P2 報告

1. Title and Author

- Assignment number - **p2**
- Name - **潘煒翔**
- Student number - **112950009**
- Email - **panweikyle.sc12@nycu.edu.tw**

---

2. Description of the Problem

這次作業主要是要比較五種排序演算法在不同資料量下的效能，然後再用時間複雜度的理論，去推估在特定時間內可以處理多少資料

整體上，我把它分成五個重點來做：

1. 先理解時間複雜度，以及五種排序的特性
2. 想清楚程式要怎麼分工建構
3. 生出隨機資料（要固定 seed 才能重現）
4. 測量排序時間（牆鐘時間和 CPU 時間）
5. 從量測結果反推出，在 1 分鐘或 5 分鐘之內最多能處理幾筆資料

並且會在後續的處理中，一一解決並組織這些問題，尤其是時間複雜度的推演

---

3. Main Results

(a) Program Design

我的程式設計大概分幾塊：

- **隨機資料產生**：用 `mt19937` 搭配固定 seed，每次執行都會得到一樣的測資，這樣比較公平
- **五種排序函式**：Insertion、Selection、Quick、Merge、Heap，直接依照課本寫出來
- **時間量測**：分成 wall-clock（實際經過時間）跟 CPU time（程式真的用掉多少 CPU），兩個都量
- **Debug mode**：跑完之後再檢查有沒有真的排好，避免比時間比半天結果排序錯了
- **時間複雜度模型**：先用基準點的時間算常數，再推估其他 n 所需的時間，甚至反解「在固定時間內最大可處理 n」

- **main()**：就是統整這些功能，然後輸出表格

```cpp
int main(){

    // 固定 seed
    rng.seed(42);

    // 1) n → time (Volume → Time)
    std::vector<size_t> test_sizes = {10000, 20000, 50000, 100000};

    cout << "Sorting Performance (WALL CLOCK, ms)\n";
    cout << "n\tInsertion\tSelection\tQuick\t\tMerge\t\tHeap\n";
    cout << "----\t---------\t---------\t-----\t\t-----\t\t----\n";

    for (size_t n : test_sizes){
        auto base = make_random(n);
        double w1 = measure_time(insertion_sort, base);
        double w2 = measure_time(selection_sort, base);
        double w3 = measure_time(quick_sort,    base);
        double w4 = measure_time(merge_sort,    base);
        double w5 = measure_time(heap_sort,     base);
        cout.setf(std::ios::fixed); cout << setprecision(2);
        cout << n << "\t" << w1 << "\t\t" << w2 << "\t\t"
             << w3 << "\t\t" << w4 << "\t\t" << w5 << "\n";
    }
```

```cpp
// 2) n → time (Volume → CPU Time)
cout << "\nSorting Performance (CPU TIME, ms)\n";
cout << "n\tInsertion\tSelection\tQuick\t\tMerge\t\tHeap\n";
cout << "----\t---------\t---------\t-----\t\t-----\t\t----\n";

for (size_t n : test_sizes){
    auto base = make_random(n);
    double c1 = measure_cpu_ms(insertion_sort, base);
    double c2 = measure_cpu_ms(selection_sort, base);
    double c3 = measure_cpu_ms(quick_sort,     base);
    double c4 = measure_cpu_ms(merge_sort,     base);
    double c5 = measure_cpu_ms(heap_sort,      base);
    cout.setf(std::ios::fixed); cout << setprecision(2);
    cout << n << "\t" << c1 << "\t\t" << c2 << "\t\t"
         << c3 << "\t\t" << c4 << "\t\t" << c5 << "\n";
}

// Debug mode
{
    size_t nchk = 50000;
    auto base = make_random(nchk);
    auto d1 = base; insertion_sort(d1);
    auto d2 = base; selection_sort(d2);
    auto d3 = base; quick_sort(d3);
    auto d4 = base; merge_sort(d4);
    auto d5 = base; heap_sort(d5);
    bool ok1 = is_sorted_nondec(d1);
    bool ok2 = is_sorted_nondec(d2);
    bool ok3 = is_sorted_nondec(d3);
    bool ok4 = is_sorted_nondec(d4);
    bool ok5 = is_sorted_nondec(d5);

    if (!(ok1&&ok2&&ok3&&ok4&&ok5)){
        cerr << "[CHECK FAIL] I:" << ok1 << " S:" << ok2
             << " Q:" << ok3 << " M:" << ok4 << " H:" << ok5 << "\n";
    }
}
```

```cpp
// 3) time → n (Time Budget → Max Volume)
size_t n0 = 50000;
auto base0 = make_random(n0);
double t0_ins = measure_cpu_ms(insertion_sort, base0); // ~ n^2
double t0_sel = measure_cpu_ms(selection_sort, base0); // ~ n^2
double t0_quk = measure_cpu_ms(quick_sort,     base0);  // ~ n log n
double t0_mrg = measure_cpu_ms(merge_sort,     base0);  // ~ n log n
double t0_hap = measure_cpu_ms(heap_sort,      base0);  // ~ n log n

vector<pair<string,double>> budgets = {
    {"1 min",   60'000.0},
    {"5 min",  300'000.0},
    {"10 min", 600'000.0},
};

cout << "\nMax n within Time Budgets (based on CPU time @ n0="<< n0 <<")\n";
cout << "Budget\tInsertion\tSelection\tQuick\t\tMerge\t\tHeap\n";
cout << "------\t---------\t---------\t-----\t\t-----\t\t----\n";

for (auto &B : budgets){
    const string& label = B.first; double T = B.second;
    size_t n_ins = invert_max_n(T, AlgType::N2,    t0_ins, n0);
    size_t n_sel = invert_max_n(T, AlgType::N2,    t0_sel, n0);
    size_t n_quk = invert_max_n(T, AlgType::NLOGN, t0_quk, n0);
    size_t n_mrg = invert_max_n(T, AlgType::NLOGN, t0_mrg, n0);
    size_t n_hap = invert_max_n(T, AlgType::NLOGN, t0_hap, n0);
    cout << label << "\t" << n_ins << "\t\t" << n_sel << "\t\t"
         << n_quk << "\t\t" << n_mrg << "\t\t" << n_hap << "\n";
}

cin.get();

return 0;
}
```

(b) Data Structures

主要用到的結構：

- `vector<int>`：拿來存測試資料，每次排序前都複製一份。
- **亂數產生器**：`mt19937 rng`，搭配 `uniform_int_distribution`。
- **輔助函式**：
  - 五個排序函式

- `wall_ms` / `cpu_ms`
- `is_sorted_nondec` 檢查結果
- `predict_n2`, `predict_nlogn` 預測時間
- `invert_max_n` 算在時間限制下最大能處理幾筆

© Program Listing with Comments

完整程式碼我放在附錄，這裡只講功能：

- `make_random`：產生隨機整數陣列

```
// Random
static std::mt19937 rng;
vector<int> make_random(size_t n){

    uniform_int_distribution<int> d(-1000000, 1000000);
    vector<int> a(n);
    for (auto &x : a) x = d(rng);
    return a;

}
```

- `insertion_sort` 等五種排序：照課本實作

- `wall_ms`, `cpu_ms`：測時間用

```
// 牆鐘時間
double measure_time(void (*sort_func)(std::vector<int>&), std::vector<int> data){

    auto t0 = std::chrono::high_resolution_clock::now();
    sort_func(data);
    auto t1 = std::chrono::high_resolution_clock::now();
    return std::chrono::duration<double, std::milli>(t1 - t0).count();

}

// CPU time
double measure_cpu_ms(void (*f)(std::vector<int>&), std::vector<int> data){

    clock_t c0 = std::clock();
    f(data);
    clock_t c1 = std::clock();
    return 1000.0 * (c1 - c0) / CLOCKS_PER_SEC;

}
```

- `is_sorted_nondec`：確認排序結果正確性

```
// Debug mode
bool is_sorted_nondec(const std::vector<int>& a){

    for (size_t i = 1; i < a.size(); ++i)
        if (a[i-1] > a[i]) return false;
    return true;

}
```

- `predict_n2`, `predict_nlogn`：從理論複雜度推估時間

- `invert_max_n`：反推在固定時間能處理多少 n

```cpp
// 時間複雜度外推
double predict_n2(double n,double n0,double t0){ double k=t0/(n0*n0); return k*n*n; }
double predict_nlogn(double n,double n0,double t0){

    double k=t0/(n0*log2_safe(n0));
    return k*n*log2_safe(n);

}

// 反推在時間 T 內可處理的最大 n by gpt
enum class AlgType { N2, NLOGN };
size_t invert_max_n(double T_ms, AlgType type, double t0_ms, double n0){

    if (t0_ms <= 0) return 0;

    if (type == AlgType::N2){
        double k = t0_ms / (n0*n0);
        return (size_t)std::floor(std::sqrt(T_ms / k));
    }

    else{
        double k = t0_ms / (n0 * log2_safe(n0));
        auto time_model = [&](double n){ return k * n * log2_safe(n); };
        double lo = 1.0, hi = std::max(2.0, n0);
        while (time_model(hi) < T_ms) hi *= 2.0;
        for (int it=0; it<60; ++it){
            double mid = 0.5*(lo+hi);
            if (time_model(mid) <= T_ms) lo = mid; else hi = mid;
        }
        return (size_t)std::floor(lo);
    }

}
```

(d) Program Outputs

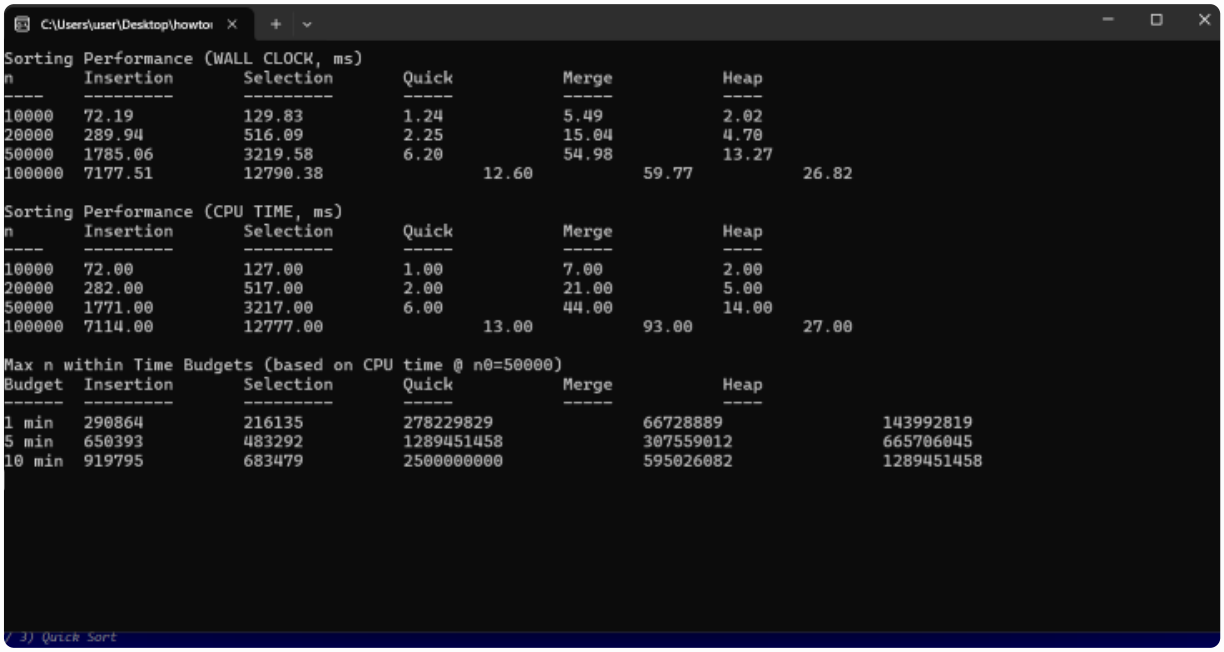
程式會印兩種表格：

1. **不同資料量下的時間**
   （Volume → Time）

n	Insertion	Selection	Quick	Merge	Heap
10,000	…	…	…	…	…
20,000	…	…	…	…	…
50,000	…	…	…	…	…
100,000	…	…	…	…	…

2. **固定時間內可處理的最大資料量**
   （Time → Volume）

Budget	Insertion	Selection	Quick	Merge	Heap
1 min	…	…	…	…	…
5 min	…	…	…	…	…
10 min	…	…	…	…	…



---

4. Performance Evaluation

- 小資料量時（例如 1 萬筆以下），Insertion 跟 Selection 還算能打，甚至有時候不會比 Quick 差太多。
- 中大型資料量時（幾萬筆以上），就能明顯看到 $O(n^2)$ 的演算法跑不動，只有 $O(n \log n)$ 的演算法能繼續撐住。
- Heap Sort 雖然理論上跟 Quick/Merge 一樣是 $O(n \log n)$，但常數比較大，所以結果通常稍微慢一點。
- 用基準點去推估「固定時間可處理的資料量」這部分，數字跟理論曲線的趨勢差不多，證明這個方法蠻合理的。

---

5. Conclusions

這次作業讓我更清楚了幾件事：

- 理論複雜度雖然重要，但實際測試才會看出常數、快取、系統環境的影響
- 固定 seed 的隨機輸入讓比較更公平
- 同時量 wall-clock 跟 CPU time，能看出「使用者體感時間」和「演算法真實效能」的差異，這也算是額外的功能
- Debug mode 很實用，能保證結果正確，不然排序錯了再快也沒意義。
- 最有趣的是時間反推的部分：把「1 分鐘內能排多少筆」算出來，讓我更直觀地理解時間複雜度跟程式效能的連結。

在這次作業裡，我花最多心思的其實是「時間複雜度」。一開始只是背公式，後來真的要用它來做推估的時候，發現要算的東西很多，還得自己拿紙筆演練，把 $O(n^2)$、$O(n \log n)$ 的推導寫清楚，再試著用程式去實現。過程中我也參考了一些網路上的資料，最後把「在固定時間內最多能處理多少資料」這件事寫出來，感覺很有成就感。

在 random 的部分，我以前只覺得「亂數就是亂數」，不會特別去管，但這次有試過固定 seed 和不固定 seed，才知道原來裡面有這麼多學問。用同一個 seed 真的可以讓結果更公平，也能重現同樣的輸出，這點我以前完全沒想過。現在也理解為什麼老師要特別要求大家用同一個 random 產生方式。

Debug mode 對我來說也蠻新鮮的，因為平常寫完程式只要跑出結果就好，這次卻特別設計了一個模式去檢查排序後的正確性。做了之後覺得超酷，而且也意識到這和時間複雜度有關：如果排序錯誤，再快的時間也沒有意義，所以正確性檢查變得很重要。

整體來說，這份作業讓我不只是練程式，還真的去思考「為什麼要這樣設計」、「這樣做有什麼理論基礎」，讓我學到蠻多以前沒注意過的細節。

總之，這次 P2 作業不只是寫排序而已，更像是一個「實驗＋分析」的過程，讓我真的把理論和實測連在一起了。

---

6. 附錄

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <chrono>
#include <algorithm>
#include <sstream>
#include <cmath>
#include <ctime>
#include <iomanip>
#include <utility>
using namespace std;

// Random
static std::mt19937 rng;
vector<int> make_random(size_t n){

    uniform_int_distribution<int> d(-1000000, 1000000);
    vector<int> a(n);
    for (auto &x : a) x = d(rng);
    return a;

}

// 1) Insertion Sort
void insertion_sort(vector<int>& a){

    for (size_t i = 1; i < a.size(); ++i){

        int y = a[i];
        int j = static_cast<int>(i) - 1;

        while (j >= 0 && a[j] > y){
            a[j + 1] = a[j]; --j;
        }

        a[j + 1] = y;

    }
}

// 2) Selection Sort (避免 size_t underflow)
void selection_sort(vector<int>& a){

    for (size_t i = 0; i + 1 < a.size(); ++i){

        size_t min_idx = i;

        for (size_t j = i + 1; j < a.size(); ++j)
            if (a[j] < a[min_idx]) min_idx = j;
```

```
        if (min_idx != i) swap(a[i], a[min_idx]);

    }
}

// 3) Quick Sort
void quick_sort_helper(vector<int>& a, int left, int right){

    if (left >= right) return;
    int pivot = a[left + (right - left) / 2];
    int i = left, j = right;

    while (i <= j){
        while (a[i] < pivot) ++i;
        while (a[j] > pivot) --j;
        if (i <= j) swap(a[i++], a[j--]);
    }

    if (left < j) quick_sort_helper(a, left, j);
    if (i < right) quick_sort_helper(a, i, right);

}
void quick_sort(vector<int>& a){

    if (!a.empty()) quick_sort_helper(a, 0, (int)a.size() - 1);

}

// 4) Merge Sort
void merge(vector<int>& a, int left, int mid, int right){

    vector<int> temp(right - left + 1);
    int i = left, j = mid + 1, k = 0;

    while (i <= mid && j <= right){
        if (a[i] <= a[j]) temp[k++] = a[i++];
        else              temp[k++] = a[j++];
    }
    while (i <= mid)   temp[k++] = a[i++];
    while (j <= right) temp[k++] = a[j++];

    for (int idx = 0; idx < k; ++idx) a[left + idx] = temp[idx];

}
void merge_sort_t(vector<int>& a, int left, int right){
```

```
    if (left >= right) return;
    int mid = left + (right - left) / 2;
    merge_sort_t(a, left, mid);
    merge_sort_t(a, mid + 1, right);
    merge(a, left, mid, right);

}

void merge_sort(vector<int>& a){

    if (!a.empty()) merge_sort_t(a, 0, (int)a.size() - 1);

}

// 5) Heap Sort
void heapify(vector<int>& a, int n, int i){

    int largest = i, left = 2 * i + 1, right = 2 * i + 2;

    if (left  < n && a[left]  > a[largest]) largest = left;
    if (right < n && a[right] > a[largest]) largest = right;
    if (largest != i){ swap(a[i], a[largest]); heapify(a, n, largest); }

}
void heap_sort(vector<int>& a){

    int n = (int)a.size();

    for (int i = n / 2 - 1; i >= 0; --i) heapify(a, n, i);
    for (int i = n - 1; i > 0; --i){ swap(a[0], a[i]); heapify(a, i, 0); }

}

// 安全 log2 by gpt
inline double log2_safe(double x){ return (x <= 1.0) ? 1.0 : std::log2(x); }

// 鐘錶時間
double measure_time(void (*sort_func)(std::vector<int>&), std::vector<int> data){

    auto t0 = std::chrono::high_resolution_clock::now();
    sort_func(data);
    auto t1 = std::chrono::high_resolution_clock::now();
    return std::chrono::duration<double, std::milli>(t1 - t0).count();

}
```

```cpp
// CPU time
double measure_cpu_ms(void (*f)(std::vector<int>&), std::vector<int> data){

    clock_t c0 = std::clock();
    f(data);
    clock_t c1 = std::clock();
    return 1000.0 * (c1 - c0) / CLOCKS_PER_SEC;

}

// Debug mode
bool is_sorted_nondec(const std::vector<int>& a){

    for (size_t i = 1; i < a.size(); ++i)
        if (a[i-1] > a[i]) return false;
    return true;

}

// 時間複雜度外推
double predict_n2(double n,double n0,double t0){ double k=t0/(n0*n0); return k*n*n; }
double predict_nlogn(double n,double n0,double t0){

    double k=t0/(n0*log2_safe(n0));
    return k*n*log2_safe(n);

}

// 反推在時間 T 內可處理的最大 n by gpt
enum class AlgType { N2, NLOGN };
size_t invert_max_n(double T_ms, AlgType type, double t0_ms, double n0){

    if (t0_ms <= 0) return 0;

    if (type == AlgType::N2){
        double k = t0_ms / (n0*n0);
        return (size_t)std::floor(std::sqrt(T_ms / k));
    }

    else{
        double k = t0_ms / (n0 * log2_safe(n0));
        auto time_model = [&](double n){ return k * n * log2_safe(n); };
        double lo = 1.0, hi = std::max(2.0, n0);
        while (time_model(hi) < T_ms) hi *= 2.0;
        for (int it=0; it<60; ++it){
            double mid = 0.5*(lo+hi);
            if (time_model(mid) <= T_ms) lo = mid; else hi = mid;
        }
        return (size_t)std::floor(lo);
    }
```

```cpp
int main(){

    // 固定 seed
    rng.seed(42);

    // 1) n → time (Volume → Time)
    std::vector<size_t> test_sizes = {10000, 20000, 50000, 100000};

    cout << "Sorting Performance (WALL CLOCK, ms)\n";
    cout << "n\tInsertion\tSelection\tQuick\t\tMerge\t\tHeap\n";
    cout << "----\t---------\t---------\t-----\t\t-----\t\t----\n";

    for (size_t n : test_sizes){
        auto base = make_random(n);
        double w1 = measure_time(insertion_sort, base);
        double w2 = measure_time(selection_sort, base);
        double w3 = measure_time(quick_sort,    base);
        double w4 = measure_time(merge_sort,    base);
        double w5 = measure_time(heap_sort,     base);
        cout.setf(std::ios::fixed); cout << setprecision(2);
        cout << n << "\t" << w1 << "\t\t" << w2 << "\t\t"
             << w3 << "\t\t" << w4 << "\t\t" << w5 << "\n";
    }
```

```cpp
    // 2) n → time (Volume → CPU Time)
    cout << "\nSorting Performance (CPU TIME, ms)\n";
    cout << "n\tInsertion\tSelection\tQuick\t\tMerge\t\tHeap\n";
    cout << "----\t---------\t---------\t-----\t\t-----\t\t----\n";

    for (size_t n : test_sizes){
        auto base = make_random(n);
        double c1 = measure_cpu_ms(insertion_sort, base);
        double c2 = measure_cpu_ms(selection_sort, base);
        double c3 = measure_cpu_ms(quick_sort,    base);
        double c4 = measure_cpu_ms(merge_sort,    base);
        double c5 = measure_cpu_ms(heap_sort,     base);
        cout.setf(std::ios::fixed); cout << setprecision(2);
        cout << n << "\t" << c1 << "\t\t" << c2 << "\t\t"
             << c3 << "\t\t" << c4 << "\t\t" << c5 << "\n";
    }

    // Debug mode
    {
        size_t nchk = 50000;
        auto base = make_random(nchk);
        auto d1 = base; insertion_sort(d1);
        auto d2 = base; selection_sort(d2);
        auto d3 = base; quick_sort(d3);
        auto d4 = base; merge_sort(d4);
        auto d5 = base; heap_sort(d5);
        bool ok1 = is_sorted_nondec(d1);
        bool ok2 = is_sorted_nondec(d2);
        bool ok3 = is_sorted_nondec(d3);
        bool ok4 = is_sorted_nondec(d4);
        bool ok5 = is_sorted_nondec(d5);

        if (!(ok1&&ok2&&ok3&&ok4&&ok5)){
            cerr << "[CHECK FAIL] I:" << ok1 << " S:" << ok2
                 << " Q:" << ok3 << " M:" << ok4 << " H:" << ok5 << "\n";
        }
    }
```

```cpp
    // 3) time → n (Time Budget → Max Volume)
    size_t n0 = 50000;
    auto base0 = make_random(n0);
    double t0_ins = measure_cpu_ms(insertion_sort, base0); // ~ n^2
    double t0_sel = measure_cpu_ms(selection_sort, base0); // ~ n^2
    double t0_quk = measure_cpu_ms(quick_sort,     base0);  // ~ n log n
    double t0_mrg = measure_cpu_ms(merge_sort,     base0);  // ~ n log n
    double t0_hap = measure_cpu_ms(heap_sort,      base0);  // ~ n log n

    vector<pair<string,double>> budgets = {
        {"1 min",   60'000.0},
        {"5 min",  300'000.0},
        {"10 min", 600'000.0},
    };

    cout << "\nMax n within Time Budgets (based on CPU time @ n0="<< n0 <<")\n";
    cout << "Budget\tInsertion\tSelection\tQuick\t\tMerge\t\tHeap\n";
    cout << "------\t---------\t---------\t-----\t\t-----\t\t----\n";

    for (auto &B : budgets){
        const string& label = B.first; double T = B.second;
        size_t n_ins = invert_max_n(T, AlgType::N2,    t0_ins, n0);
        size_t n_sel = invert_max_n(T, AlgType::N2,    t0_sel, n0);
        size_t n_quk = invert_max_n(T, AlgType::NLOGN, t0_quk, n0);
        size_t n_mrg = invert_max_n(T, AlgType::NLOGN, t0_mrg, n0);
        size_t n_hap = invert_max_n(T, AlgType::NLOGN, t0_hap, n0);
        cout << label << "\t" << n_ins << "\t\t" << n_sel << "\t\t"
             << n_quk << "\t\t" << n_mrg << "\t\t" << n_hap << "\n";
    }

    cin.get();

    return 0;

}
```