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

KIẾN TRÚC MÁY TÍNH

Team testcase: Bài tập lớn

Sắp xếp chuỗi theo Quick Sort

GVHD: Trần Thanh Bình

SV[1]: Trần Khánh Tùng - 1814710
Nguyễn Hoàng Việt - 1814771
Nguyễn Long Vũ - 1814816

Thành phố Hồ Chí Minh, 15/11/2019

[1]Đóng góp của các thành viên là như nhau, thứ tự sắp xếp này được quyết định bởi người được phân viết phần đầu.

Mục lục

Danh sách hình vẽ

Mở đầu:

Bài báo cáo này trình bày cách hiện thực thuật toán sắp xếp (Sorting Algorithm) Quicksort bằng hợp ngữ MIPS (MIPS Assembly Language) trên môi trường lập trình MARS (MIPS Assembler and Runtime Simulator), là môi trường lập trình MIPS được tạo ra chủ yếu cho việc giáo dục, học tập, tìm hiểu về hợp ngữ [1].

Thuật toán Quicksort sẽ được hiện thực trên một dãy gồm 50 số nguyên dương. Sau khi áp dụng giải thuật, dãy sẽ được sắp xếp lại theo thứ tự tăng dần, từ bé đến lớn.

Báo cáo gồm có 5 phần: Cơ sở lí thuyết, Cấu trúc chương trình, Kết quả chạy, Kế hoạch và phân chia công việc, Tài liệu tham khảo.

1 CƠ SỞ LÍ THUYẾT

1.1 Bài toán sắp xếp

Sắp xếp (sorting) là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định [2]. Chẳng hạn như thứ tự tăng dần (hay 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 Công nghệ thông tin với các mục đích khác nhau: sắp xếp dữ liệu trong máy tính để tìm kiếm cho thuận lợi, sắp xếp các kết quả xử lý để in ra trên bảng biểu,... 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 [3]:

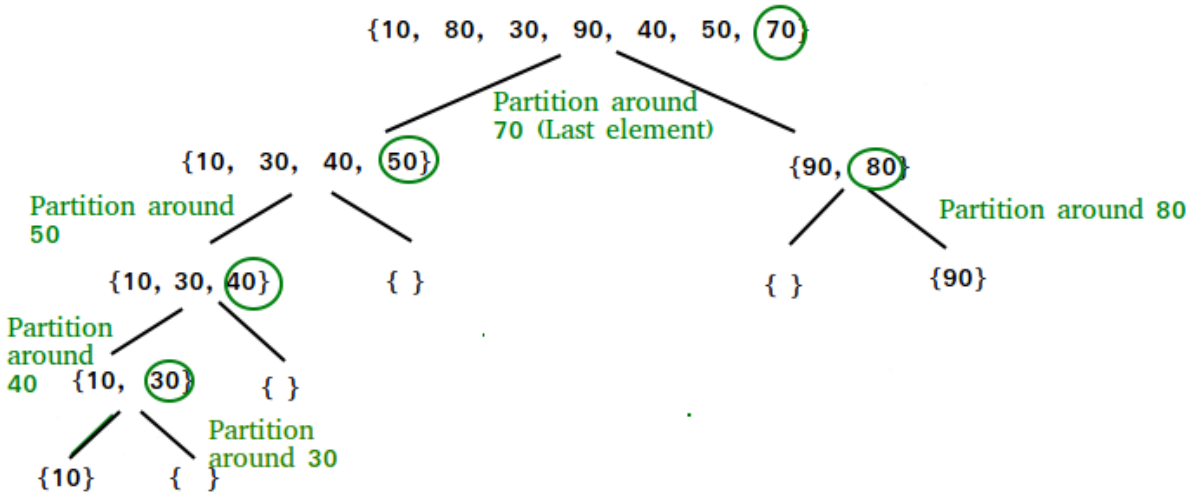
- **Thời gian** chạy. Đối với các dữ liệu rất lớn, các thuật toán không hiệu quả sẽ chạy rất chậm và không thể ứng dụng trong thực tế.
- **Bộ nhớ**. Các thuật toán nhanh đòi hỏi đệ quy sẽ có thể phải tạo ra các bản copy của dữ liệu đang xử lí. Với những hệ thống mà bộ nhớ có giới hạn (ví dụ embedded 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, trở thành state-of-the-art, như: Bubble/Shell sort, Insertion sort, Selection sort, Quick sort, Merge sort.
Trong giới hạn của báo cáo này, nhóm sẽ hiện thực thuật toán Quicksort trên một dãy 50 số nguyên bằng hợp ngữ MIPS.

1.2 Thuật toán Quicksort

1.2.1 Mô tả

Thuật toán Quicksort (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 [4]. Đúng như tên gọi, quicksort là một thuật toán rất hiệu quả (rất "nhanh") và rất thông dụng.



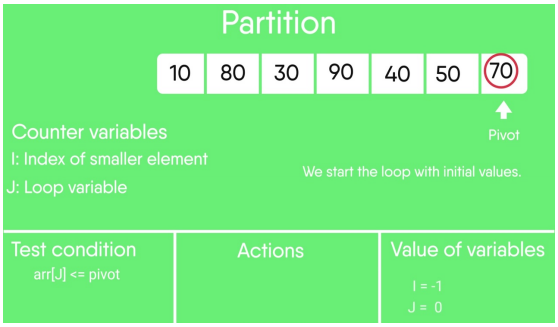
Hình 1: Minh họa cách hoạt động của quicksort [5]

Ý tưởng chủ đạo của quicksort 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 tí 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.

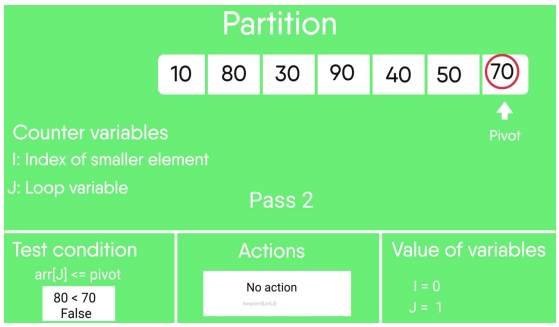
1.2.2 Hàm phân hoạch (phân đoạn) dãy số

Mấu chốt chính của thuật toán quick sort là việc phân đoạn dãy số. Mục tiêu của công việc này là: Cho một dãy và một phần tử x là pivot. Đặt x vào đúng vị trí của dãy đã sắp xếp. Di chuyển tất cả các phần tử của dãy mà nhỏ hơn x sang bên trái vị trí của x, và di chuyển tất cả các phần tử của dãy mà lớn hơn x sang bên phải vị trí của x.
Khi đó ta sẽ có 2 dãy con: dãy bên trái của x và dãy bên phải của x. Tiếp tục công việc với mỗi dãy con (chọn pivot, phân đoạn) cho tới khi dãy được sắp xếp.

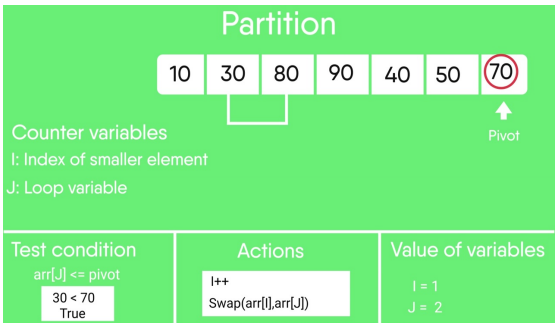
Thuật toán phân đoạn: có nhiều cách để hiện thực thuật toán này. Ở đây nhóm sẽ trình bày cách phân chia Lomuto, chính là cách mà nhóm đã hiện thực thuật toán quicksort bằng MIPS.
Cách phân chia này (Lomuto partition scheme) được phát minh bởi Nico Lomuto và được phổ biến bởi Bentley thông qua cuốn sách Programming Pearls [8] và bởi Cormen cùng những người đồng nghiệp qua quyển Introduction to Algorithms [9]. 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.



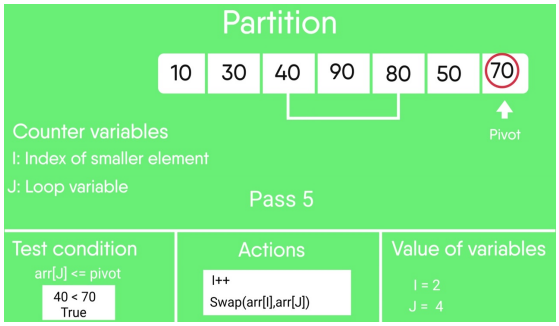
(a) Chọn pivot, khởi tạo giá trị ban đầu cho i và j, đổi chỗ giá trị đầu (nhỏ hơn pivot) với chính nó (lưu ý ở đây ta cộng i lên 1 trước rồi mới tiến hành đổi vị trí)



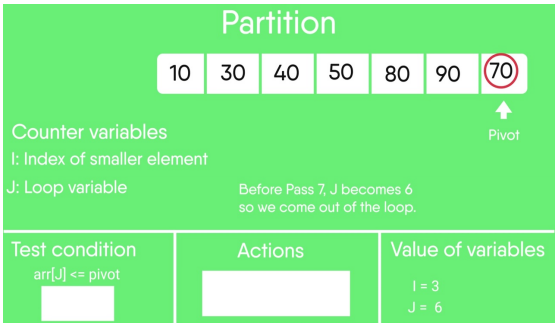
(b) Lần lặp thứ 2, giá trị tại ví trị j lúc này lớn hơn pivot nên ta không làm gì



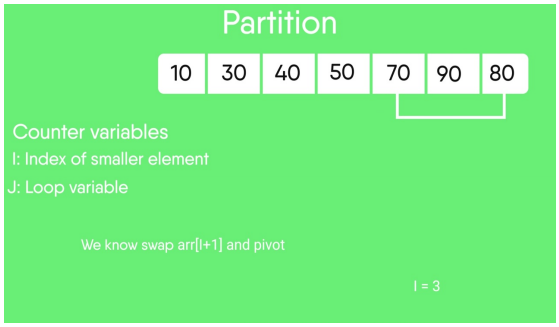
(c) Ở lần lặp này, 30 <= 70 nên ta sẽ cộng i lên 1 rồi đổi chỗ giá trị ở 2 vị trí



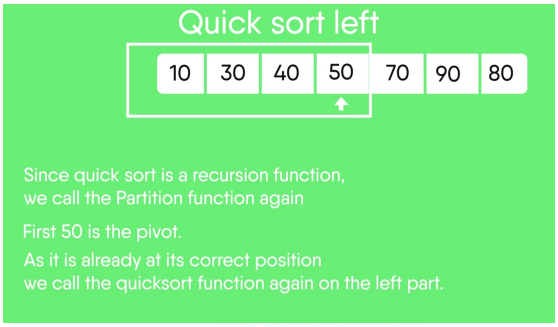
(d) 2 lần lặp tiếp theo không có gì thay đổi, ta bỏ qua đến lần lặp thứ 5, tiến hành đổi chỗ. Lưu ý qua từng lần lặp chỉ có giá trị của j được tăng lên, còn giá trị của i chỉ tăng lên 1 khi ở lần lặp đó có hoán đổi



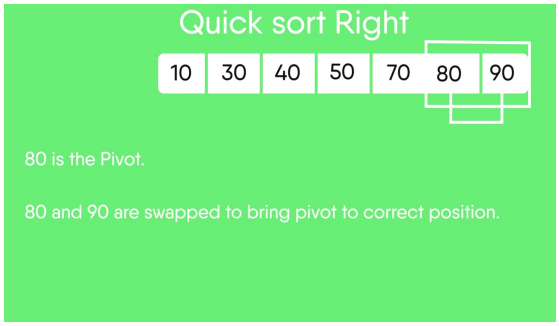
(e) Đến lần lặp cuối, vị trí j là vị trí của pivot, còn i là chỉ cuối cùng trong dãy vừa sắp có giá trị nhỏ hơn pivot



(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 quicksort cho dãy bên trái pivot



(h) Và cho dãy bên phải

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

1.2.3 Chọn điểm pivot

Quicksort là một thuận 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 [6]:

- Luôn chọn điểm đầu tiên.
- Luôn chọn điểm cuối cùng (đây là cách hiện thực của nhóm).
- Chọn một điểm ngẫu nhiên.
- Chọn điểm trung vị [7].

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

$$T(n) = T(k) + T(n-k-1) + o(n)$$

2 số hạng đầu tiên là của 2 lần gọi đệ qui để gọi hàm quicksort 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 giá trị khác. Nếu chúng ta chọn cách chọn điểm pivot luôn là điểm đầu tiên, hoặc 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:

$$T(n) = T(0) + T(n-1) + o(n)$$

Tương đương với:

$$T(n) = T(n-1) + o(n)$$

Tức 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 thức nó, chúng ta cần phải biết tất cả 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 trung vị nào cũng sẽ có ưu, nhược điểm của nó.

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 chi 0x10010000. Trong chương trình của nhóm sẽ có khai báo trước mảng kiểu word chứ 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", "\nPivot Index la: ",...
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 Vùng chứa code .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 quicksort. Hàm nhận vào 3 thông số: địa chỉ bắt đầu của dãy 50 số nguyên, vị trí bắt đầu, vị trí kết thúc của dãy cần sắp xếp
Hàm có các nhiệm vụ sau:

- In các giá trị low (vị trí bắt đầu của chuỗi đang xét), high (vị trí kết thúc) của dãy đang xét, 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 **label** *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.
- Gọi 2 hàm Quicksort con sau khi hàm Partition trả về vị trí đúng cho pivot hiện tại. Hàm Quicksort đầu tiên sẽ cần truyền vào cho nó 3 giá trị là địa chỉ dãy, vị trí bắt đầu sẽ là giá trị low hiện tại, vị trí kết thúc sẽ là vị trí của pivot - 1. Còn hàm high chỉ khác vị trí bắt đầu là pivot+1, vị trí kết thúc là high.
- Thoát khỏi hàm.

```
low = 10, high = 40, pivot = arr[high] = 40
        Pivot Index la: 40
```

(a) Ví dụ 1

```
low = 9, high = 8, pivot = arr[high] = 8
       Low co gia tri lon hon hoac bang High, return khoi ham Quicksort.
```

(b) Ví dụ 2

Hình 3: 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 thuyết 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: địa chỉ bắt đầu của dãy 50 số nguyên, địa chỉ bắt đầu và kết thúc của dãy đang xét.
Bản chất hàm giống như ta đã phân tích ở mục 1.2.2, ta chỉ cần triển khai lại bằng hợp ngữ MIPS.

2.2.4 Hàm printLoop

Thực ra, đây không hẳn là một hàm, vì nó chỉ là một phần trong hàm main, chỉ chạy một lần duy nhất khi hàm Quicksort được gọi bởi hàm main trả về. Nhưng với chức năng riêng biệt của nó, nhóm vẫn tách ra đây thành 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 độ dài của dãy.

```
Ket qua: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Hình 4: Ví dụ về kết quả in ra của hàm printLoop, với đầu vào là một dãy 20 số nguyên theo thứ tự từ 0 đến 19

3 KẾT QUẢ CHẠY

3.1 Thống kê số lệnh, loại lệnh (instruction type) của chương trình

Từ cơ sở lý thuyết ban đầu kết hợp với cấu trúc chương trình chặt chẽ, nhóm đã hoàn thành được một chương trình sắp xếp tăng dần dãy số, với mã bên dưới là hợp ngữ MIPS. Chương trình thực thi có chức năng như sau:

Ban đầu ta có dãy 50 số chưa được sắp xếp:
17767, 9158, -26519, 18547, -9135, 23807, -27574, 22764, -57559, -33587, 22714, -10325, 16882, 7931, -22045, 57670, 124, -40254, -63404, 10232, -56549, -5656, -12825, 17293, -61578, -55974, -1746, 29283, -15821, 55199, 50377, -63590, -1178, -41678, 20493, 55223, 47665, 58456, 12451, 55642, -40667, -30371, -20219, 41751, 43096, 23273, 33886, -22316, 48555, 36018.

Sau khi sắp xếp, ta sẽ nhận được dãy tăng dần như sau:
-63590, -63404, -61578, -57559, -56549, -55974, -41678, -40667, -40254, -33587, -30371, -27574, -26519, -22316, -22045, -20219, -15821, -12825, -10325, -9135, -5656, -1746, -1178, 124, 7931, 9158, 10232, 12451, 16882, 17293, 17767, 18547, 20493, 22714, 22764, 23273, 23807, 29283, 33886, 36018, 41751, 43096, 47665, 48555, 50377, 55199, 55223, 55642, 57670, 58456.

Để thống kê số lệnh thực thi được chính xác, nhóm đã đưa ra 20 testcase ngẫu nhiên (trong đó có cả trường hợp xấu nhất), sau đó cho chương trình chạy và dùng 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.Đây là testcase mà nhóm đã random được (các số nằm trong khoảng từ -2^{16} cho đến $2^{16}-1$, trừ trường hợp đầu tiên là trường hợp xấu nhất được nhóm tự thêm vào, *nhóm sử dụng ngôn ngữ C++ để tạo chương trình tạo ngẫu nhiên các số này, đây là source code của chương trình này*). Số liệu thống kê như sau:

Đối với chương trình có in ra trong quá trình chạy:

Testcase thứ	R-Type	I-Type	J-Type	Tổng số lệnh
1	6320	12200	1373	19893
2	2626	4407	353	7386
3	2668	4566	387	7621
4	2772	4800	413	7985
5	2692	4613	386	7691
6	2736	4711	396	7843
7	2552	4330	383	7265
8	2774	4636	368	7778
9	2584	4522	388	7494
10	2782	4947	458	8187
11	2698	4577	379	7654
12	2536	4263	348	7147
13	2580	4343	381	7304
14	2900	4993	432	8325
15	2642	4411	366	7419
16	2470	4173	349	6992
17	2444	4124	352	6920
18	2686	4555	381	7622
19	2446	4180	363	6989
20	2522	4190	344	7056
TB khi không có TH 1	2637.37	4491.63	380.37	7509.37
Trung bình	2821.5	4877.05	430	8128.55

Đối với chương trình không in ra trong quá trình chạy:

Testcase thứ	R-Type	I-Type	J-Type	Tổng số lệnh
1	4388	9565	1324	15277
2	1249	2432	304	3985
3	1328	2635	338	4301
4	1432	2869	364	4665
5	1352	2682	337	4371
6	1396	2780	347	4523
7	1212	2399	334	3945
8	1323	2573	319	4215
9	1355	2723	339	4417
10	1516	3104	409	5029
11	1321	2602	330	4253
12	1196	2332	299	3827
13	1203	2368	332	3903
14	1486	2974	383	4843
15	1228	2392	317	3937
16	1167	2286	300	3753
17	1141	2237	303	3681
18	1309	2580	332	4221
19	1180	2337	314	3831
20	1145	2215	295	3655
TB khi không có TH 1	1291.53	2553.68	331.37	4176.58
Trung bình	1446.35	2904.25	381	4731.6

Nhận xét: Với những trường hợp ngẫu nhiên, số lệnh mang tính ổn định. Với 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 $o(nlog_2n)$. Trường hợp xấu nhất này nhóm chọn là khi dãy số có thứ tự sắp xếp giảm dần. Nhưng với 2 trường hợp xấu nhất khác, số liệu lại khác so với trường hợp xấu nhất trên, chẳng hạn với dãy có thứ tự sắp xếp tăng dần và dãy có các số hạng bằng nhau:

Trường hợp	R-Type	I-Type	J-Type	Tổng số lệnh
Dãy tăng dần	7570	14700	1373	23643
Dãy không đổi	7570	14700	1373	23643
Dãy giảm dần	6320	12200	1373	19893

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 giải thuật 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 3.4GHz
- Máy tính thực thi từng câu lệnh theo mô hình Single Cycle Processor.
  - Chỉ số CPI là 1.
  - Thời gian mỗi chu kì (TimesPerCycle) được tính theo công thức:

$$TimesPerCycle = \frac{1}{Frequency} = \frac{1}{3.4 \times 10^9}$$

– Thời gian thực thi được tính theo công thức:

$$ExecuteTimes = CPI \times TimesPerCycle \times NumberOfInstructions$$

Tính toán cho các trường hợp:

a) Trường hợp chương trình in ra quá trình chạy

Testcase thứ	Tổng số lệnh	Thời gian thực thi(s)
1	19893	5.85E-06
2	7386	2.17E-06
3	7621	2.24E-06
4	7985	2.35E-06
5	7691	2.26E-06
6	7843	2.31E-06
7	7265	2.14E-06
8	7778	2.29E-06
9	7494	2.20E-06
10	8187	2.41E-06
11	7654	2.25E-06
12	7147	2.10E-06
13	7304	2.15E-06
14	8325	2.45E-06
15	7419	2.18E-06
16	6992	2.06E-06
17	6920	2.04E-06
18	7622	2.24E-06
19	6989	2.06E-06
20	7056	2.08E-06
TB không có TH 1	7509.368421	2.21E-06
Trung bình	8128.55	2.38E-06

b) Trường hợp không in ra quá trình chạy

Testcase thứ	Tổng số lệnh	Thời gian thực thi(s)
1	15277	4.49E-06
2	3985	1.17E-06
3	4301	1.27E-06
4	4665	1.37E-06
5	4371	1.29E-06
6	4523	1.33E-06
7	3945	1.16E-06
8	4215	1.24E-06
9	4417	1.30E-06
10	5029	1.48E-06
11	4253	1.25E-06
12	3827	1.13E-06
13	3903	1.15E-06
14	4843	1.42E-06
15	3937	1.16E-06
16	3753	1.10E-06
17	3681	1.08E-06
18	4221	1.24E-06
19	3831	1.13E-06
20	3655	1.08E-06
TB không có TH 1	4176.578947	1.23E-06
Trung bình	4731.6	1.39E-06

4 KẾ HOẠCH VÀ PHÂN CHIA CÔNG VIỆC

Kế hoạch:
Nhóm bắt đầu thực hiện đề tài vào tuần 44.

- Tuần 44: Bắt đầu tìm hiểu về thuật toán Quicksort, lên ý tưởng về cách hiện thực bằng hợp ngữ MIPS.
- Tuần 45: Phân chia công việc, viết code, kiểm tra sửa lỗi cho code.
- Tuần 46: Bắt đầu viết báo cáo, hoàn chỉnh code.
- Tuần 47: Hoàn thành báo cáo, chốt lại hết các vấn đề còn dang dở của code, báo cáo.
- Tuần 48: Báo cáo về đề tài của nhóm.

Phân chia công việc:
Các thành viên trong nhóm đều có đóng góp như nhau. Cách phân chia công việc sau chỉ mang tính tương đối, do kế hoạch công việc cần có sự thứ tự trước và sau (ví dụ: sau khi hoàn thành code chương trình thì mới có thể test thử, hoàn thành việc kiểm tra code mới bắt đầu viết báo cáo,...) nên đây chỉ là cách phân chia ban đầu, thực tế các thành viên luôn phải hỗ trợ nhau để hoàn thành công việc theo kịp kế hoạch cũng như để các công việc của các thành viên được rải đều. Nên sự đóng góp của các thành viên là như nhau.

Trần Khánh Tùng - 1814710:	Viết code, viết báo cáo.
Nguyễn Hoàng Việt - 1814771:	Viết hàm random ra các dãy số, kiểm tra, sửa lỗi code.
Nguyễn Long Vũ - 1814816:	Tính toán số lệnh, thời gian chạy, viết báo cáo.

Trong quá trình thực hiện nhóm đã sử dụng một số công cụ để làm việc hiệu quả hơn:
- **Overleaf:** Là một trang web giúp soạn thảo văn bản LaTeX. Việc soạn thảo không chỉ đến từ một người mà cả một nhóm có thể thực hiện chung với nhau thông qua tính năng đa người dùng (multi user).
- **Github:** Là công cụ lưu trữ và chia sẻ mã nguồn miễn phí. Người dùng là sinh viên của trường sẽ được tự động nâng cấp lên thành viên Pro của Github mà không phải tốn phí.
- **Google Photos:** Google Photos là một công cụ lưu trữ hình ảnh của Google. Google Photos cung cấp một bộ nhớ lưu trữ không giới hạn miễn phí khi chọn lựa thiết lập "high quality". Ngoài ra Google Photos còn sử dụng tính năng nhận diện cao cấp cùng với kho lưu trữ dữ liệu không lồ của Google nó có thể nhận ra chủ đề cho bức ảnh một cách dễ dàng. Google Photos còn tự động sắp xếp hình ảnh được gởi lên theo vị trí và thời gian, từ đó giúp thuận tiện hơn trong việc tìm kiếm hình ảnh.
- **Facebook Messenger:** Là một ứng dụng nhắn tin cho phép thành lập các nhóm nhiều người dùng cùng nhắn tin, tương tác với nhau. Đây là công cụ chính để nhóm trao đổi ý tưởng, phân chia nhiệm vụ, làm việc với nhau.

5 TÀI LIỆU THAM KHẢO

Tài liệu

[1] https://courses.missouristate.edu/KenVollmar/MARS/

[2] Lê Minh Hoàng, *Giải thuật & lập trình*.

[3] Sorting Introduction: https://www.topcoder.com/community/competitive-programming/tutorials/sorting/.

[4] https://vi.wikipedia.org/wiki/Charles_Antony_Richard_Hoare

[5] Quicksort: https://www.geeksforgeeks.org/quick-sort/

[6] Quicksort: https://www.geeksforgeeks.org/quick-sort/

[7] Median: https://en.wikipedia.org/wiki/Median

[8] Jon Bentley (1999). *Programming Pearls*. Addison-Wesley Professional.

[9] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. "Quicksort". *Introduction to Algorithms (3rd ed.)*. MIT Press and McGraw-Hill.

Ngoài ra, nhóm còn tham khảo từ một số trang web, nguồn tài liệu:

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