**Pedoman Teknis: Mengukur Performa dengan monitoring.h dan Menulis Kode C++ yang Benar-Benar Lulus ≤ 1 Detik & ≤ 32 MB**

Dokumen ini menjelaskan secara runtut apa yang **bisa** dan **tidak bisa** dijamin oleh monitoring.h, bagaimana cara **mengukur dengan benar**, serta **praktik terbaik** agar program C++ Anda benar-benar memenuhi batas waktu dan memori di judge.

**1) monitoring.h Itu Mengukur, Bukan Menyelamatkan**

monitoring.h hanyalah alat ukur. Ia **tidak**:

* Membatasi waktu eksekusi menjadi ≤ 1 detik.
* Membatasi memori menjadi ≤ 32 MB.

Yang ia lakukan:

* Mengambil **selang waktu** antara start\_monitoring() dan stop\_monitoring() (dengan jam dinding/wall-clock).
* Membaca **pemakaian memori saat itu** (snapshot) dari /proc/self/status (Linux), misalnya melalui nilai VmRSS:.

Artinya, kalau algoritma Anda lambat atau boros memori, **hasil monitoring ikut buruk**. Dan kalau peletakan start/stop Anda salah, angka yang tampil bisa **menipu** (misalnya terlihat cepat karena Anda mulai ukur setelah pekerjaan berat selesai).

**2) Hal Penting yang Tidak Dijamin oleh monitoring.h**

1. **Tidak Membatasi Waktu/Memori**  
   Header Anda **hanya membaca** waktu (wall-clock) dan VmRSS ketika stop\_monitoring() dipanggil. Ia bukan pembatas.
2. **VmRSS adalah Snapshot, Bukan Puncak**  
   VmRSS menunjukkan **pemakaian saat ini**. Sangat mungkin puncak (peak) memori Anda **melewati 32 MB** saat proses berjalan, kemudian menurun sebelum stop\_monitoring(). Laporan terlihat kecil padahal puncaknya besar. Banyak judge menilai **puncak** memori, bukan snapshot.  
   **Solusi**: selain VmRSS, baca juga VmHWM (High Water Mark) yang merepresentasikan **peak RSS**.
3. **Wall-Time vs CPU-Time**  
   monitoring.h biasanya menghitung **wall-time** (dipengaruhi sleep, blocking IO, penjadwalan OS). Sejumlah judge menilai **CPU-time** (waktu prosesor murni). Jadi kalau ada sleep atau input lambat, wall-time besar; CPU-time mungkin kecil—dan sebaliknya.

**3) Cara Mengukur yang Valid (Agar Angkanya Bermakna)**

1. **Ukur Hanya Waktu Komputasi**
   * **Baca semua input dulu.**  
     Baru panggil start\_monitoring().
   * Setelah selesai komputasi dan (biasanya) output, panggil stop\_monitoring() lalu cetak hasil ke **std::cerr** (bukan std::cout) agar **tidak mengacaukan format keluaran** ke judge.
2. **Ukur Puncak Memori**  
   Selain VmRSS, bacalah juga **VmHWM** dari /proc/self/status untuk mendekati cara judge menghitung memori puncak:
   * Cari baris berawalan VmHWM:  
     Parse nilainya seperti saat membaca VmRSS:.
3. **Gunakan std::chrono::steady\_clock**  
   steady\_clock adalah jam **monotonik** yang tidak mundur/maju karena sinkronisasi NTP atau perubahan jam sistem. Ini membuat pengukuran durasi lebih **andal** daripada high\_resolution\_clock (yang bisa merujuk ke system clock).
4. **Jangan Mengandalkan Header untuk “Menyelamatkan” Kode**  
   Desain algoritma dan implementasi rapi tetap nomor satu. Header hanya memverifikasi.

**4) Praktik Terbaik Agar Benar-Benar ≤ 1 Detik & ≤ 32 MB**

* **Pilih algoritma O(N)** dengan konstanta kecil dan sesuai batas input.  
  Hindari hal-hal yang mudah meledak seperti sort yang tidak perlu, nested loop besar, dsb.
* **Percepat IO**:
* ios::sync\_with\_stdio(false);
* cin.tie(nullptr);

Gunakan '\n' alih-alih std::endl (karena endl memaksa flush).

* **Hindari alokasi besar / berulang**:
  + Untuk kebutuhan penandaan sederhana, gunakan **array statis** yang ukurannya sesuai constraint, misalnya bool seen[1001].
  + Jika perlu wadah dinamis, vector.reserve(kapasitas) untuk menghindari realokasi berulang.
* **Minimalkan struktur berat**:  
  Gunakan array, bitset, atau unordered\_map kecil bila perlu. Hindari map kalau tidak wajib (karena map berbasis tree—lebih berat daripada hash table).
* **Hindari rekursi dalam** (risiko stack besar) dan hindari penggandaan string tanpa reserve.
* **Kompilasi rilis**:
* -O2 -std=c++17

Hindari -g dan sanitizer saat submit (overhead). Aktifkan sanitizer hanya saat debug lokal.

* **Jangan pakai sleep** atau menunggu input manual ketika Anda sedang mengukur performa.

**5) Mengenal dan Memakai bool seen[1001] dengan Benar**

**Tujuan**: seen[x] menandai apakah suatu nilai x (misalnya 1–1000) **pernah muncul** di input.  
**Manfaat**:

* Sangat cepat: akses O(1).
* Hemat memori: bool biasanya 1 byte/elemen (implementasi-defined), jadi untuk 1001 elemen kira-kira ~1 KB. Bahkan bisa lebih hemat lagi kalau pakai std::bitset<1001>.

**Cara Memakai** (contoh dari soal “Ringkasan Bilangan”):

1. **Deklarasi dan inisialisasi**:
2. static bool seen[1001] = {false}; // index 0 tidak dipakai, range 1..1000
3. **Menandai kehadiran** (saat membaca input):
4. int x;
5. // setelah cin >> x;
6. if (1 <= x && x <= 1000) {
7. seen[x] = true;
8. }
9. **Memindai untuk membentuk rentang berurutan**:
10. for (int i = 1; i <= 1000; ) {
11. if (!seen[i]) { ++i; continue; }
12. int l = i, r = i;
13. while (r + 1 <= 1000 && seen[r + 1]) ++r;
14. // sekarang [l..r] adalah satu rentang berurutan
15. // cetak sebagai "l-r" atau hanya "l" bila l==r
16. // ...
17. i = r + 1; // lompat ke setelah rentang
18. }

**Alternatif lebih hemat / aman ukuran**:

* std::bitset<1001> seen;
  + Set: seen.set(x);
  + Cek: seen.test(x) atau seen[x].
  + Kelebihan: padat (bit-packed), ukuran kompilasi tetap.
* std::array<bool, 1001> seen{};
  + Mirip array statis, gaya modern C++.

**Kapan perlu vector<bool>?**  
vector<bool> punya spesialisasi bit-packed, tapi sering “aneh” secara semantik (bukan referensi bool biasa). Untuk indeks kecil dan *fixed size*, lebih sederhana pakai bool seen[] atau bitset.

**6) Contoh monitoring.h yang Lebih Akurat (Wall-Time + VmRSS + VmHWM)**

#ifndef MONITORING\_H

#define MONITORING\_H

#include <chrono>

#include <fstream>

#include <iostream>

#include <string>

namespace monitoring {

using clock\_t = std::chrono::steady\_clock; // monotonic clock

static clock\_t::time\_point \_t0, \_t1;

static long \_vmrss\_kb = 0; // snapshot current RSS

static long \_vmhwm\_kb = 0; // high-water mark (peak RSS)

inline void start\_monitoring() {

\_t0 = clock\_t::now();

}

inline void stop\_monitoring() {

\_t1 = clock\_t::now();

std::ifstream f("/proc/self/status");

std::string line;

while (std::getline(f, line)) {

if (line.rfind("VmRSS:", 0) == 0) {

std::sscanf(line.c\_str(), "VmRSS: %ld kB", &\_vmrss\_kb);

} else if (line.rfind("VmHWM:", 0) == 0) {

std::sscanf(line.c\_str(), "VmHWM: %ld kB", &\_vmhwm\_kb);

}

}

}

inline void print\_monitoring(long limit\_ms, long limit\_mb) {

auto us = std::chrono::duration\_cast<std::chrono::microseconds>(\_t1 - \_t0).count();

long ms = static\_cast<long>(us / 1000);

long limit\_us = limit\_ms \* 1000;

std::cerr << "Execution time (wall): " << ms << " ms\n";

std::cerr << "Memory RSS (now): " << \_vmrss\_kb << " KB\n";

std::cerr << "Memory HWM (peak): " << \_vmhwm\_kb << " KB\n";

const long mem\_limit\_kb = limit\_mb \* 1024;

bool ok\_t = (us <= limit\_us);

bool ok\_m = (\_vmhwm\_kb ? \_vmhwm\_kb : \_vmrss\_kb) <= mem\_limit\_kb;

if (ok\_t && ok\_m) {

std::cerr << "✅ Constraints met: ≤ " << limit\_ms << " ms & ≤ " << limit\_mb << " MB\n";

} else {

std::cerr << "❌ Constraints NOT met\n";

}

}

}

using monitoring::start\_monitoring;

using monitoring::stop\_monitoring;

using monitoring::print\_monitoring;

#endif // MONITORING\_H

**Catatan**

* Cetak **ke std::cerr** agar tidak mengganggu jawaban (std::cout).
* Batas bisa Anda parameterkan, misalnya print\_monitoring(1000, 32) untuk 1 detik & 32 MB.
* Bekerja di **Linux** (mengandalkan /proc/self/status). Untuk platform lain, perlu pendekatan berbeda.

**7) Contoh Program Utuh (Menggunakan bool seen[1001] dan monitoring.h yang Benar)**

Contoh berikut menyelesaikan **Ringkasan Bilangan** secara efisien, dengan pengukuran waktu yang **dimulai setelah input terbaca**. Output tetap bersih (cocok untuk judge). Log monitoring ke stderr.

// file: main.cpp

#include <bits/stdc++.h>

using namespace std;

#ifdef ENABLE\_MONITORING

#include "monitoring.h"

#endif

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int N;

if (!(cin >> N)) return 0;

vector<int> arr(N);

for (int i = 0; i < N; ++i) cin >> arr[i];

#ifdef ENABLE\_MONITORING

start\_monitoring(); // mulai ukur SETELAH input selesai

#endif

// Presence array: tandai angka yang muncul (1..1000)

static bool seen[1001] = {false};

for (int x : arr) {

if (1 <= x && x <= 1000) seen[x] = true;

}

// Bentuk rangkuman berurutan

bool first = true;

for (int i = 1; i <= 1000; ) {

if (!seen[i]) { ++i; continue; }

int l = i, r = i;

while (r + 1 <= 1000 && seen[r + 1]) ++r;

if (!first) cout << ",";

first = false;

if (l == r) cout << l;

else cout << l << "-" << r;

i = r + 1;

}

cout << '\n';

#ifdef ENABLE\_MONITORING

stop\_monitoring();

// targetkan 1 detik & 32 MB (ubah sesuai kebutuhan Anda)

print\_monitoring(1000, 32);

#endif

return 0;

}

**Kompilasi & Jalankan (dengan monitoring lokal):**

g++ -O2 -std=c++17 main.cpp -DENABLE\_MONITORING -o run

# jalankan dengan input dari file

./run < input.txt 1>out.txt 2>log.txt

# - out.txt : jawaban untuk judge (stdout)

# - log.txt : metrik monitoring (stderr)

**Analisis singkat performa contoh di atas**:

* **Waktu**: O(1000) untuk scanning + O(N) untuk baca input; sangat kecil → aman < 1 detik.
* **Memori**: seen[1001] ~ 1 KB + vector<int> sebesar N (dengan N kecil–sedang biasanya aman jauh < 32 MB).

**Tips**: Jika N besar (misalnya puluhan juta), jangan simpan semua input di vector<int>. Anda bisa langsung menandai seen[x] saat membaca input (streaming), lalu tidak perlu menyimpan arr—ini akan memangkas memori drastis:

int N; cin >> N;

static bool seen[1001] = {false};

for (int i = 0; i < N; ++i) {

int x; cin >> x;

if (1 <= x && x <= 1000) seen[x] = true;

}

Dengan pendekatan ini, memori tetap ~1 KB terlepas dari N.

**8) Tambahan Opsional (Makin Akurat & Aman untuk Submit)**

* **Selalu bungkus monitoring dengan #ifdef ENABLE\_MONITORING**  
  Agar saat submit ke judge, Anda tinggal **tidak mendefinisikan** macro tersebut—output jadi steril.
* **Tampilkan dua angka memori**  
  VmRSS (snapshot saat ini) **dan** VmHWM (peak). Laporkan keduanya di log agar Anda tahu kondisi sebenarnya.
* **Jangan print ke stdout** dari header monitoring  
  Ini **wajib** agar format jawaban untuk judge tidak rusak.

**9) Checklist Singkat Sebelum Submit**

1. start\_monitoring() **setelah** semua input terbaca.
2. stop\_monitoring() **setelah** komputasi/print jawaban.
3. Log monitoring ke **std::cerr**; keluaran jawaban ke **std::cout**.
4. Gunakan steady\_clock untuk durasi.
5. Baca **VmHWM** selain VmRSS.
6. Kompilasi rilis: -O2 -std=c++17.
7. Periksa algoritma, IO, dan alokasi memori sesuai constraint.
8. **Matikan** monitoring saat submit (jangan definisikan ENABLE\_MONITORING).

**Penutup**

Kesimpulannya, monitoring.h **tidak** membuat program Anda “otomatis” ≤ 1 detik dan ≤ 32 MB—ia hanya mengukur. Agar lulus batasan, Anda harus:

* memilih algoritma yang tepat,
* menulis kode yang efisien (contohnya dengan pola bool seen[1001]/bitset untuk penandaan cepat dan hemat memori),
* serta mengukur performa dengan benar (waktu komputasi murni dan puncak memori).

Dengan disiplin tersebut, angka di log monitoring akan merefleksikan performa sebenarnya—dan peluang lolos di judge menjadi maksimal.