ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

KIẾN TRÚC MÁY TÍNH (CO2007)

ĐỀ TÀI 5 BÀI TẬP LỚN: SẮP XẾP CHUỖI THEO THUẬT TOÁN QUICK SORT

Giảng viên hướng dẫn: Vũ Trọng Thiên

Nhóm 7 - Sinh viên thực hiện: Phạm Duy Quang - 2011899
Nguyễn Trọng Vinh - 2015070
Trần Đình Hậu - 1711261

Thành phố Hồ Chí Minh, Tháng 06 Năm 2022

Mục lục

1 CƠ SỞ LÝ THUYẾT

1.1 Bài toán sắp xếp

Sắp xếp (Sorting) là quá trình bố trí lại các phần tử của một tập hợp đối tượng nào đó theo một thứ tự nhất định. Chẳng hạn như thứ tự tăng dần (hoặc giảm dần) đối với một dãy số, thứ tự từ điển đối với các từ,... Yêu cầu về sắp xếp thường xuyên xuất hiện trong các ứng dụng của công nghệ thông tin đối với các mục đích khác nhau như: sắp xếp dữ liệu trên máy tính để việc tìm kiếm thuận lợi hơn, sắp xếp các kết quả xử lý để phục vụ việc trình chiếu, báo cáo,... Nói chung, dữ liệu cần được sắp xếp có thể ở nhiều dạng khác nhau, với tiêu chuẩn so sánh giữa 2 điểm dữ liệu khác nhau.

Có nhiều cách (thuật toán) để giải quyết cùng 1 bài toán sắp xếp, với những ưu, nhược điểm khác nhau. Thông thường, để so sánh các thuật toán với nhau, ta dựa vào các tiêu chí sau:

- **Thời gian chạy:** Đối với các dữ liệu rất lớn, các thuật toán không hiệu quả sẽ chạy rất chậm và không thể ứng dụng trong thực tế.
- **Bộ nhớ:** Các thuật toán nhanh đòi hỏi đệ quy sẽ có thể phải tạo ra các bản copy của dữ liệu đang xử lý. Với những hệ thống mà bộ nhớ có giới hạn (ví dụ: embedded system,...), một vài thuật toán sẽ không thể chạy được.
- **Độ ổn định (stability):** Độ ổn định được định nghĩa dựa trên điều gì sẽ xảy ra với các phần tử có giá trị giống nhau.
  - Đối với thuật toán sắp xếp ổn định, các phần tử bằng với giá trị bằng nhau sẽ giữ nguyên thứ tự trong mảng trước khi sắp xếp.
  - Đối với thuật toán sắp xếp không ổn định, các phần tử có giá trị bằng nhau sẽ có thể có thứ tự bất kỳ.

Dựa vào đó, người ta đã nghiên cứu, tìm ra rất nhiều thuật toán sắp xếp nổi bật như: Bubble Sort, Shell Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort,...

Trong giới hạn của bài báo cáo bài tập lớn này, nhóm sẽ hiện thực thuật toán Quick Sort trên một dãy 50 số nguyên bằng hợp ngữ MIPS với công cụ MARS.

1.2 Thuật toán Quick Sort

1.2.1 Mô tả

Thuật toán Quick Sort (còn có tên gọi khác là partition-exchange sort, hay phương pháp đổi chỗ từng phần) là một thuật toán sắp xếp phát triển bởi C.A.R Hoarec. Đúng như tên gọi, Quick Sort là một thuật toán rất hiệu quả (rất "nhanh") và rất thông dụng.

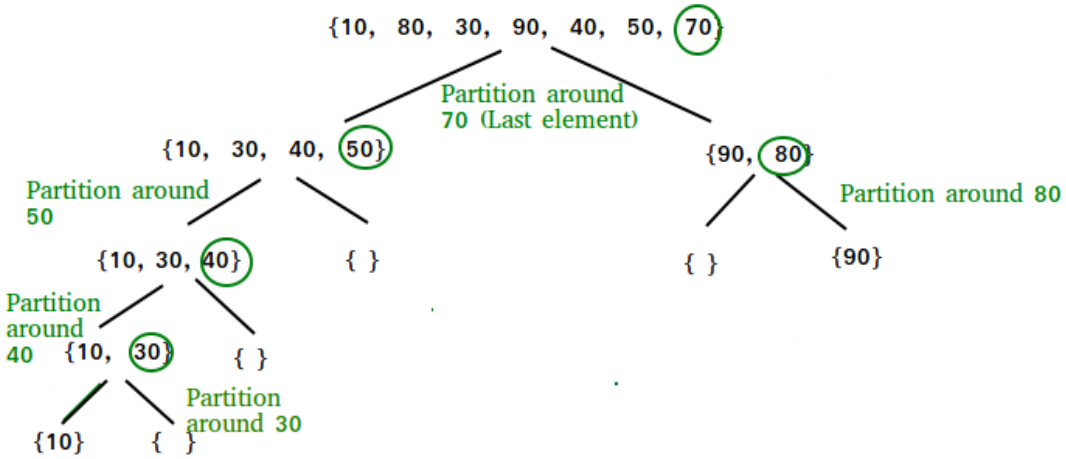
Ý tưởng chủ đạo của Quick Sort dựa trên cách mà con người hiện thực sắp xếp. Đầu tiên, chọn một điểm dữ liệu bất kì trong danh sách (ta gọi là một nút), giả sử là nút cuối cùng như trong hình gọi là nút làm trục (pivot node).

Tiếp theo chúng ta phân hoạch các nút còn lại trong danh sách cần sắp xếp sao cho từ vị trí 0 (danh sách bắt đầu từ vị trí 0 đến vị trí n – 1, với n là độ dài của danh sách) đến vị

trí pivot – 1 đều có nội dung nhỏ hơn hoặc bằng nút làm trục, các nút từ vị trí pivot + 1 đến n – 1 đều có nội dung lớn hơn nút làm trục.

Quá trình lại tiếp tục như thế với hai danh sách con từ trị trí 0 đến vị trí pivot – 1 và từ vị trí pivot + 1 đến vị trí n – 1... Sau cùng chúng ta sẽ được danh sách có thứ tự.

Mấu chốt của thuật toán là việc **chọn điểm pivot** thích hợp và **phân hạch dãy số**.

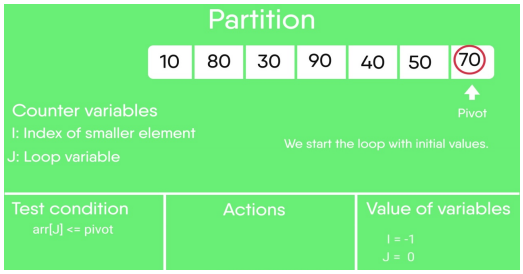


Hình 1: *Minh họa cách hoạt động của thuật toán Quick Sort*

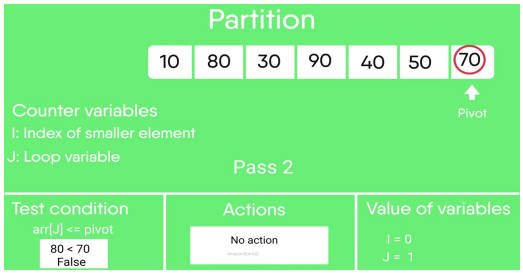
1.2.2 Hàm phân hoạch (phân đoạn) dãy số

Phương pháp Quick Sort phụ thuộc vào hai yếu tố chính là chọn pivot và cơ chế phân hoạch. Về cơ chế phân hoạch, thao tác này sẽ trả về một mảng và một phần tử pivot. Phần tử pivot sẽ được đặt đúng vị trí vào mảng đã sắp xếp. Tất cả các phần tử của mảng nhỏ hơn phần tử pivot sẽ nằm bên trái vị trí pivot và các phần tử của mảng lớn hơn phần tử pivot sẽ được chuyển sang bên phải vị trí pivot. Khi đó kết quả của một lần lập là hai mảng con, mảng bên trái của pivot và mảng bên phải pivot. Tiếp tục công việc với mỗi mảng con (chọn pivot và phân hoạch dãy số) cho tới khi mảng được sắp xếp.

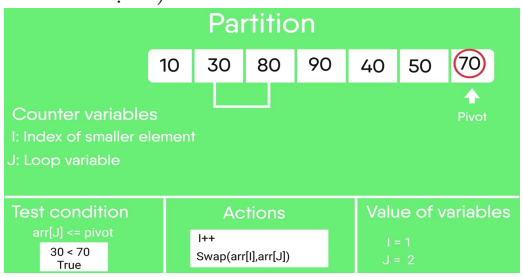
Có nhiều cách để hiện thực thuật toán phân hoạch (phân đoạn) dãy số. Ở đây nhóm sẽ trình bày phương pháp phân chia Lomuto, chính là cách mà nhóm đã hiện thực thuật toán Quick Sort bằng hợp ngữ MIPS. Với phương pháp phân chia này (Lomuto partition scheme), đầu tiên ta sẽ chọn pivot là điểm cuối cùng của danh sách. Thuật toán sẽ tạo 2 biến i và j. Biến i dùng để lưu vị trí (index) trong khi chúng ta lặp qua dãy số bằng biến j, sao cho những giá trị nhỏ hơn hoặc bằng pivot sẽ nằm từ vị trí bắt đầu đến i (pivot sẽ nằm ở vị trí i), những giá trị từ i + 1 đến hết dãy đang xét sẽ lớn hơn pivot. Nói cách khác, qua những lần lặp qua dãy thông qua biến j (bằng cách cho giá trị của j chạy từ vị trí bắt đầu đến kết thúc danh sách), nếu giá trị tại vị trí đang xét (vị trí j) nhỏ hơn hoặc bằng pivot, ta sẽ đổi chỗ nó với giá trị tại vị trí i và cộng thêm 1 vào i, nếu ngược lại giá trị tại j lớn hơn pivot, chúng ta sẽ bỏ qua đến vòng lặp tiếp theo.



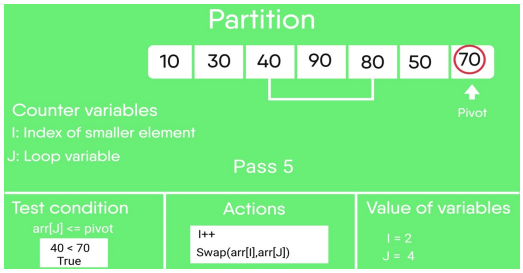
(a) Chọn pivot, khởi tạo giá trị ban đầu cho i và j, đổi chỗ giá trị đầu (nhỏ hơn pivot) với chính nó (lưu ý ở đây ta cộng i lên 1 trước rồi mới tiến hành đổi vị trí)



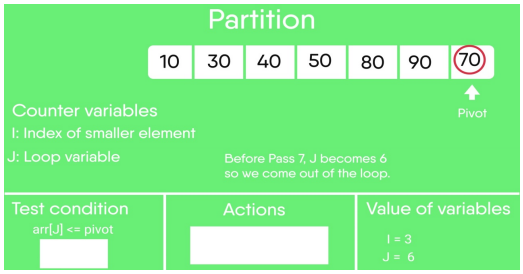
(b) Lần lặp thứ 2, giá trị tại ví trị j lúc này lớn hơn pivot nên ta không làm gì



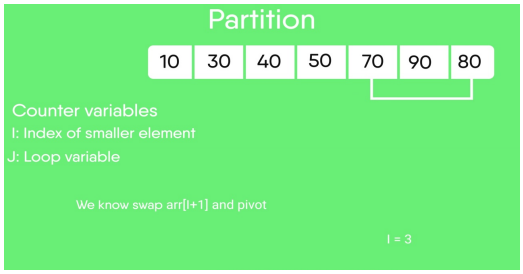
(c) Ở lần lặp này, $30 \leq 70$ nên ta sẽ cộng i lên 1 rồi đổi chỗ giá trị ở 2 vị trí



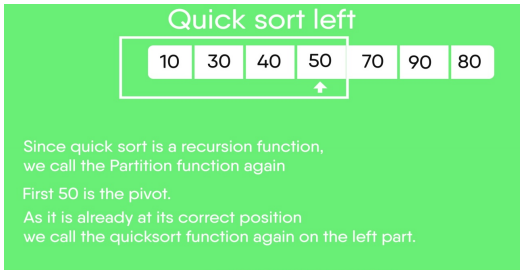
(d) 2 lần lặp tiếp theo không có gì thay đổi, ta bỏ qua đến lần lặp thứ 5, tiến hành đổi chỗ. Lưu ý qua từng lần lặp chỉ có giá trị của j được tăng lên, còn giá trị của i chỉ tăng lên 1 khi ở lần lặp đó có hoán đổi



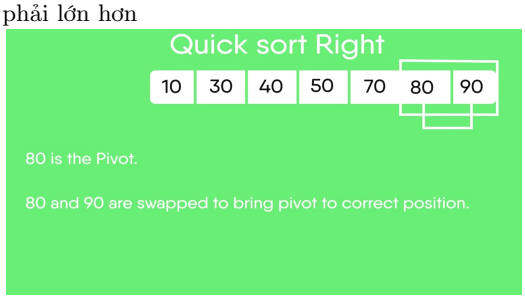
(e) Đến lần lặp cuối, vị trí j là vị trí của pivot, còn i là chỉ số cuối cùng trong dãy vừa sắp có giá trị nhỏ hơn pivot



(f) Cộng i lên 1 và hoán đổi vị trí i và j. Khi đó chúng ta đã đưa được pivot về đúng vị trí, và nửa bên trái sẽ nhỏ hơn hoặc bằng pivot, nửa bên phải lớn hơn



(g) Tiến hành gọi đệ quy lại hàm Quick Sort cho dãy bên trái pivot



(h) Và cho dãy bên phải pivot

Hình 2: *Minh họa cách hoạt động của phương pháp phân chia Lomuto*

1.2.3 Chọn điểm pivot

Quick Sort là một thuật toán chia để trị (Divide and Conquer). Nó sẽ chọn một điểm trong danh sách làm pivot và phân hoạch danh sách xung quanh điểm pivot này. Có nhiều cách để chọn điểm pivot khác nhau như:

- Luôn chọn điểm đầu tiên.
- Luôn chọn điểm cuối cùng **(và đây là cách hiện thực của nhóm)**.
- Chọn một điểm ngẫu nhiên.
- Chọn điểm trung vị.

Độ phức tạp về mặt thời gian (time complexity) của Quick Sort có thể được viết như sau:

$$\mathbf{T(n) = T(k) + T(n - k - 1) + O(n)}$$

Hai số hạng đầu tiên là của hai lần gọi đệ quy để gọi hàm Quick Sort cho vị trí từ đầu đến pivot – 1 (k số hạng) và từ pivot + 1 đến cuối dãy. Số hạng cuối cùng là thời gian tiêu tốn của một lần gọi hàm phân hoạch.

Người ta tính được rằng, trung bình thuật toán sẽ chạy với độ phức tạp là O(n log(n)). Trường hợp xấu nhất, tiêu tốn nhiều thời gian nhất sẽ xảy ra khi hàm phân hoạch luôn có pivot là giá trị lớn nhất hoặc nhỏ nhất trong dãy cần phân hoạch, hoặc bằng với tất cả các giá trị khác. Nếu chúng ta chọn cách chọn điểm pivot luôn là điểm đầu tiên hoặc điểm cuối cùng của dãy cần phân hoạch thì trường hợp xấu nhất xảy ra khi dãy đã được sắp xếp sẵn theo thứ tự tăng dần hoặc giảm dần, hoặc tất cả các phần tử trong dãy có giá trị bằng nhau. Công thức trên trở thành:

$$\mathbf{T(n) = T(0) + T(n - 1) + O(n)}$$

Tương đương với:

$$\mathbf{T(n) = T(n - 1) + O(n)}$$

Tức là thời gian tiêu tốn trong trường hợp này sẽ là $O(n^2)$, rất lớn nếu so với trường hợp trung bình O(n log(n)).

Chúng ta có thể cải tiến thuật toán bằng cách chọn pivot là một *điểm ngẫu nhiên*, nhưng khi đó độ ổn định của thuật toán (dù ban đầu thuật toán cũng đã là một thuật toán không ổn định) sẽ giảm xuống.

Một lựa chọn nữa là sử dụng *điểm trung vị* làm pivot. Ưu điểm của cách làm này là nó bảo đảm 2 phân hoạch sẽ có độ lớn (độ dài) xấp xỉ nhau. Tuy nhiên, để hiện thực nó, chúng ta cần phải biết tất cả các giá trị trong dãy để tìm trung vị, một việc rất khó và tiêu tốn nhiều tài nguyên để hiện thực trong thực tế.

Tóm lại, cách lựa chọn điểm pivot nào cũng sẽ có ưu, nhược điểm của nó, ta tùy vào mục đích sắp xếp mà từ đó chọn ra phương pháp phù hợp.

2 CẤU TRÚC CHƯƠNG TRÌNH

2.1 Vùng bộ nhớ .data:

Vùng bộ nhớ (.data) dùng để khai báo các biến, dữ liệu sẵn có. Vùng bộ nhớ khi sử dụng hợp ngữ MIPS và công cụ MARS sẽ luôn bắt đầu từ địa chỉ 0x10010000. Trong chương trình của nhóm sẽ có khai báo trước mảng kiểu word chứa 50 số nguyên cần sắp xếp theo thứ tự từ bé đến lớn (đầu vào của chương trình). Ngoài ra, nhóm còn khai báo một số chuỗi kiểu asciiz để in thông tin trong quá trình chạy, ví dụ như chuỗi: "Bat dau qua trinh sort.\n", "\tPivot Index la: ",...

Mặt khác, vì yêu cầu của đề bài tập lớn là chỉ sử dụng các lệnh thật, nên ta không thể sử dụng các lệnh *load address* để load các dữ liệu này theo tên của chúng, mà ta phải tính giá trị bắt đầu của dữ liệu cần dùng tương ứng và gán giá trị đó vào biến chúng ta muốn lưu địa chỉ vào. Địa chỉ của vùng dữ liệu là một số 32 bit, nên ta không thể dùng các lệnh *add, addi* vì các lệnh này chỉ cho phép làm việc với số 16 bit, thay vào đó ta sẽ dùng lệnh *lui* và lệnh *ori*.

2.2 Phần thân chương trình .text:

2.2.1 Hàm main:

Đây là hàm chính của chương trình và là hàm được thực thi đầu tiên khi chạy chương trình.

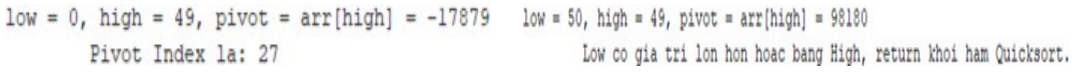
Nhiệm vụ của hàm này là khởi tạo các giá trị ban đầu cho các thanh ghi để truyền vào hàm Quicksort, gồm có địa chỉ đầu tiên của dãy số, vị trí bắt đầu (0) và vị trí kết thúc (49) của dãy. Sau khi hàm Quicksort thực hiện xong công việc và trở về hàm main, hàm main sẽ chạy một vòng lặp để in ra dãy đã được sắp xếp. Cuối cùng, sau khi thực hiện hết các công việc, hàm sẽ dùng lệnh *syscall* để thoát khỏi chương trình.

2.2.2 Hàm Quicksort:

Đây là hàm hiện thực chính của thuật toán Quick Sort. Hàm khởi tạo với 3 giá trị đầu vào là: địa chỉ bắt đầu của mảng 50 số nguyên, vị trí đầu mảng, vị trí kết thúc của mảng cần sắp xếp. Hàm sẽ thực hiện các nhiệm vụ sau:

- In các giá trị *low* (vị trí bắt đầu của mảng đang xét), *high* (vị trí kết thúc của mảng) và giá trị *pivot*.
- Gọi hàm *partition*.
- Sau khi hàm hoàn thành gọi hàm *partition* để đưa pivot về đúng vị trí, hàm sẽ in ra vị trí mới của *pivot*. Tuy nhiên, trong trường hợp hàm *partition* không được gọi do giá trị *low* lớn hơn *high* trong lần chạy đó của hàm Quicksort **(đây là điều kiện thoát của hàm Quicksort)** thì hàm sẽ nhảy xuống một hàm khác là *exitQuicksort* và in ra: "Low co gia tri lon hon hoac bang High, return khoi ham Quicksort.", sau đó sẽ thoát khỏi hàm trở về hàm Quicksort đã gọi đệ quy nó hoặc trở về hàm main.
- Lúc này chương trình sẽ gọi đệ quy 2 lần hàm Quicksort một lần nữa sau khi hàm Partition trả về vị trí đúng cho giá trị pivot, nhưng tham số đầu vào của hàm Quicksort lúc này là: địa chỉ của dãy, vị trí bắt đầu là giá trị low hiện tại và vị trí kết thúc dãy là vị trí của pivot – 1 đối với hàm Quicksort thứ nhất; còn hàm Quicksort thứ hai sẽ có vị trí bắt đầu là pivot + 1, vị trí kết thúc là high.

- Thoát khỏi hàm.



(a) Ví dụ 1 **(b)** Ví dụ 2

Hình 3: *Các ví dụ giá trị in ra bởi hàm Quicksort*

Trong quá trình hiện thực hàm Quicksort cũng như hiện thực các hàm khác, chúng ta phải đảm bảo các yêu cầu trong việc gọi hàm như: cung cấp vùng nhớ stack để lưu các giá trị cần thiết khi gọi các hàm lồng nhau, truyền tham số, bảo toàn giá trị của các thanh ghi,. . .

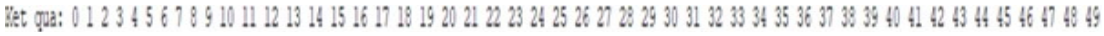
2.2.3 Hàm partition:

Hàm thực hiện nhiệm vụ sắp xếp pivot về đúng vị trí của nó trong trường hợp toàn bộ dãy số đã được sắp xếp theo đúng thứ tự từ bé đến lớn. Hàm này chỉ được gọi bởi hàm Quicksort, nhận vào 3 thông số bao gồm: địa chỉ bắt đầu của dãy 50 số nguyên, địa chỉ bắt đầu và kết thúc của dãy đang xét. Nhiệm vụ và các thông tin cụ thể về hàm *partition* đã được trình bày ở phần 1.2.2, ở đây ta chỉ cần hiện thực lại hàm bằng hợp ngữ MIPS.

2.2.4 Hàm printLoop:

Thực chất đây là một phần của hàm main, chỉ thực hiện 1 lần khi hàm Quicksort thực thi xong nhằm in ra kết quả sắp xếp của hàm Quicksort. Nhưng với chức năng riêng biệt của hàm so với các hàm còn lại của chương trình, ta vẫn gọi và tách ra như một hàm riêng.

Hàm có khả năng in ra một dãy theo thứ tự từ vị trí 0, 1, 2,... đến cuối dãy, với đầu vào là địa chỉ bắt đầu của dãy và kích thước của dãy cần in ra.



Hình 4: *Ví dụ về kết quả in ra của hàm printLoop*

3 KẾT QUẢ THỰC THI CHƯƠNG TRÌNH

3.1 Thống kê số lệnh và loại lệnh (instruction types) của chương trình

Từ cơ sở lý thuyết và cấu trúc chương trình được trình bày ở phần trên, nhóm đã hoàn thành được chương trình sắp xếp dãy số tăng dần với mã bên dưới là hợp ngữ MIPS. Ban đầu ta có dãy gồm 50 số nguyên chưa được sắp xếp:

-63080, -48429, -86772, 6939, -28043, -81060, 90053, -93273, -45102, 46709, -30450, 73112, 98180, 34294, 70178, 1341, 55844, -90138, -52535, 10807, -85199, -36247, 903, 67315, -4793, -30485, 90996, 8895, -39032, 33217, 21644, -88613, -28885, -47593, 43420, -38531, -34263, 62639, -34159, -23066, -46240, -17991, -95391, 50866, -16562, -36135, -60359, 7545, -71583, -17879.

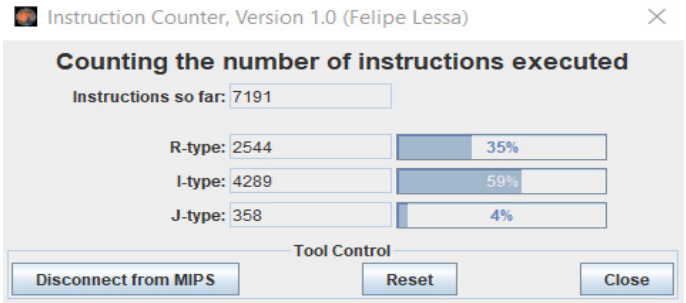
Sau khi sắp xếp ta sẽ nhận được dãy 50 số nguyên tăng dần:

-95391, -93273, -90138, -88613, -86772, -85199, -81060, -71583, -63080, -60359, -52535, -48429, -47593, -46240, -45102, -39032, -38531, -36247, -36135, -34263, -34159, -30485, -30450,

-28885, -28043, -23066, -17991, -17879, -16562, -4793, 903, 1341, 6939, 7545, 8895, 10807, 21644, 33217, 34294, 43420, 46709, 50866, 55844, 62639, 67315, 70178, 73112, 90053, 90996, 98180.

Để thống kê số lệnh thực thi được chính xác nhóm đã thực hiện 13 testcases ngẫu nhiên với 3 testcases cho những trường hợp xấu nhất. Sau đó sử dụng tool của MARS để đếm số lệnh, loại lệnh của mỗi trường hợp, từ đó đưa ra giá trị trung bình số lệnh và loại lệnh của chương trình này.

- Nhóm đã random ra 10 testcases từ trang web https://www.randomlists.com/random-numbers với các dãy 50 số nguyên nằm trong khoảng từ -99999 cho đến 99999 (các số nguyên trong dãy có thể lặp lại), cộng thêm 3 trường hợp xấu nhất được nhóm tự thêm vào, đây là đường link dẫn đến danh sách các testcases cho chương trình của nhóm.
- Dùng tool của MARS để đếm số lệnh, loại lệnh của mỗi trường hợp:



Hình 5: *Ví dụ về việc dùng tool của MARS để đếm số lệnh, loại lệnh của một trường hợp*

- Đường link Video demo code chương trình của nhóm về đề tài bài tập lớn này: https://youtu.be/9IX-Fiv-ylk

Số liệu thống kê về chương trình của nhóm như sau:

Testcase	R-type	I-type	J-type	Tổng số lệnh
1 (Dãy tăng dần)	7570	14700	1373	23643
2 (Dãy không đổi)	7570	14700	1373	23643
3 (Dãy giảm dần)	6320	12200	1373	19893
4	2544	4289	358	7191
5	2500	4255	371	7126
6	2652	4464	358	7474
7	2814	5044	491	8349
8	2630	4514	370	7514
9	2430	4081	337	6848
10	2594	4452	380	7426
11	2638	4382	345	7365
12	2448	4133	353	6934
13	2572	4304	358	7234
Trung bình không tính các worst-case	2582.2	4391.8	372.1	7346.1
Trung bình	3637.077	6578.308	603.0769	10818.46

Nhận xét: Với những trường hợp ngẫu nhiên, số lệnh mang tính ổn định. Với các trường hợp xấu nhất, số lệnh tăng gấp nhiều lần so với giá trị trung bình. Chính vì khi ở trường hợp

xấu nhất, thời gian tiêu tốn sẽ là $O(n^2)$, còn thời gian tiêu tốn trung bình sẽ là $O(n \log(n))$.

Nguyên nhân gây nên sự khác nhau này chính là sự hoán đổi pivot với giá trị tại vị trí i sau mỗi lần thực thi hàm Partition. Tức có nghĩa, đối với thuật toán này thì trường hợp xấu nhất chính là khi dãy tăng dần, giảm dần và không đổi. Còn đối với chương trình của nhóm, thì trường hợp xấu nhất chính là khi dãy tăng dần hoặc không đổi.

3.2 Tính toán thời gian thực thi chương trình

Dữ liệu cho trước:

- Máy tính thực thi chạy ở tần số 3.4 GHz.
- Máy tính thực thi từng câu lệnh theo mô hình Single Cycle Processor:
  - Chỉ số CPI là 1.
  - Thời gian mỗi chu kỳ: Time Per Cycle $= \frac{1}{\text{Frequency}} = \frac{1}{3.4 \times 10^9}$ (s)
  - Thời gian thực thi được tính theo công thức:

    Execute Time = CPI $\times$ Time Per Cycle $\times$ Number Of Instructions (s)

Tính toán các trường hợp cho chương trình của nhóm:

Testcase	Tổng số lệnh	Thời gian thực thi (s)
1 (Dãy tăng dần)	23643	6.954E-06
2 (Dãy không đổi)	23643	6.954E-06
3 (Dãy giảm dần)	19893	5.851E-06
4	7191	2.115E-06
5	7126	2.096E-06
6	7474	2.198E-06
7	8349	2.456E-06
8	7514	2.210E-06
9	6848	2.014E-06
10	7426	2.184E-06
11	7365	2.166E-06
12	6934	2.039E-06
13	7234	2.128E-06
Trung bình không tính các worst-case	7346.1	2.161E-06
Trung bình	10818.46	3.182E-06

4 TÀI LIỆU THAM KHẢO

[1] Giới thiệu về bài toán sắp xếp, truy cập từ: https://www.topcoder.com/thrive/articles/Sorting

[2] Các thuật toán sắp xếp cơ bản, truy cập từ: https://viblo.asia/p/cac-thuat-toan-sap-xep-co-ban-Eb85ooNO52G

[3] Thuật toán sắp xếp Quick Sort, truy cập từ:

- https://www.geeksforgeeks.org/quick-sort
- https://en.wikipedia.org/wiki/Quicksort
- https://nguyenvanhieu.vn/thuat-toan-sap-xep-quick-sort

[4] Phương pháp phân chia Hoare và Lomuto trong thuật toán sắp xếp Quick Sort, truy cập từ: https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort

[5] Số trung vị (median), truy cập từ: https://en.wikipedia.org/wiki/Median