

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Báo cáo bài tập lớn

Chủ đề: Xây dựng chương trình console đơn giản kiểm thử thời gian thực thi của một số thuật toán sắp xếp.

Học phần: Project I - IT3150

GVHD: TS. Bùi Thị Mai Anh

Mã lớp: 744278

Sinh viên thực hiện: Đặng Tiến Cường

MSSV: 20220020

Lớp: CTTN-KHMT K67

Hà Nội, Ngày 27 tháng 12 năm 2024

Mục lục

1. Yêu cầu của bài tập	3
2. Tổng quan về thuật toán sắp xếp	4
2.1. Bubble Sort.....	4
2.1.1. Giới thiệu	4
2.1.2. Đặc điểm.....	4
2.1.3. Triển khai thuật toán	4
2.2 Insertion Sort	7
2.2.1. Giới thiệu	7
2.2.2. Đặc điểm.....	7
2.2.3. Triển khai thuật toán	7
2.2.4. Ứng dụng.....	8
2.3. Quick Sort.....	9
2.3.1. Giới thiệu	9
2.3.2. Đặc điểm.....	10
2.3.3. Triển khai thuật toán	10
2.3.4. Ứng dụng.....	15
Tóm lại	16
3. Xây dựng chương trình.....	17
Biểu đồ Activity Diagram	20
4. Demo chương trình	22
Quy trình đo lường thời gian thực thi của một trong ba thuật toán sắp xếp	22
Quy trình so sánh thời gian thực thi của ba thuật toán	26
5. Kết quả thực nghiệm	29
5.1. Với input n = 10,000	29
5.2. Với input n = 100,000.....	29
5.3. Với input n = 1,000,000	30

1. Yêu cầu của bài tập

Cài đặt chương trình cho phép kiểm chứng các thuật toán sắp xếp sau đây:

- Bubble Sort
- Quick Sort
- Insertion Sort

Xây dựng menu chương trình từ console cho phép người dùng:

1. Import 1 file .txt chứa một dãy các số cần phải sắp xếp – Yêu cầu số lượng phần tử cần thiết là >10.000
2. Lựa chọn 1 trong 3 thuật toán để sắp xếp
3. Người dùng sẽ được hỏi sẽ lưu lại kết quả đã sắp xếp dưới dạng file tên do người dùng cung cấp và xem thông tin kết quả chạy thuật toán: thời gian thực thi
4. So sánh thời gian thực thi của 3 thuật toán, input cần thay đổi $10.000, 100.000$ và $1.000.000$ phần tử để so sánh được hiệu năng của từng thuật toán

Sinh viên sẽ đặt system time starting – system time ending khi thuật toán đã kết thúc để tính ra thời gian chạy.

2. Tổng quan về thuật toán sắp xếp

2.1. Bubble Sort

2.1.1. Giới thiệu

Sắp xếp nổi bọt (Bubble Sort) là thuật toán sắp xếp đơn giản. Thuật toán hoạt động bằng duyệt qua toàn bộ dãy các phần tử chưa sắp xếp, bắt đầu từ phần tử đầu tiên thực hiện lặp đi lặp lại 02 thao tác chính:

1. So sánh phần tử hiện tại với phần tử ngay sau nó.
2. Thực hiện đổi chỗ nếu cần (để đưa phần tử lớn hơn về cuối dãy hoặc đầu dãy).

Quá trình này lặp đi lặp lại cho đến khi không còn cặp phần tử nào cần đổi chỗ, tức là toàn bộ dãy phần tử đã được sắp xếp theo thứ tự mong muốn.

2.1.2. Đặc điểm

Bubble Sort là thuật toán sắp xếp tại chỗ (in-place), ổn định (stable), không cần bộ nhớ phụ.

Hiệu quả thấp: Bubble Sort có độ phức tạp thời gian là $O(n^2)$ trong trường hợp xấu nhất và trung bình, nên hoạt động kém trong thực tế, đặc biệt là với dữ liệu lớn.

Ứng dụng: Bubble Sort thường chỉ được sử dụng như công cụ học thuật để giúp người học hiểu cơ bản về thuật toán sắp xếp.

2.1.3. Triển khai thuật toán

Mã giả Bubble Sort - phiên bản cơ bản

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n-1 inclusive do
            {if this pair is out of order}
            if A[i-1] > A[i] then
                {swap them and remember something changed}
                swap(A[i-1], A[i])
                swapped := true
            end if
        end for
        until not swapped
end procedure
```

Nhận xét:

Ta thấy vòng lặp **for** luôn kiểm tra từ đầu đến cuối danh sách, bất kể các phần tử lớn nhất đã được sắp xếp về đúng vị trí ở cuối danh sách. Điều này dẫn đến việc lặp lại không cần thiết ở các phần tử đã được sắp xếp trong các vòng lặp sau.

Best case: Độ phức tạp tính toán là $O(n)$ nếu mảng đã được sắp xếp (chỉ cần một lần lặp để xác định không có phần tử nào cần hoán đổi).

Worst Case: Độ phức tạp tính toán là $O(n^2)$, do vẫn kiểm tra tất cả các phần tử trong mỗi lượt.

Do đó ở phần sau, chúng ta sẽ trình bày ý tưởng để thực hiện tối ưu hóa thuật toán.

Mã giả Bubble Sort – phiên bản tối ưu hoá 1

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n - 1 inclusive do
            if A[i - 1] > A[i] then
                swap(A[i - 1], A[i])
                swapped := true
            end if
        end for
        n := n - 1
    until not swapped
end procedure
```

Ý tưởng:

Thay vì vẫn duyệt qua toàn bộ phần tử ở mỗi lần lặp, ta thực hiện giảm phạm vi kiểm tra sau mỗi vòng lặp, biến n giảm sau mỗi lượt ($n := n - 1$), nghĩa là trong vòng lặp tiếp theo, các phần tử cuối cùng (đã được sắp xếp) sẽ bị loại bỏ khỏi phạm vi kiểm tra.

Nhận xét:

Best case: $O(n)$ (nếu mảng đã sắp xếp, tương tự cách thuật toán cơ bản)

Worst Case: $O(n^2)$ nhưng ít phép so sánh hơn thuật toán cơ bản.

Average Case: Tốt hơn cách 1 do giảm số lượng phép toán cần kiểm tra.

Mã giả Bubble Sort – phiên bản tối ưu hoá 2

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        newn := 0
        for i := 1 to n - 1 inclusive do
            if A[i - 1] > A[i] then
                swap(A[i - 1], A[i])
                newn := i
            end if
        end for
        n := newn
    until n ≤ 1
```

```
end procedure
```

Ý tưởng:

Thay vì giảm phạm vi kiểm tra một cách cố định (giảm n một lượt lặp), thuật toán này chỉ kiểm tra đến phần tử cuối cùng có sự hoán đổi xảy ra.

Sau mỗi lượt lặp, nếu không có hoán đổi ở một phần tử nào đó, các phần tử phía sau nó chắc chắn đã được sắp xếp.

Biến `newn` giữ vị trí của lần hoán đổi cuối cùng trong mỗi vòng lặp. Trong lượt lặp tiếp theo, phạm vi kiểm tra sẽ được giới hạn đến `newn`, vì mọi phần tử sau vị trí đó đều đã được đảm bảo sắp xếp đúng vị trí.

Nhận xét:

- Số lần so sánh: Giảm đáng kể, đặc biệt trong các trường hợp mảng gần sắp xếp.
- Số lần hoán đổi: Không thay đổi, vì số lần hoán đổi phụ thuộc vào thứ tự ban đầu của mảng.
- Best Case: $O(n)$ (nếu mảng đã sắp xếp, thuật toán vẫn dừng sớm sau một lần lặp duy nhất)
- Worst Case: $O(n^2)$ (vì vẫn giữ bản chất của Bubble Sort)

Thuật toán này là một cải tiến đáng kể của thuật toán Bubble Sort, nhưng không phải tối ưu hoá tuyệt đối. Nó giảm đáng kể số phép so sánh cần thiết trong worst-case và đặc biệt hiệu quả hơn khi dãy số đã gần như được sắp xếp.

Mã C++ cho phiên bản tối ưu 2

```
void bubbleSort(std::vector<int>& arr, int lo, int hi) {
    int n = hi - lo + 1;
    while (n > 1) {
        int newn = lo;
        for (int i = lo; i < hi; ++i) {
            if (arr[i] > arr[i + 1]) {
                std::swap(arr[i], arr[i + 1]);
                newn = i + 1;
            }
        }
        n = newn;
    }
}
```

2.2 Insertion Sort

2.2.1. Giới thiệu

Sắp xếp chèn (Insertion Sort) là một thuật toán sắp xếp đơn giản nhưng hiệu quả nhất trong các thuật toán sắp xếp với mảng nhỏ. Thuật toán hoạt động bằng cách duyệt qua từng phần tử của mảng đầu vào, loại bỏ phần tử đó, sau đó chèn nó vào vị trí phù hợp trong phần mảng đã được sắp xếp theo thứ tự tăng dần hoặc giảm dần, tùy thuộc vào cách định nghĩa so sánh.

Các bước hoạt động:

1. Bắt đầu từ phần tử thứ hai (vị trí 1), vì phần tử đầu tiên mặc định được coi là đã được sắp xếp.
2. Chọn phần tử hiện tại và so sánh nó với các phần tử phía trước trong mảng đã được sắp xếp.
3. Nếu phần tử hiện tại nhỏ hơn thì dịch chuyển các phần tử lớn hơn sang bên phải để tạo khoảng trống.
4. Chèn phần tử hiện tại vào đúng vị trí trong mảng đã sắp xếp.
5. Lặp lại quá trình cho đến khi toàn bộ mảng được duyệt.

2.2.2. Đặc điểm

Ưu điểm:

1. Thuật toán có cách triển khai dễ hiểu, đơn giản.
2. Hiệu quả trong trường hợp mảng gần như đã được sắp xếp (best-case $O(n)$)
3. Là thuật toán sắp xếp tại chỗ (in-place), ổn định (stable), không cần bộ nhớ phụ.

Nhược điểm:

1. Hiệu suất kém khi áp dụng cho mảng có kích thước lớn do độ phức tạp trong trường hợp xấu nhất là $O(n^2)$
2. Số lần dịch chuyển phần tử có thể lớn, gây ảnh hưởng đến hiệu năng.

2.2.3. Triển khai thuật toán

Mã giả Insertion phiên bản cơ bản

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

Nhận xét:

Ở phiên bản cơ bản: Thuật toán thực hiện việc hoán đổi vị trí (swap) giữa các phần tử trong mảng để đưa phần tử cần chèn A[i] về đúng vị trí của nó. Tuy nhiên mỗi lần hoán đổi là 3 lần gán giá trị (vì phải cần một biến lưu trữ tạm thời). Điều này có thể tăng độ phức tạp tính toán nếu mảng đầu vào có kích thước lớn. Do đó chúng ta có phiên bản tối ưu hơn của thuật toán.

Mã giả Insertion phiên bản tối ưu

```
i ← 1
while i < length(A)
    x ← A[i]
    j ← i
    while j > 0 and A[j-1] > x
        A[j] ← A[j-1]
        j ← j - 1
    end while
    A[j] ← x
    i ← i + 1
end while
```

Nhận xét:

Ở phiên bản tối ưu: Thay vì liên tục hoán đổi, chỉ cần di chuyển (shift) các phần tử lớn hơn sang phải, sau đó chèn phần tử cần chèn (x) vào vị trí thích hợp. Điều này giảm số lần gán giá trị trong vòng lặp, giúp thuật toán nhanh hơn.

Mã C++ cho phiên bản tối ưu

```
void insertionSort(std::vector<int>& arr, int lo, int hi) {
    for (int i = lo + 1; i <= hi; ++i) {
        int key = arr[i];
        int j = i - 1;
        while (j >= lo && arr[j] > key) {
            arr[j + 1] = arr[j];
            --j;
        }
        arr[j + 1] = key;
    }
}
```

2.2.4. Ứng dụng

- Thích hợp cho các mảng nhỏ hoặc dữ liệu gần như đã sắp xếp.
- Thường được sử dụng trong các thuật toán sắp xếp phức tạp hơn như Quick Sort để sắp xếp các mảng con nhỏ.

2.3. Quick Sort

2.3.1. Giới thiệu

Sắp xếp nhanh (Quick Sort) là một thuật toán sắp xếp hiệu quả, dựa trên nguyên lý chia để trị. Thuật toán này được phát triển bởi nhà khoa học máy tính người Anh Tony Hoare vào năm 1959 và được công bố vào năm 1961. Cho đến nay nó vẫn là thuật toán sắp xếp được sử dụng phổ biến. Tổng thể, QuickSort nhanh hơn Merge Sort và HeapSort đối với dữ liệu ngẫu nhiên, đặc biệt khi làm việc với các tập dữ liệu lớn.

Quick Sort dựa trên một quy trình phân vùng (partitioning routine). Chi tiết về việc phân vùng này có thể thay đổi đôi chút, vì Quick Sort là một tập hợp các thuật toán có liên quan đến nhau.

Khi áp dụng cho một vùng dữ liệu có ít nhất 2 phần tử, quy trình phân vùng sẽ chia vùng đó thành 2 vùng con không rỗng, sao cho không có phần tử nào trong vùng con thứ nhất lớn hơn bất kỳ phần tử nào trong vùng con thứ hai. Sau khi thực hiện phân vùng, Quick Sort sẽ sắp xếp đệ quy các vùng con này, có thể loại trừ một phần tử tại điểm phân chia đã được xác định là ở đúng vị trí cuối cùng.

Các bước hoạt động

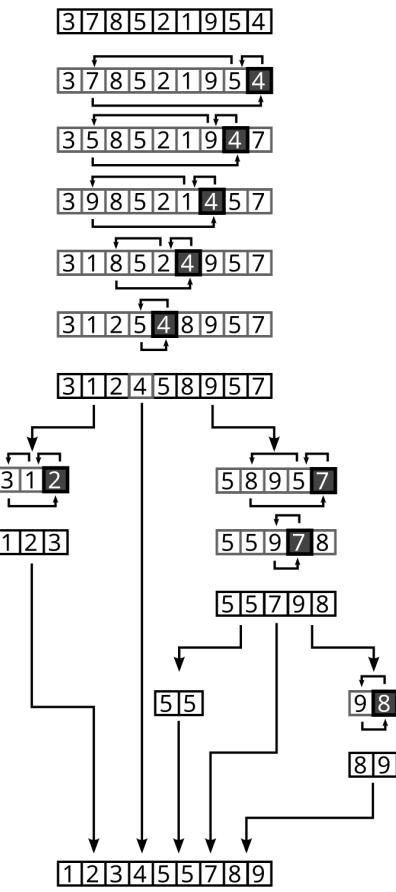
1. Trường hợp cơ sở: Nếu vùng dữ liệu có ít hơn 2 phần tử, thoát khỏi hàm ngay lập tức vì không cần thực hiện gì thêm. Trong một số trường hợp với vùng dữ liệu rất nhỏ, có thể áp dụng một phương pháp sắp xếp đặc biệt (như Insertion Sort) và bỏ qua các bước còn lại.

2. Chọn trục (Pivot): Nếu vùng dữ liệu có từ 02 phần tử trở lên, chọn một giá trị làm trục (pivot) (cách chọn phụ thuộc vào quy trình phân vùng, có thể liên quan đến tính ngẫu nhiên)

3. Phân vùng:

Sắp xếp lại các phần tử trong vùng: Xác định một điểm chia sao cho tất cả các phần tử có giá trị nhỏ hơn trục sẽ nằm trước điểm chia, và các phần tử có giá trị lớn hơn trục sẽ nằm sau nó. Các phần tử có giá trị bằng trục có thể nằm ở bất kỳ bên nào.

4. Sắp xếp đệ quy:



Áp dụng đệ quy QuickSort cho vùng con nằm trước điểm chia và vùng con nằm sau điểm chia.

2.3.2. Đặc điểm

Độ phức tạp tính toán:

Average case: $O(n \log n)$, do mỗi bước chia đôi mảng và sắp xếp hai vùng con.

Worst case: $O(n^2)$, xảy ra khi dữ liệu đã được sắp xếp và trực tiếp được chọn không cân bằng (ví dụ: luôn là phần tử nhỏ nhất hoặc lớn nhất của mảng)

Ưu điểm:

Tại chỗ (in-place): Có thể thực hiện tại chỗ (in-place), chỉ cần thêm một lượng bộ nhớ bổ sung, thường là $O(\log n)$ cho ngăn xếp đệ quy.

Hiệu suất cao: Trung bình, Quick Sort nhanh hơn Merge Sort và Heap Sort khi xử lý dữ liệu ngẫu nhiên.

Đa năng: Dễ dàng điều chỉnh để xử lý các loại dữ liệu khác nhau (ví dụ: Thêm Median-of-Three để chọn trực tiếp tốt hơn)

Nhược điểm:

Không ổn định (not stable): Không giữ được thứ tự của các phần tử bằng nhau, đây là hạn chế trong một số ứng dụng.

Yêu cầu đệ quy: Gây ra rủi ro tràn ngăn xếp (stack overflow) khi kích thước mảng dữ liệu phân hoạch không đều.

2.3.3. Triển khai thuật toán

Thuật toán Quick Sort thường được triển khai dưới hai mô hình phổ biến: Lomuto Partition Scheme và Hoare Partition Scheme. Ở phần dưới, chúng ta sẽ trình bày về ý tưởng, ưu điểm và nhược điểm của 02 mô hình này.

Lomuto Partition Scheme

Ý tưởng:

1. Chọn một phần tử làm pivot (thường là phần tử cuối cùng)
2. Phân vùng mảng bằng cách đưa cách phần tử nhỏ hơn pivot về phía bên trái và các phần tử lớn hơn về phía bên phải.
3. Thực hiện hoán đổi dựa trên một chỉ số duy nhất để xác định vị trí kết thúc của các phần tử nhỏ hơn pivot.

Mã giả (Pseudocode)

```
// Sorts (a portion of) an array, divides it into partitions,
// then sorts those
algorithm quicksort(A, lo, hi) is
    // Ensure indices are in correct order
    if lo >= hi || lo < 0 then
        return
```

```

// Partition array and get the pivot index
p := partition(A, lo, hi)

// Sort the two partitions
quicksort(A, lo, p - 1) // Left side of pivot
quicksort(A, p + 1, hi) // Right side of pivot

// Divides array into two partitions
algorithm partition(A, lo, hi) is
    pivot := A[hi] // Choose the last element as the pivot

    // Temporary pivot index
    i := lo

    for j := lo to hi - 1 do
        // If the current element is less than or equal to the pivot
        if A[j] <= pivot then
            // Swap the current element with the element at the temporary pivot index
            swap A[i] with A[j]
            // Move the temporary pivot index forward
            i := i + 1

    // Swap the pivot with the last element
    swap A[i] with A[hi]
    return i // the pivot index

```

Ưu điểm:

1. Dễ triển khai, mã nguồn đơn giản hơn.
2. Phù hợp với các mảng nhỏ hoặc khi pivot được chọn tối ưu.

Nhược điểm:

1. Hiệu suất kém với dữ liệu chứa nhiều phần tử trùng lặp hoặc khi pivot được chọn không tốt.
2. Thời gian xử lý trường hợp tệ nhất là $O(n^2)$.

Hoare Partition Scheme

Ý tưởng:

1. Chọn một phần tử làm pivot (thường là phần tử đầu tiên hoặc giữa mảng).
2. Sử dụng hai chỉ số, một di chuyển từ trái sang phải và một từ phải sang trái.
3. Hoán đổi các phần tử ở hai phía nếu cần để đảm bảo các phần tử nhỏ hơn pivot nằm ở bên trái và lớn hơn pivot nằm ở bên phải.
4. Kết thúc khi hai chỉ số gặp nhau.

Mã giả (Pseudocode)

```

// Sorts (a portion of) an array, divides it into partitions,
then sorts those
algorithm quicksort(A, lo, hi) is
  if lo >= 0 && hi >= 0 && lo < hi then
    p := partition(A, lo, hi)
    quicksort(A, lo, p) // Note: the pivot is now included
    quicksort(A, p + 1, hi)

// Divides array into two partitions
algorithm partition(A, lo, hi) is
  // Pivot value
  pivot := A[lo] // Choose the first element as the pivot

  // Left index
  i := lo - 1

  // Right index
  j := hi + 1

  loop forever
    // Move the left index to the right at least once and while
    // the element at
    // the left index is less than the pivot
    do i := i + 1 while A[i] < pivot

    // Move the right index to the left at least once and while
    // the element at
    // the right index is greater than the pivot
    do j := j - 1 while A[j] > pivot

    // If the indices crossed, return
    if i >= j then return j

    // Swap the elements at the left and right indices
    swap A[i] with A[j]

```

Ưu điểm:

1. Hiệu suất tốt hơn với các mảng có nhiều phần tử trùng lặp.
2. Ít hoán đổi phần tử hơn so với Lomuto, do đó tận dụng tốt hơn bộ nhớ đệm.

Nhược điểm:

Mã nguồn phức tạp hơn so với Lomuto

So sánh Hoare's scheme và Lomuto's scheme

1. Hiệu suất của Hoare tốt hơn Lomuto

Hoare's partition scheme thực hiện số lần hoán đổi trung bình ít hơn gấp ba lần so với Lomuto.

Khi tất cả giá trị đều bằng nhau, Hoare tạo ra các phân vùng cân bằng, trong khi Lomuto không làm được điều này.

2. Độ phức tạp thời gian

Giống như Lomuto, Hoare cũng có thể bị giảm hiệu suất xuống $O(n^2)$ khi dữ liệu đã được sắp xếp và pivot được chọn là phần tử đầu tiên hoặc cuối cùng.

Tuy nhiên, nếu chọn pivot là phần tử giữa, dữ liệu được sắp xếp sẽ tạo ra phân vùng cân bằng với hầu như không có hoán đổi, dẫn đến trường hợp tốt nhất $O(n \log n)$.

3. Tính ổn định và vị trí cuối của pivot

Hoare không tạo ra một sắp xếp ổn định (stable sort), vì các phần tử bằng pivot có thể không giữ nguyên vị trí ban đầu. Vị trí cuối cùng của pivot không nhất thiết nằm ở chỉ số được trả về sau bước phân vùng.

Hoare chia hai phân vùng: Từ lo đến p (các phần tử \leq pivot) và từ $p + 1$ đến hi (các phần tử \geq pivot). Điều này khác với Lomuto, nơi hai phân vùng là từ lo đến $p - 1$ và $p + 1$ đến hi .

Lựa chọn Pivot

Như đã trình bày, ở trường hợp tồi nhất nếu chọn phần tử đầu hoặc cuối làm pivot sẽ dẫn đến hiệu suất tệ khi áp dụng trên mảng đã sắp xếp. Ta có thể giải quyết vấn đề này bằng cách chọn ngẫu nhiên một chỉ số làm pivot, chọn chỉ số giữa của phân vùng, hoặc (đặc biệt với các phân vùng dài hơn) chọn giá trị trung vị của ba phần tử: phần tử đầu tiên, phần tử giữa và phần tử cuối của phân vùng làm pivot.

Quy tắc **Median-Of-Three** (Trung vị của ba phần tử) này khắc phục được trường hợp mảng đã sắp xếp (hoặc sắp xếp ngược) và cung cấp một ước lượng tốt hơn cho pivot tối ưu (trung vị thực sự) so với việc chọn bất kỳ phần tử nào, khi không có thông tin nào về thứ tự dữ liệu đầu vào.

Mã giả (Pseudocode)

```
mid := [lo + (hi - lo) / 2]
if A[mid] < A[lo]
    swap A[lo] with A[mid]
if A[hi] < A[lo]
    swap A[lo] with A[hi]
if A[mid] < A[hi]
    swap A[mid] with A[hi]
pivot := A[hi]
```

Ý tưởng của đoạn mã: Đưa giá trị trung vị vào vị trí $A[hi]$, sau đó giá trị mới của $A[hi]$ được sử dụng làm pivot.

Tối ưu thuật toán Quick Sort

Một số cách để tối ưu được Sedgewick đề xuất và được sử dụng rộng rãi trong thực tế, bao gồm:

1. Để đảm bảo sử dụng tối đa $O(log n)$ bộ nhớ, trước tiên hãy thực hiện đệ quy với phía nhỏ hơn của phân vùng, sau đó sử dụng gọi đuôi (tail call) để đệ quy với phía còn lại, hoặc cập nhật các tham số để không bao gồm phía nhỏ hơn đã được sắp xếp, và lặp lại để sắp xếp phía lớn hơn.

2. Khi số lượng phần tử nhỏ hơn một ngưỡng nhất định (khoảng 10 phần tử), chuyển sang một thuật toán sắp xếp không đệ quy như insertion sort. Thuật toán này thực hiện ít phép hoán đổi, so sánh hoặc các thao tác khác trên mảng nhỏ.

3. Biến thể cũ hơn của tối ưu hoá trên: Khi số lượng phần tử nhỏ hơn ngưỡng k, chỉ cần dừng lại; sau khi toàn bộ mảng đã được xử lý, thực hiện insertion sort trên mảng đó. Việc dừng đệ quy sớm sẽ khiến mảng đạt trạng thái k-sorted (mỗi phần tử cách vị trí cuối cùng của nó tối đa k vị trí). Trong trường hợp này, insertion sort mất thời gian $O(kn)$ để hoàn tất, nghĩa là tuy nhiên k là một hằng số. So với phương pháp tối ưu hoá “nhiều lần sắp xếp nhỏ”, phiên bản này có thể thực hiện ít lệnh hơn, nhưng sử dụng bộ nhớ đệm (cache) trong các máy tính hiện đại không hiệu quả bằng.

Mã C++ cho phiên bản tối ưu:

1. Sử dụng **Hoare's scheme** để giảm số phép swap ở mỗi bước phân vùng.
2. Chọn pivot theo quy tắc **Median-Of-Three** để phân hoạch cân bằng.
3. Sử dụng **Tail Call Optimization** để sử dụng tối đa $O(log n)$ bộ nhớ
4. Sử dụng **Insertion Sort** cho mảng nhỏ (khi giá trị ngưỡng threshold = 10) để tận dụng tối đa ưu thế của cache.

```
void quickSort(std::vector<int> &arr, int lo, int hi, const int threshold) {
    while (lo < hi) {
        if (hi - lo + 1 <= threshold) {
            insertionSort(arr, lo, hi);
            return;
        }

        int p = partition(arr, lo, hi);
        if (p - lo < hi - p) {
            quickSort(arr, lo, p, threshold);
            lo = p + 1;
        } else {
            quickSort(arr, p + 1, hi, threshold);
            hi = p;
        }
    }
}
```

```

        }
    }

int partition(std::vector<int> &arr, int lo, int hi) {
    int pivot = medianOfThree(arr, lo, hi);
    int i = lo - 1;
    int j = hi + 1;
    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);
        do {
            j--;
        } while (arr[j] > pivot);

        if (i >= j) return j;

        std::swap(arr[i], arr[j]);
    }
}

int medianOfThree(std::vector<int> &arr, int lo, int hi) {
    int mid = lo + (hi - lo) / 2;

    if (arr[mid] < arr[lo]) {
        std::swap(arr[mid], arr[lo]);
    }
    if (arr[hi] < arr[lo]) {
        std::swap(arr[lo], arr[hi]);
    }
    if (arr[mid] < arr[hi]) {
        std::swap(arr[mid], arr[hi]);
    }
    return arr[hi];
}

```

2.3.4. Ứng dụng

1. Xử lý dữ liệu lớn: Nhờ tốc độ trung bình nhanh và yêu cầu bộ nhớ thấp, Quick Sort thường được dùng để sắp xếp các tệp dữ liệu lớn trong thực tế.

2. Các thư viện chuẩn: Thường là thuật toán được sử dụng làm nền tảng cho hàm sắp xếp trong các ngôn ngữ lập trình như C++, Java và Python.

3. Hệ thống thời gian thực: Hiệu suất cao làm cho Quick Sort phù hợp với các ứng dụng yêu cầu xử lý dữ liệu nhanh, như hệ thống giao dịch tài chính hoặc xử lý dữ liệu thời gian thực.

4. Ứng dụng trên phần cứng hạn chế: Nhờ yêu cầu bộ nhớ thấp, Quick Sort phù hợp để triển khai trên thiết bị có tài nguyên hạn chế, như hệ thống nhúng (embedded systems).

Tóm lại

Thuật toán	Tại chỗ (in-place)	Ôn định (Stable)	Memory	Best	Average	Worst	Method
Bubble Sort	Yes	Yes	$O(1)$	$O(n)$	$O(n^2)$	$O(n^2)$	Exchanging
Insertion Sort	Yes	Yes	$O(1)$	$O(n^2)$	$O(n)$	$O(n^2)$	Insertion
Quick Sort	Yes	No	$O(log n)$	$O(n log n)$	$O(n log n)$	$O(n^2)$	Partitioning

3. Xây dựng chương trình

Xây dựng chương trình kiểm thử thuật toán sắp xếp có số lần lựa chọn vô hạn bao gồm các mục sau:

1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.

Mô tả quy trình thực thi của chương trình như sau:

1. Nếu người dùng nhập “1” (Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp)

1.1. Chương trình sẽ thông báo cần import một file input .txt có số lượng lớn hơn 10,000 phần tử. Người dùng cần nhập tên file input có đuôi .txt.

Nếu người dùng nhập đúng file có đuôi .txt thì sẽ chuyển đến **bước 1.2.**

Ngược lại, nếu nhập sai thì hệ thống sẽ yêu cầu nhập lại cho đến khi tên file hợp lệ.

1.2. Chương trình sẽ yêu cầu người dùng nhập số lượng phần tử muốn sắp xếp. Người dùng nhập số lượng phần tử muốn sắp xếp.

Nếu số phần tử lớn hơn 10000 thì sẽ chuyển đến **bước 1.3.**

Ngược lại, nếu nhập sai thì chương trình yêu cầu nhập lại cho đến khi số phần tử là hợp lệ.

1.3. Chương trình sẽ yêu cầu người dùng lựa chọn một trong ba thuật toán để sắp xếp. Người dùng sẽ chọn một trong ba thuật toán sắp xếp.

- a. Thuật toán Bubble Sort
- b. Thuật toán Insertion Sort
- c. Thuật toán Quick Sort

Nếu người dùng nhập đúng một trong 3 ký tự a, b hoặc c thì sẽ chuyển đến **bước 1.4.**

Ngược lại, nếu nhập sai thì chương trình sẽ yêu cầu nhập lại cho đến khi lựa chọn thuật toán là hợp lệ.

1.4. Chương trình sẽ hiển thị thời gian thực thi của thuật toán tương ứng sau khi chạy xong và yêu cầu người dùng nhập tên file output muốn lưu kết quả chạy thuật toán.

Nếu người dùng nhập đúng tên file (có đuôi .txt) thì **bắt đầu vòng lặp mới, hiện thị lại menu chương trình**.

Ngược lại, nếu nhập sai thì chương trình yêu cầu nhập lại cho đến khi tên file hợp lệ.

2. Nếu người dùng chọn “2” (Đo lường thời gian thực thi của 03 thuật toán sắp xếp)

2.1. Chương trình sẽ yêu cầu người dùng chọn một trong ba input để đo lường thời gian thực thi của ba thuật toán. Người dùng sẽ nhập a, b hoặc c để chọn một trong ba input:

- a. input n = 10000
- b. input n = 100000
- c. input n = 1000000.

Nếu người dùng nhập đúng thì chương trình sẽ chuyển đến bước 2.2.

Ngược lại, nếu nhập sai thì chương trình sẽ yêu cầu nhập lại cho đến khi lựa chọn input là hợp lệ.

2.2. Chương trình thực hiện import file:

“**input1e4.txt**” chứa 10,000 phần tử được sinh ngẫu nhiên nếu người dùng chọn input n = 10,000

“**input1e5.txt**” chứa 10,0000 phần tử được sinh ngẫu nhiên nếu người dùng chọn input n = 100,000

“**input1e6.txt**” chứa 1,000,000 phần tử được sinh ngẫu nhiên nếu người dùng chọn input n = 1,000,000

Khi đọc xong file thì chương trình sẽ thông báo đã đọc xong file tương ứng, rồi chuyển đến bước 2.3

2.3. Chương trình thực hiện đo thời gian thực thi của 03 thuật toán

Chương trình hiển thị lần lượt thời gian thực thi của mỗi thuật toán sau khi thuật toán đó chạy xong.

Ví dụ: Với đầu vào input n = 10000

Thời gian thực thi của Bubble Sort: 0.791255 giây

Thời gian thực thi của Insertion Sort: 0.805066 giây

Thời gian thực thi của Quick Sort: 0.0017004 giây

Sau khi đã đo xong thời gian thực thi của ba thuật toán, hiện thị ra thông báo đã đo xong thời gian thực thi của ba thuật toán. Sau đó chuyển đến **bước 2.4**

2.4. Chương trình sẽ kiểm tra tính chính xác của thuật toán.

Để đảm bảo thời gian thực thi là chính xác. Chương trình sẽ kiểm tra tính chính xác bằng cách so sánh ba mảng đã sắp xếp với nhau.

Nếu ba mảng đã sắp xếp như nhau thì chuyển đến **bước 2.5**.

Nếu không thì sẽ thông báo “Có sự khác biệt giữa các thuật toán sắp xếp, không thực hiện lưu kết quả vào file”

2.5. Chương trình sẽ ghi kết quả quá trình chạy cả 03 thuật toán vào file:

“**output1e4.txt**” nếu người dùng chọn input n = 10,000.

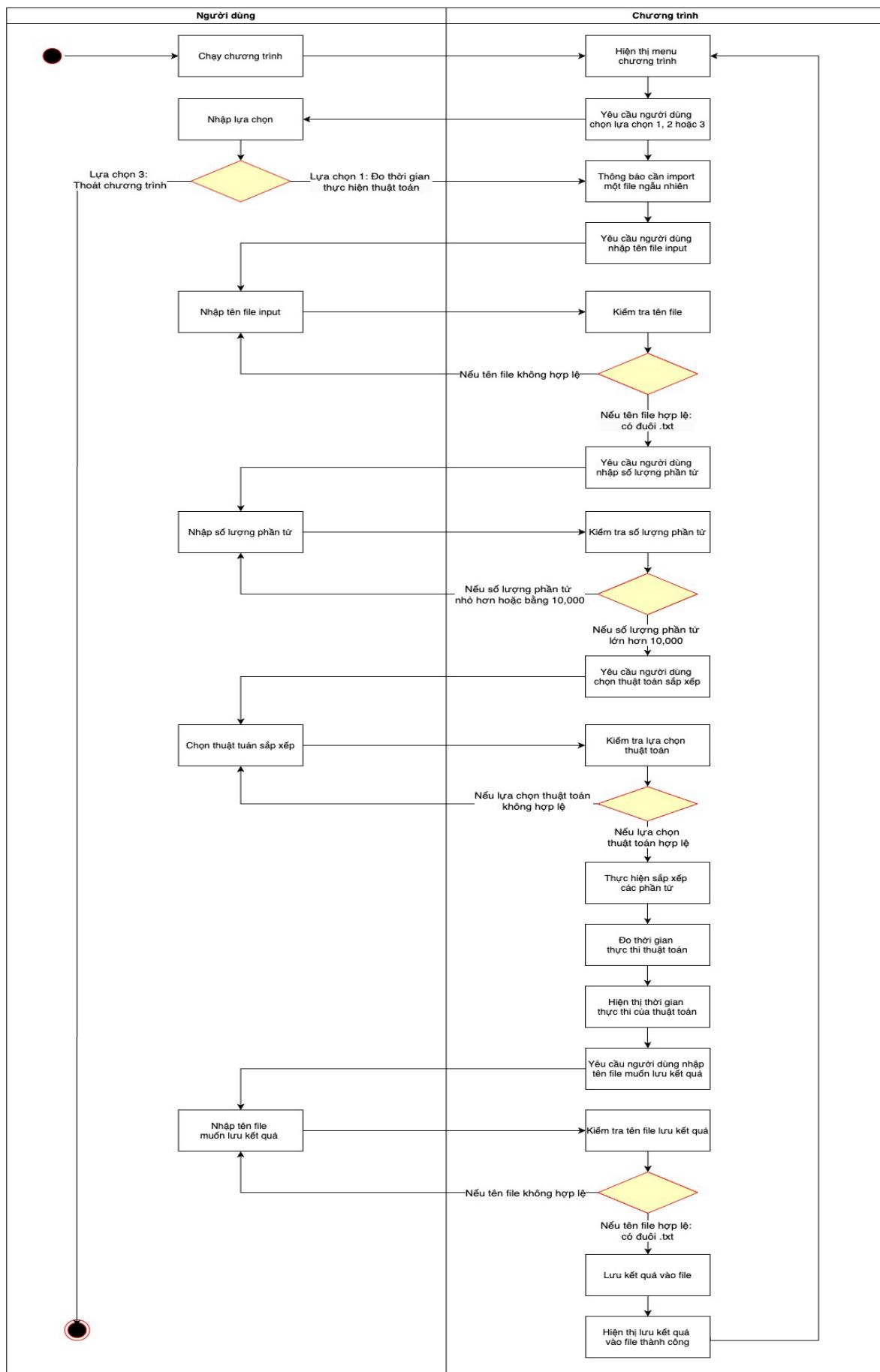
“**output1e5.txt**” nếu người dùng chọn input n = 100,000.

“**output1e6.txt**” nếu người dùng chọn input n = 1,000,000.

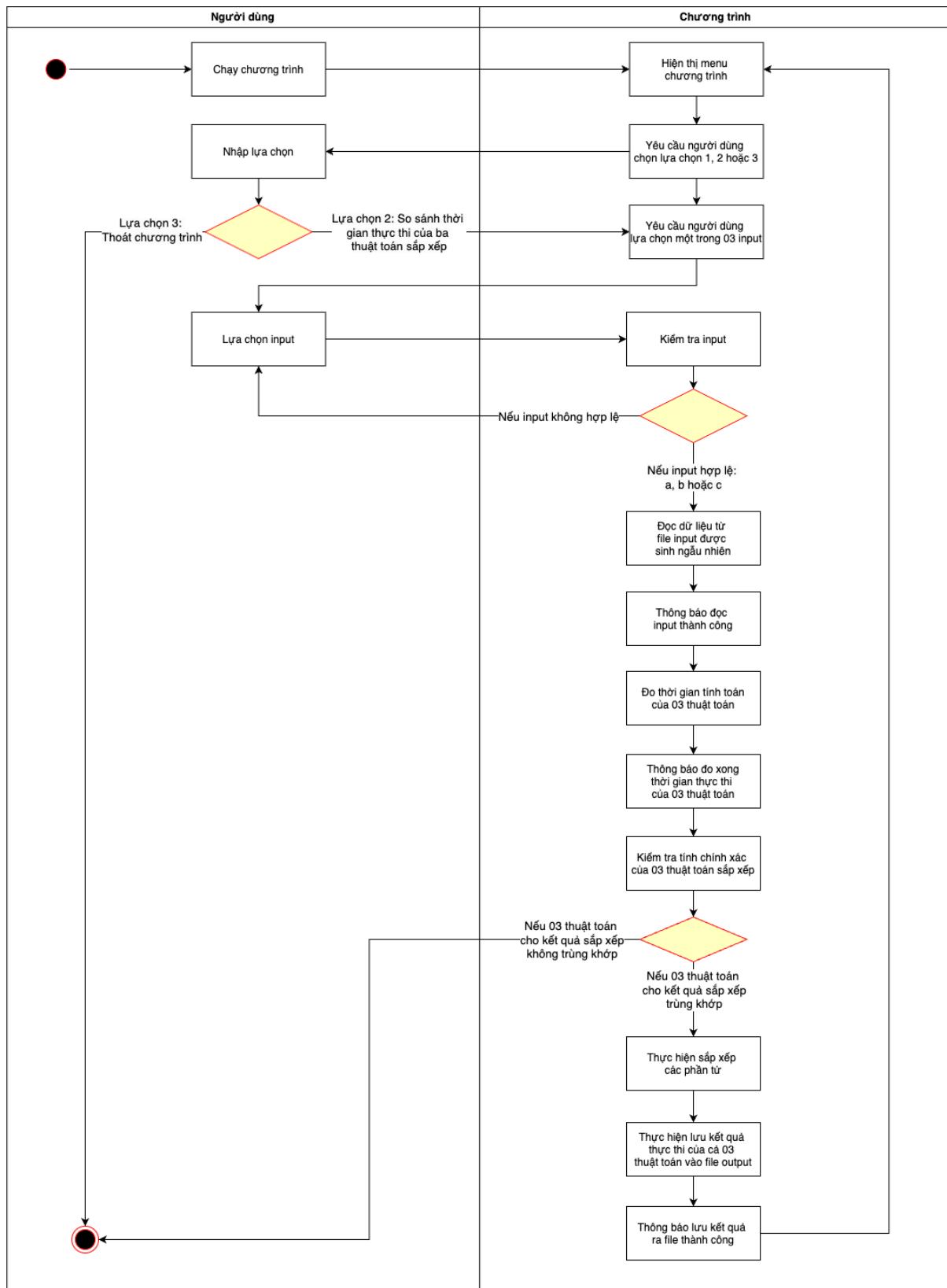
Sau khi ghi xong kết quả ra file, chương trình sẽ thông báo đã lưu kết quả vào file thành công. Sau đó bắt đầu vòng lặp mới, hiển thị lại menu chương trình.

Biểu đồ Activity Diagram

Quy trình đo lường thời gian thực thi của ba thuật toán sắp xếp

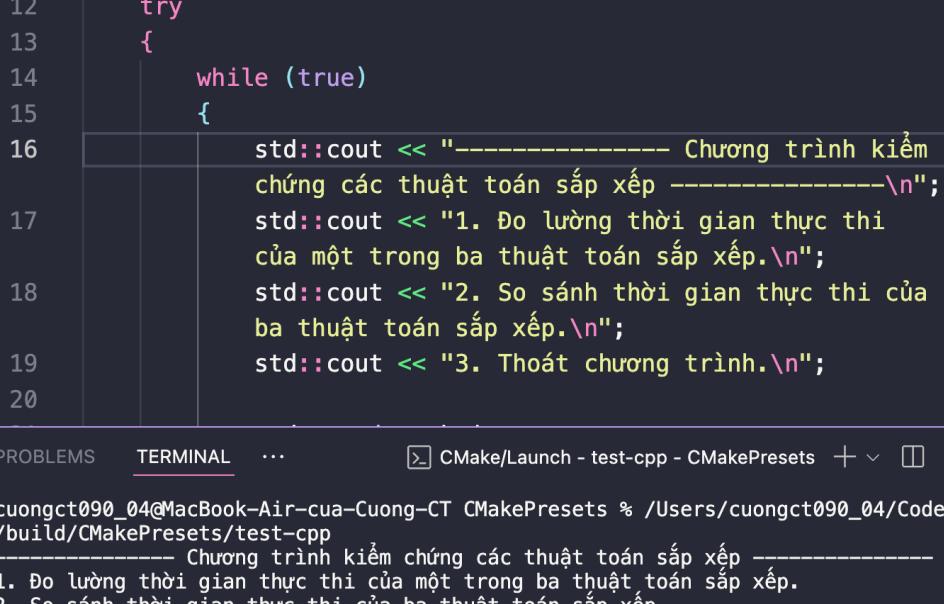


Quy trình so sánh thời gian thực thi của ba thuật toán sắp xếp



4. Demo chương trình

Quy trình đo lường thời gian thực thi của một trong ba thuật toán sắp xếp



The screenshot shows a portion of a C++ program in a code editor. The code includes a try-catch block, a while loop, and std::cout statements for displaying menu options related to sorting algorithms. The code editor has a dark theme with syntax highlighting. Below the code, the terminal shows the output of the program, which lists three menu options: 1. Measure execution time of one of the three sorting algorithms, 2. Compare execution times of the three sorting algorithms, and 3. Exit the program. It also prompts the user to enter a choice.

```
10 int main(int, char **)
11 {
12     try
13     {
14         while (true)
15         {
16             std::cout << "----- Chương trình kiểm
17             chứng các thuật toán sắp xếp -----<\n";
18             std::cout << "1. Đo lường thời gian thực thi
19             của một trong ba thuật toán sắp xếp.\n";
20             std::cout << "2. So sánh thời gian thực thi của
21             ba thuật toán sắp xếp.\n";
22             std::cout << "3. Thoát chương trình.\n";
23         }
24     }
25 }
```

PROBLEMS TERMINAL ...

CMake/Launch - test-cpp - CMakePresets + ⌂ ⌄ ⌁ ⌂ ⌁ ⌂ ⌁

- cuongct090_04@MacBook-Air-cua-Cuong-CT CMakePresets % /Users/cuongct090_04/Code/test/out/build/CMakePresets/test-cpp ----- Chương trình kiểm chứng các thuật toán sắp xếp -----
 1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
 2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
 3. Thoát chương trình.

--> Vui lòng nhập lựa chọn của bạn: █

Người dùng cần chọn 1 để tiếp tục chương trình. Sau khi chọn 1 chương trình sẽ hiển thị thông báo cần import một file ngẫu nhiên lớn hơn 10,000 phần tử.

Giả sử người dùng nhập một file có tên cuonginput, chương trình sẽ thông báo tên file phải có đuôi .txt và yêu cầu nhập lại.

PROBLEMS TERMINAL ...

CMake/Launch - test-cpp - CMakePresets + ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

cuongct090_04@MacBook-Air-cua-Cuong-CT CMakePresets % /Users/cuongct090_04/Code/test/o
ut/build/CMakePresets/test-cpp

----- Chương trình kiểm chứng các thuật toán sắp xếp -----

1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.

--> Vui lòng nhập lựa chọn của bạn: 1

Chương trình cần import một file ngẫu nhiên lớn hơn 10,000 phần tử
Hãy nhập tên file dữ liệu đầu vào (ví dụ input.txt): cuonginput
Lỗi: Tên file phải có đuôi .txt. Vui lòng nhập lại!
Hãy nhập tên file dữ liệu đầu vào (ví dụ input.txt): cuonginput.txt
Hãy nhập số lượng phần tử cần sắp xếp: ■

Sau khi nhập đúng tên file có đuôi .txt, người dùng sẽ cần phải nhập số lượng phần tử cần sắp xếp. Nếu số lượng nhập vào nhỏ hơn hoặc bằng 10,000 thì chương trình yêu cầu nhập lại.

```
----- Chương trình kiêm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 1

Chương trình cần import một file ngẫu nhiên lớn hơn 10,000 phần tử
Hãy nhập tên file dữ liệu đầu vào (ví dụ input.txt): cuonginput
Lỗi: Tên file phải có đuôi .txt. Vui lòng nhập lại!
Hãy nhập tên file dữ liệu đầu vào (ví dụ input.txt): cuonginput.txt
Hãy nhập số lượng phần tử cần sắp xếp: 10000
Lỗi: Số lượng phần tử phải lớn hơn 10,000. Vui lòng nhập lại!
Hãy nhập số lượng phần tử cần sắp xếp: 999999
Lựa chọn một trong 03 thuật toán sau:
    a. Bubble Sort
    b. Insertion Sort
    c. Quick Sort
--> Vui lòng nhập lựa chọn của bạn: ■
```

Sau đó, người dùng cần chọn a, b hoặc c để chọn một trong ba thuật toán để sắp xếp.

----- Chương trình kiểm chứng các thuật toán sắp xếp -----

- Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
- So sánh thời gian thực thi của ba thuật toán sắp xếp.
- Thoát chương trình.

--> Vui lòng nhập lựa chọn của bạn: 1

Chương trình cần import một file ngẫu nhiên lớn hơn 10,000 phần tử
Hãy nhập tên file dữ liệu đầu vào (ví dụ input.txt): cuonginput

Lỗi: Tên file phải có đuôi .txt. Vui lòng nhập lại!

Hãy nhập tên file dữ liệu đầu vào (ví dụ input.txt): cuonginput.txt

Hãy nhập số lượng phần tử cần sắp xếp: 10000

Lỗi: Số lượng phần tử phải lớn hơn 10,000. Vui lòng nhập lại!

Hãy nhập số lượng phần tử cần sắp xếp: 9999999

Lựa chọn một trong 03 thuật toán sau:

- Bubble Sort
- Insertion Sort
- Quick Sort

--> Vui lòng nhập lựa chọn của bạn: c

Thời gian thực thi của thuật toán Quick Sort với đầu vào n = 9999999: 2.91185 giây.

Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt):

Khi đó thời gian xong, người dùng cần nhập tên file muốn lưu kết quả. Giả sử người dùng nhập tên file cuongoutput, chương trình thông báo tên file phải có đuôi .txt và yêu cầu nhập lại.

Sau khi thực hiện xong thì ta thấy cả file **cuonginput.txt**, **cuongoutput.txt** đều nằm trong thư mục **out/build/CmakePresets**

```
752776571 -349410496 544076189 639389541 794230558 578939744 -990586537 -635705139  
692536517 -956449610 661476994 394488136 -641278275 -707157256 -246318888  
653625603 -698838648 -275110526 405083103 -196151892 -218915931 530883325  
126439948 385197433 133844668 654816250 533861823 389968037 825404355 154152003  
-959988131 -831925650 171592013 253697181 -38183387 246074278 -106434991 131143622  
-915311980 146512860 65769206 568110421 554894316 -823842296 -235644399 -105639329  
774825506 -817198899 190044301 -678266087 -665627638 -491687624 416644843  
-832334578 -248792695 596627748 -953853085 990834858 -484741539 187842344  
-562568415 -703322775 877898959 804566464 -921506041 -36857573 -750747696  
-892158638 697031229 87909373 -845934649 902666862 -620208121 -380114784  
-922312738 548278520 707997444 277620213 675824548 297012842 -373054896 337487396  
843000493 -914203969 -619733474 -629514848 895555078 -511649559 -462490260  
-611897895 -745537200 -757293053 938509437 -698134793 845332757 -370765799  
751879353 -115460086 -571008108 -664849713 503603581 -504930324 344110703  
447986436 973378182 79016521 794788844 -969852995 737524514 55723503 -625652024  
345888179 876665389 241879333 233616938 434038405 282386550 220542641 809732251  
-785148337 99767452 395185856 319197951 808755125 -413614733 -263638884 177998468  
-884200542 -348237420 -84502882 483108784 496636352 -901385914 756248571 -31026100  
034766760 125607808 670655602 0700200 75261056 92100015 10756056 826870612
```

b Insertion Sort
c. Quick Sort
--> Vui lòng nhập lựa chọn của bạn: c
Thời gian thực thi của thuật toán Quick Sort với đầu vào n = 9999999: 2.91185 giây.
Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt): cuongoutput
Lỗi: Tên file phải có đuôi .txt. Vui lòng nhập lại!
Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt): cuongoutput.txt
Đã lưu kết quả vào file thành công.

Ln 1, Col 103886428 Spaces: 4 UTF-8 LF Plain Text Go Live

```
1 Thời gian thực thi của thuật toán Quick Sort với input n = 9999999: 2.911846 giây  
2
```

b Insertion Sort
c. Quick Sort
--> Vui lòng nhập lựa chọn của bạn: c
Thời gian thực thi của thuật toán Quick Sort với đầu vào n = 9999999: 2.91185 giây.
Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt): cuongoutput
Lỗi: Tên file phải có đuôi .txt. Vui lòng nhập lại!
Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt): cuongoutput.txt
Đã lưu kết quả vào file thành công.

Ln 1, Col 1 Spaces: 4 UTF-8 LF Plain Text Go Live

Quy trình so sánh thời gian thực thi của ba thuật toán

Người dùng cần chọn 2 để tiếp tục chương trình

The screenshot shows a terminal window in VS Code. The title bar says "CMake/Launch - test-cpp - CMakePresets". The terminal content is as follows:

```
Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt): cuongoutput
Lỗi: Tên file phải có đuôi .txt. Vui lòng nhập lại!

Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt): cuongoutput.txt
Đã lưu kết quả vào file thành công.

----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 2

Lựa chọn các input sau:
  a. n = 10,000
  b. n = 100,000
  c. n = 1,000,000
--> Vui lòng lựa chọn input của bạn: 
```

Người dùng cần lựa chọn a, b hoặc c để chọn một trong ba input.

The screenshot shows a terminal window in VS Code. The title bar says "CMake/Launch - test-cpp - CMakePresets". The terminal content is as follows:

```
Hãy nhập tên file bạn muốn lưu kết quả (ví dụ: output.txt): cuongoutput.txt
Đã lưu kết quả vào file thành công.

----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 2

Lựa chọn các input sau:
  a. n = 10,000
  b. n = 100,000
  c. n = 1,000,000
--> Vui lòng lựa chọn input của bạn: b
Đã đọc xong dữ liệu từ file input1e5.txt

```

Dữ liệu sẽ được sinh ngẫu nhiên vào file **input1e5.txt** chương trình sẽ đọc dữ liệu từ file này. Sau đó, chương trình sẽ thực hiện đo thời gian thực thi của 03 thuật toán với đầu vào input đã chọn, mỗi khi đo xong một thuật toán thì sẽ hiển thị thông báo. Và để chắc chắn dữ liệu là chính xác thì trước khi lưu kết quả vào file, chương trình sẽ kiểm tra dữ liệu đã sắp xếp có trùng khớp hay không. Nếu trùng khớp thì thực hiện lưu kết quả vào file **output1e5.txt**.

```
----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 2

Lựa chọn các input sau:
    a. n = 10,000
    b. n = 100,000
    c. n = 1,000,000
--> Vui lòng lựa chọn input của bạn: b
Đã đọc xong dữ liệu từ file input1e5.txt
thời gian thực thi của thuật toán Bubble Sort: 81.5137 giây.
Thời gian thực thi của thuật toán Insertion Sort: 14.7354 giây.
Thời gian thực thi của thuật toán Quick Sort: 0.0214287 giây.
Đã đo xong thời gian thực thi của 03 thuật toán.
Tất cả các thuật toán sắp xếp cho cùng một kết quả!
Đã lưu kết quả vào file output1e5.txt thành công.

----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 
```

Sau khi thực hiện xong, trong thư mục **out/build/CmakePresets** sẽ chứa file **input1e5.txt** và file **output1e5.txt**

```
out > build > CMakePresets > output > output1e5.txt
5 Thời gian thực thi của thuật toán với input n = 100000:
8   3. quick Sort: 0.021527/ giây
9 Thời gian thực thi của thuật toán với input n = 100000:
10  1. Bubble Sort: 81.026108 giây
11  2. Insertion Sort: 15.389414 giây
12  3. Quick Sort: 0.021141 giây
13 Thời gian thực thi của thuật toán với input n = 100000:
14  1. Bubble Sort: 81.627398 giây
15  2. Insertion Sort: 14.692205 giây
16  3. Quick Sort: 0.020965 giây
17 Thời gian thực thi của thuật toán với input n = 100000:
18  1. Bubble Sort: 81.513726 giây
19  2. Insertion Sort: 14.735435 giây
20  3. Quick Sort: 0.021429 giây
```

Lựa chọn các input sau:
a. n = 10,000
b. n = 100,000
c. n = 1,000,000
--> Vui lòng lựa chọn input của bạn: b
Đã đọc xong dữ liệu từ file input1e5.txt
Thời gian thực thi của thuật toán Bubble Sort: 81.5137 giây.
Thời gian thực thi của thuật toán Insertion Sort: 14.7354 giây.
Thời gian thực thi của thuật toán Quick Sort: 0.0214287 giây.
Đã đọ xong thời gian thực thi của 03 thuật toán.
Tất cả các thuật toán sắp xếp cho cùng một kết quả!
Đã lưu kết quả vào file output1e5.txt thành công.

----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 1
Chương trình kết thúc thành công.

Đến đây, nếu người dùng chọn 3 thì chương trình sẽ kết thúc.

```
b. n = 100,000
c. n = 1,000,000
--> Vui lòng lựa chọn input của bạn: b
Đã đọc xong dữ liệu từ file input1e5.txt
Thời gian thực thi của thuật toán Bubble Sort: 81.5137 giây.
Thời gian thực thi của thuật toán Insertion Sort: 14.7354 giây.
Thời gian thực thi của thuật toán Quick Sort: 0.0214287 giây.
Đã đọ xong thời gian thực thi của 03 thuật toán.
Tất cả các thuật toán sắp xếp cho cùng một kết quả!
Đã lưu kết quả vào file output1e5.txt thành công.

----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 3
Chương trình kết thúc thành công.
```

cuongct090_04@MacBook-Air-cua-Cuong-CT CMakePresets %

5. Kết quả thực nghiệm

5.1. Với input $n = 10,000$

The screenshot shows a terminal window with the following content:

```
out > build > CMakePresets > output > output1e4.txt
1 Thời gian thực thi của thuật toán với input n = 10000:
2   1. Bubble Sort: 0.791255 giây
3   2. Insertion Sort: 0.143463 giây
4   3. Quick Sort: 0.001704 giây
5 Thời gian thực thi của thuật toán với input n = 10000:
6   1. Bubble Sort: 0.805066 giây
7   2. Insertion Sort: 0.144528 giây
8   3. Quick Sort: 0.001722 giây
9 Thời gian thực thi của thuật toán với input n = 10000:
10  1. Bubble Sort: 0.800704 giây
11  2. Insertion Sort: 0.146046 giây
12  3. Quick Sort: 0.001962 giây
13 Thời gian thực thi của thuật toán với input n = 10000:
14  1. Bubble Sort: 0.804272 giây
15  2. Insertion Sort: 0.146086 giây
16  3. Quick Sort: 0.001694 giây
17 Thời gian thực thi của thuật toán với input n = 10000:
18  1. Bubble Sort: 0.798854 giây
19  2. Insertion Sort: 0.145219 giây
20  3. Quick Sort: 0.001713 giây
```

Below the terminal, there is a message in Vietnamese:

Đã do xong thời gian thực thi của 03 thuật toán.
Tất cả các thuật toán sắp xếp cho cùng một kết quả!
Đã lưu kết quả vào file output1e4.txt thành công.

Chương trình kiểm chứng các thuật toán sắp xếp

- Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
- So sánh thời gian thực thi của ba thuật toán sắp xếp.
- Thoát chương trình.

Vui lòng nhập lựa chọn của bạn:

5.2. Với input n = 100,000

The screenshot shows a terminal window with the following content:

```
← → ⚡ test
EXPLORER ... C main.cpp output1e5.txt x output1e6.txt C++ exec-time.cpp h exec-time...
TEST
  out / build / CMakePr...
    input
      input1e5.txt
      input1e6.txt
    output
      cuong1.txt
      cuong2.txt
      cuongoutput.txt
      output1e4.txt
      output1e5.txt
      output1e6.txt
  Testing
    > 20241226-0614
    > 20241228-0449
    > 20241229-0457
    > Temporary
      TAG
    cmake_install.cmake
OUTLINE
TIMELINE

1 Thời gian thực thi của thuật toán với input n = 100000:
  1. Bubble Sort: 82.397207 giây
  2. Insertion Sort: 15.801367 giây
  3. Quick Sort: 0.020935 giây
2 Thời gian thực thi của thuật toán với input n = 100000:
  1. Bubble Sort: 81.473876 giây
  2. Insertion Sort: 14.691303 giây
  3. Quick Sort: 0.021327 giây
3 Thời gian thực thi của thuật toán với input n = 100000:
  1. Bubble Sort: 81.026108 giây
  2. Insertion Sort: 15.389414 giây
  3. Quick Sort: 0.021141 giây
4 Thời gian thực thi của thuật toán với input n = 100000:
  1. Bubble Sort: 81.627398 giây
  2. Insertion Sort: 14.692205 giây
  3. Quick Sort: 0.020965 giây
5 Thời gian thực thi của thuật toán với input n = 100000:
  1. Bubble Sort: 81.513726 giây
  2. Insertion Sort: 14.735435 giây
  3. Quick Sort: 0.021429 giây

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS CMake/Launch
Đã do xong thời gian thực thi của 03 thuật toán.
Tất cả các thuật toán sắp xếp cho cùng một kết quả!
Đã lưu kết quả vào file output1e4.txt thành công.
----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của 3 thuật toán sắp xếp.
3. Thoát chương trình.
--> Vui lòng nhập lựa chọn của bạn: 1
Launchpad 0 △ 0 ⌂ Build ⌂ ⌂ Ln 21, Col 1 Tab Size: 4
```

5.3. Với input n = 1,000,000

```
out > build > CMakePresets > output > output1e6.txt
1 Thời gian thực thi của thuật toán với input n = 1000000:
2   1. Bubble Sort: 9158.092726 giây
3   2. Insertion Sort: 1482.784329 giây
4   3. Quick Sort: 0.248446 giây
5 Thời gian thực thi của thuật toán với input n = 1000000:
6   1. Bubble Sort: 9395.880286 giây
7   2. Insertion Sort: 1516.446106 giây
8   3. Quick Sort: 0.351440 giây
9 Thời gian thực thi của thuật toán với input n = 1000000:
10  1. Bubble Sort: 8482.625371 giây
11  2. Insertion Sort: 1477.487124 giây
12  3. Quick Sort: 0.247327 giây
13

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS CMake/Launch
--> Vui lòng nhập lựa chọn của bạn: 2
Lựa chọn các input sau:
a. n = 10,000
b. n = 100,000
c. n = 1,000,000
-> Vui lòng lựa chọn input của bạn: a
Đã đọc xong dữ liệu từ file input1e4.txt
Thời gian thực thi của thuật toán Bubble Sort: 0.798854 giây.
Thời gian thực thi của thuật toán Insertion Sort: 0.145219 giây.
Thời gian thực thi của thuật toán Quick Sort: 0.00171287 giây.
Đã xong thời gian thực thi của 03 thuật toán.
Tất cả các thuật toán sắp xếp cho cùng một kết quả!
Đã lưu kết quả vào file output1e4.txt thành công.

----- Chương trình kiểm chứng các thuật toán sắp xếp -----
1. Đo lường thời gian thực thi của một trong ba thuật toán sắp xếp.
2. So sánh thời gian thực thi của ba thuật toán sắp xếp.
3. Thoát chương trình.
-> Vui lòng nhập lựa chọn của bạn: 1
Ln 13, Col 1 Tab Size:
```

Nhận xét

Với 02 thuật toán sắp xếp **Bubble Sort** và **Insertion Sort** có độ phức tạp trung bình là $O(n^2)$ thì chênh lệnh thời gian thực thi giữa các đầu vào input vào khoảng $10^2 = 100$ lần, tức là:

Thời gian thực thi trong trường hợp input n = 100,000 gấp khoảng 100 lần thời gian thực thi trong trường hợp input n = 10,000.

Thời gian thực thi trong trường hợp input n = 1,000,000 gấp khoảng 100 lần thời gian thực thi trong trường hợp input n = 100,000.

Với thuật toán **QuickSort** có độ phức tạp trung bình là $O(n \log n)$ thì chênh lệnh thời gian thực thi giữa các đầu vào input vào khoảng $10 \log_2 10 \approx 33.2193$ lần, tức là:

Thời gian thực thi trong trường hợp input n = 100,000 gấp khoảng 33.2193 lần thời gian thực thi trong trường hợp input n = 10,000.

Thời gian thực thi trong trường hợp input n = 1,000,000 gấp khoảng 33.2193 lần thời gian thực thi trong trường hợp input n = 100,000.

Tài liệu tham khảo

- https://en.wikipedia.org/wiki/Bubble_sort
- https://en.wikipedia.org/wiki/Insertion_sort
- <https://en.wikipedia.org/wiki/Quicksort>