solution

$$\begin{aligned} a \log a + b \log b &\leq a \log(a + b) + b \log(a + b) \\ &= (a + b) \log(a + b) \end{aligned}$$

(f) (5 points)

Prove that the algorithm in **(c)** runs in $O(nk \log n \log k)$.

Solution

Define $f(n, k)$ as the time for running the algorithm. Define $g(n, k) = cnk \log n \log k$ for some large enough c.
We use strong induction to show that $f(n, k) \leq g(n, k)$.
Base case: $f(n, 1)$ is $O(n)$, therefore $O(nk \log n \log k)$.
Suppose that the size of groups are $l_1, l_2, \ldots l_L$ and $r_1, r_2, \ldots r_R$ when cut by $k/2$. Then

$$\begin{aligned} f(n, k) &= nk \log n + \sum_{i=1}^{L} f(l_i, k/2) + \sum_{i=1}^{R} f(r_i, k/2) \\ &\leq (n \log n)k + \sum_{i=1}^{L} (c(k/2) \log(k/2))(l_i \log l_i) + \sum_{i=1}^{R} (c(k/2) \log(k/2))(r_i \log r_i) \\ &\leq (n \log n)k + 2cn \log n(k/2) \log(k/2) \\ &\leq cn \log nk \log k \end{aligned}$$

The last inequality holds when $2c \log 2 \geq 1$. Since we can choose c to be large enough, this inequality is correct.

You got the champion and is about to receive the award. Suddenly, you notice that something is wrong: you finished much faster than you expected! Although this is a great news, someone suspected that you were cheating. To prove your innocence, you use all your power in math and finally figured out the reason.
Warning: the following problem might use out-of-class knowledge and might be difficult.
Note: Although (g) is hard, you can get at most 2 partial credits if you write a correct reason why the bound in **(f)** could not be tight.
Another note: you are allowed to reference (without proof, but make sure it is correct) out-of-class theorems in subproblem (g).

(g) (bonus 5 points)

Prove that the algorithm in **(c)** runs in $O(nk \log n)$.

Solution

In **(f)**, we overestimate the answer because we didn't consider the reduce of n. Briefly speaking, it's not possible to maintain subproblems of size $f(n, k/2)$ for both sides simultaneously.
When all strings are halved, we can denote each string by a tuple $s = (\alpha, \beta)$, where α is the group it belongs to in the first half, and define β similarly. Notice that each tuple are distinct.
Define $\mathbb{A}$ as the collection of the groups, and $d(\alpha)$ as the size of group $\alpha \in \mathbb{A}$ in the first half. Define $\mathbb{B}, \beta, d(\beta)$ similarly. Moreover, define $\mathbb{S}$ as the set of the (tuple representation) of the string.

Lemma 1.

$$\sum_{(\alpha,\beta)\in\mathbb{S}} d(\alpha)d(\beta) \le n^2$$

pf. Observe that $\mathbb{S} \subseteq \mathbb{A} \times \mathbb{B}$.

$$\sum_{(\alpha,\beta)\in\mathbb{S}} d(\alpha)d(\beta) \le \sum_{(\alpha,\beta)\in\mathbb{A}\times\mathbb{B}} d(\alpha)d(\beta) = \left(\sum_{\alpha\in\mathbb{A}} d(\alpha)\right)\left(\sum_{\beta\in\mathbb{B}} d(\beta)\right) = n^2$$

□

Lemma 2.

$$\sum_{\alpha\in\mathbb{A}} d(\alpha)\log d(\alpha) + \sum_{\beta\in\mathbb{B}} d(\beta)\log d(\beta)$$

pf. The key observation is the first equation. Notice that if we assign (α, β) with a "score" $d(\alpha)\log d(\alpha)$, then the sum of LHS is equal to the sum of the score over $\mathbb{S}$.

$$\begin{aligned}\sum_{\alpha\in\mathbb{A}} d(\alpha)\log d(\alpha) + \sum_{\beta\in\mathbb{B}} d(\beta)\log d(\beta) &= \sum_{(\alpha,\beta)\in\mathbb{S}} \log(d(\alpha)) + \log(d(\beta)) \\ &= \sum_{(\alpha,\beta)\in\mathbb{S}} \log(d(\alpha)d(\beta)) \\ &= \log(\prod_{(\alpha,\beta)\in\mathbb{S}} d(\alpha)d(\beta)) \\ &\le \log(n^n) = n\log n\end{aligned}$$

The last line comes from A.M.-G.M. inequality and **Lemma 1**. □

Now, to prove the complexity, we only start from the recursion formula in the solution of (f), and instead we suppose $f(n, k') \le cnk' \log n$ for $k' = k/2$:

$$\begin{aligned}f(n,k) &= nk\log n + \sum_{i=1}^{L} f(l_i, k/2) + \sum_{i=1}^{R} f(r_i, k/2) \\ &= nk\log n + c(k/2)(\sum_{i=1}^{L} l_i \log l_i + \sum_{i=1}^{R} r_i \log r_i) \\ &\le nk\log n + ck/2 \cdot n\log n \\ &\le cnk\log n\end{aligned}$$

which completes the proof.

Note:
1. Some claimed that 6(e) is not tight, so that the bound should be $o(nk \log n \log k)$. Although I'll give 1 point to you, I would like to remind that this is not correct. For example:

$$1 + 2 + ... + n = O(n^2)$$

since

$$1 + 2 + ... + n \leq 2n + 2n + ... + 2n = 2n^2$$

Although these bound are not tight, they still give a (asymptotically) tight bound.
2. Some uses hashtable. However, it does not provide constant time insertion/finding **in the worse case**. Also, since we only give you a character comparator, it's not that easy to hash a string.
3. When you try to write down a long proof (such as in this problem), it is often a good practice to break down in claims, statements or lemmas. This not only help the graders but also help you to check the logics.