

ĐÁP ÁN ĐỀ CHÍNH THỨC
(Đáp án gồm có 06 trang)

Tổng quan đề thi:

Tên bài	Tệp chương trình	Tệp dữ liệu vào	Tệp kết quả
Bài 1: Gấp hạc giấy	HACGIAY.*	HACGIAY.INP	HACGIAY.OUT
Bài 2: Số nguyên tố lớn nhất	NGTMAX.*	NGTMAX.INP	NGTMAX.OUT
Bài 3: Ăn khế trả vàng	TRAVANG.*	TRAVANG.INP	TRAVANG.OUT
Bài 4: Xếp hàng	XEPHANG.*	XEPHANG.INP	XEPHANG.OUT

Ghi chú: Dấu * đại diện cho phần mở rộng, tùy theo ngôn ngữ lập trình có thể là PAS hoặc CPP hoặc PY. Thời gian thực hiện chương trình không quá 01 giây, bộ nhớ không quá 1024MB.

Bài 1. (5,0 điểm) GẤP HẠC GIẤY

Mỗi ngày gấp được tối đa k con hạc. Suy ra, trong p ngày ta sẽ gấp được $p \times k$ con hạc. Ta cần tìm số ngày ít nhất để gấp đủ n con hạc nên ta sẽ xét n chia k là số ngày cần để gấp $\lfloor \frac{n}{k} \rfloor \times k$ con hạc. Vậy số hạc còn thiếu là $n - \lfloor \frac{n}{k} \rfloor \times k$ dễ thấy $0 \leq n - \lfloor \frac{n}{k} \rfloor \times k < k$. Nếu $n - \lfloor \frac{n}{k} \rfloor \times k > 0$ thì ta sẽ cần thêm một ngày để gấp đủ số hạc còn thiếu. Vậy kết quả của bài toán là $\frac{n}{k} + (n \% k > 0)$.

Độ phức tạp: $O(1)$.

Chương trình tham khảo bằng ngôn ngữ lập trình C++:

```
#include <bits/stdc++.h>
#define int long long

using namespace std;

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
    freopen("HACGIAY.INP", "r", stdin);
    freopen("HACGIAY.OUT", "w", stdout);
    int n, k; cin >> n >> k;
    cout << n/k+(n%k > 0);
    return 0;
}
```

Chương trình tham khảo bằng ngôn ngữ lập trình Python:

```
import sys

def main():
    sys.stdin = open("HACGIAY.INP", "r")
    sys.stdout = open("HACGIAY.OUT", "w")
    n, k = map(int, input().split())
    print(n//k+(n%k > 0))

if __name__ == "__main__":
    main()
```

Bài 2. (5,0 điểm) SỐ NGUYÊN TỐ LỚN NHẤT

Bài toán này ta phải duyệt tất cả các số và kiểm tra từng số có phải số nguyên tố hay không, để có thể tìm ra số nguyên tố lớn nhất trong các số đó. Như vậy độ phức tạp và tối ưu thuật toán cho bài toán này chủ yếu nằm ở việc **kiểm tra số nguyên tố**.

Nhắc lại: Số nguyên tố là một số nguyên dương khác 1 chỉ có đúng hai ước là 1 và chính nó.

⇒ Những số nguyên **không lớn hơn 1** đều **không** phải là số nguyên tố.

Như vậy, ta sẽ bỏ qua tất cả các số nguyên **không lớn hơn 1** trong thao tác duyệt.

❖ **Subtask 1: Có 60% số điểm tương ứng 60% số test có $0 < n \leq 10^3$ và $|a_i| \leq 10^3$.**

Để kiểm tra p có phải là số nguyên tố hay không, ta sẽ duyệt các số từ 1 đến p và đếm số lượng số chia hết cho p (nói cách khác là đếm số lượng ước của p). Nếu số lượng số chia hết cho p đếm được đúng bằng 2 thì số nguyên p là một số nguyên tố.

Độ phức tạp: $O(n \times a_i) \approx O(10^3 \times 10^3)$.

Code minh họa hàm kiểm tra số nguyên tố bằng ngôn ngữ lập trình C++

```
bool isPrime(int p) {
    int cnt = 0;
    for (int i = 1; i <= p; ++i)
        if (p%i == 0)
            ++cnt;
    return cnt == 2;
}
```

Code minh họa hàm kiểm tra số nguyên tố bằng ngôn ngữ lập trình Python

```
def isPrime(p):
    cnt = 0
    for i in range(1, p+1):
        if p%i == 0:
            cnt = cnt+1
    return cnt == 2
```

❖ **Subtask 2: Có 20% số điểm tương ứng 20% số test có $0 < n \leq 10^3$ và $|a_i| \leq 10^6$.**

Từ cách phân tích một số nguyên p thành tích hai số $a \times b$. Ta có thể thấy rằng, chỉ cần xét các ước trong khoảng từ 1 đến $\sqrt{p}$ là có thể duyệt toàn bộ các ước của p . Như định nghĩa đã nhắc lại, p là một số nguyên tố khi p chỉ chia hết cho 1 và chính nó. Do đó, nếu p chia hết cho một số trong khoảng $2 \rightarrow \sqrt{p}$ thì p không phải là một số nguyên tố.

Cách tối ưu: Chỉ duyệt từ 2 đến $\sqrt{p}$, nếu p chia hết cho một trong các số đó thì p không phải là một số nguyên tố.

Độ phức tạp: $O(n \times \sqrt{a_i}) \approx O(10^3 \times 10^3)$.

Code minh họa hàm kiểm tra số nguyên tố bằng ngôn ngữ lập trình C++

```
bool isPrime(int p) {
    for (int i = 2; i*i <= p; ++i)
        if (p%i == 0)
            return false;
    return true;
}
```

Code minh họa hàm kiểm tra số nguyên tố bằng ngôn ngữ lập trình Python

```
def isPrime(p):
    for i in range(2, int(math.sqrt(p))+1):
        if p%i == 0:
            return False
    return True
```

❖ **Subtask 3: Có 20% số điểm tương ứng 20% số test có $10^3 < n \leq 10^6$ và $|a_i| \leq 10^6$.**

Do các số cần kiểm tra không quá 10^6 , nên ta có thể sử dụng **sàng số nguyên tố** để kiểm tra nhanh một số có phải là số nguyên hay không trong độ phức tạp $O(1)$ với chi phí xây dựng ban đầu là $O(10^6 \times \log(\log 10^6))$.

Chương trình tham khảo bằng ngôn ngữ lập trình C++:

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

using namespace std;

bool isPrime[1000001];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
```

```

freopen("NGTMAX.INP", "r", stdin);
freopen("NGTMAX.OUT", "w", stdout);
isPrime[1] = 1;
for (int i = 2; i*i <= 1e6; ++i)
    if (!isPrime[i])
        for (int j = i*i; j <= 1e6; j += i)
            isPrime[j] = true;
int n, res = 0; cin >> n;
for (int i = 1, a; i <= n; ++i) {
    cin >> a;
    if (a > 1 && !isPrime[a])
        res = max(res, a);
}
cout << res;
return 0;
}

```

Chương trình tham khảo bằng ngôn ngữ lập trình Python:

```

import sys

def main():
    sys.stdin = open("NGTMAX.INP", "r")
    sys.stdout = open("NGTMAX.OUT", "w")
    isPrime = [False]*(10**6+1)
    isPrime[1] = True
    for i in range(2, int(1e6**0.5)+1):
        if not isPrime[i]:
            for j in range(i*i, 10**6+1, i):
                isPrime[j] = True
    n, res = int(input()), 0
    a = map(int, input().split())
    for i in a:
        if i > 1 and not isPrime[i]:
            res = max(res, i)
    print(res)

if __name__ == "__main__":
    main()

```

Bài 3. (5,0 điểm) ĂN KHÉ TRẢ VÀNG

❖ **Subtask 1:** Có 70% số điểm tương ứng 70% số test có $0 < n \leq 10^2$.

Duyệt trâu các vị trí hai đầu mút l, r và dùng một vòng lặp nữa để tính tổng các phần tử trong đoạn hai đầu mút đó.

Độ phức tạp: $\approx O(n^3)$.

Code minh họa bằng ngôn ngữ lập trình C++

```

for (int l = 1; l <= n; ++l)
    for (int r = l; r <= n; ++r) {
        int s = 0;
        for (int i = l; i <= r; ++i)
            s += a[i];
        cnt += s <= m;
    }

```

Code minh họa ngôn ngữ lập trình Python

```

for l in range(1, n+1):
    for r in range(l, n+1):
        s = 0
        for i in range(l, r+1):
            s += a[i]
        cnt += s <= m

```

❖ **Subtask 2:** Có 20% số điểm tương ứng 20% số test có $10^2 < n \leq 10^3$.

Sử dụng kỹ thuật mảng cộng dồn để tính tổng đoạn phần tử từ l đến r trong $O(1)$.

Xây dựng mảng cộng dồn: $for\ i = 1 \rightarrow n: pre_i = pre_{i-1} + a_i$.

Tổng đoạn phần tử từ l đến r được tính bằng: $pre_r - pre_{l-1}$.

Độ phức tạp: $\approx O(n^2)$.

Code minh họa bằng ngôn ngữ lập trình C++

```
for (int l = 1; l <= n; ++l)
    for (int r = l; r <= n; ++r)
        cnt += pre[r]-pre[l-1] <= m;
```

Code minh họa ngôn ngữ lập trình Python

```
for l in range(1, n+1):
    for r in range(l, n+1):
        cnt += pre[r]-pre[l-1] <= m
```

❖ **Subtask 3: Có 10% số điểm tương ứng 10% số test có $10^3 < n \leq 10^5$.**

Sau khi áp dụng mảng cộng dồn, dễ thấy một đoạn từ $l + 1$ đến r có tổng không quá m được biểu diễn là: $pre_r - pre_l \leq m \Leftrightarrow pre_r - m \leq pre_l$.

Dễ thấy r và l là các số chưa biết, m là số đã biết. Ta sẽ duyệt một số chưa biết để tìm số chưa biết còn lại.

Duyệt $r = 1 \rightarrow n$, khi này ta đã biết r và m còn l chưa biết. Tương đương đã biết giá trị của $pre_r - m$. Như vậy chỉ còn đếm số lượng số l trong đoạn $0 \rightarrow r - 1$ có $pre_l \geq pre_r - m$.

Do $0 < a_i \leq 10^9$, nên mảng pre là một mảng tăng dần. Suy ra, ta có thể áp dụng tìm kiếm nhị phân để đếm các số l thỏa mãn.

Tìm trong đoạn $pre[0 \rightarrow r - 1]$ vị trí idx nhỏ nhất có giá trị $\geq pre_r - m$, số lượng l thỏa mãn lúc này là: $r - idx$. Nếu không tồn tại idx thì không có l nào thỏa mãn.

Độ phức tạp: $O(n \times \log_2 n)$

Chương trình tham khảo bằng ngôn ngữ lập trình C++:

```
#include <bits/stdc++.h>
#define int long long

using namespace std;

int pre[1000001];

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
    freopen("TRAVANG.INP", "r", stdin);
    freopen("TRAVANG.OUT", "w", stdout);
    int n, m, res = 0; cin >> n >> m;
    for (int i = 1, a; i <= n; ++i) {
        cin >> a; pre[i] = pre[i-1]+a;
        int it = lower_bound(pre, pre+i, pre[i]-m)-pre;
        if (it != i) res += i-it;
    }
    cout << res;
    return 0;
}
```

Chương trình tham khảo bằng ngôn ngữ lập trình Python:

```
import sys, bisect

def main():
    sys.stdin = open("TRAVANG.INP", "r")
    sys.stdout = open("TRAVANG.OUT", "w")
    n, m = map(int, input().split())
    pre, res = [0]*(n+1), 0
    a = list(map(int, input().split()))
    for i in range(1, n+1):
        pre[i] = pre[i-1]+a[i-1]
        it = bisect.bisect_left(pre[:i], pre[i]-m)
```

```

        if it != i:
            res += i-it
    print(res)
if __name__ == "__main__":
    main()

```

Bài 4. (5,0 điểm) XẾP HÀNG

❖ **Subtask 1: Có 70% số điểm tương ứng 70% số test có $1 \leq n \leq 10^3$.**

Duyệt trâu từng cặp học sinh ở các vị trí i và j ($j < i$), đếm các cặp có $a_j < a_i$.

Độ phức tạp: $\approx O(n^2)$.

Code minh họa bằng ngôn ngữ lập trình C++

```

for (int i = 1; i <= n; ++i)
    for (int j = 1; j < i; ++j)
        cnt += a[j] > a[i];

```

Code minh họa ngôn ngữ lập trình Python

```

for i in range(1, n+1):
    for j in range(1, i):
        cnt += a[j] > a[i]

```

❖ **Subtask 2 & 3: Có 30% số điểm tương ứng 30% số test có $10^3 < n \leq 10^6$.**

Cải tiến: Với mỗi a_i , ta sử dụng cấu trúc dữ liệu đặc biệt *Fenwick Tree* (Cây chỉ số nhị phân) để đếm số lượng cặp số $a_j > a_i$ có $j < i$.

Cây chỉ số nhị phân định nghĩa các hàm:

- $get(i)$: Số lượng số có giá trị $\leq i$.
- $update(i)$: Tăng một số có giá trị bằng i .

Mã giải:

```

for i = 1 → n:
    res += get(106) - get(a[i])
    update(a[i])

```

Một cách tối ưu hơn:

```

for i = n → 1:
    res += get(a[i] - 1)
    update(a[i])

```

Độ phức tạp: $\approx O(n \times \log_2 n)$

Chương trình tham khảo bằng ngôn ngữ lập trình C++:

```

#include <bits/stdc++.h>

using namespace std;

int BIT[10000001], a[10000001];

void update(int p) { for (; p <= 1e6; p += p&-p) ++BIT[p]; }

int get(int p) {
    int res = 0;
    for (; p; p -= p&-p) res += BIT[p];
    return res;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
    freopen("XEPHANG.INP", "r", stdin);
    freopen("XEPHANG.OUT", "w", stdout);
    int n; cin >> n; long long res = 0;
    for (int i = 1; i <= n; ++i) cin >> a[i];
}

```

```

    for (int i = n; i >= 1; --i) {
        res += get(a[i]-1);
        update(a[i]);
    }
    cout << res;
    return 0;
}

```

Chương trình tham khảo bằng ngôn ngữ lập trình Python:

```

import sys

def update(BIT, p, size):
    while p <= size:
        BIT[p] += 1
        p += p & -p

def get(BIT, p):
    res = 0
    while p:
        res += BIT[p]
        p -= p & -p
    return res

def main():
    sys.stdin = open("XEPHANG.INP", "r")
    sys.stdout = open("XEPHANG.OUT", "w")
    n = int(input())
    a = list(map(int, input().split()))
    BIT, res = [0]*(10**6+1), 0
    for i in range(n-1, -1, -1):
        res += get(BIT, a[i] - 1)
        update(BIT, a[i], 10**6)
    print(res)

if __name__ == "__main__":
    main()

```

Link submit (test tự sinh): <https://codeforces.com/group/BM9qsRNlXk/contests>

Tác giả link submit: Nguyễn Minh Trí – Lớp 12A1 – THPT Thành Phố Hồng Ngự

Tester: Dương Anh Kiệt – Lớp 12A1 – THPT Thành Phố Hồng Ngự

Tác giả đáp án: Phan Trung Bạc – Lớp 12A7 – THPT Chu Văn An

Email liên hệ: trungbacphan2105@gmail.com