

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 (CO2007)

---

Bài tập lớn (Học kỳ 221)

# Merge sort số nguyên

---

GV hướng dẫn: Nguyễn Xuân Minh  
Trần Thanh Bình

Nhóm sinh viên: Nguyễn Duy Tùng 2115232  
Nguyễn Thái Tân 2112256

Thành phố Hồ Chí Minh, 12/2022



## Mục lục

<b>1</b>	<b>Giải thuật Merge sort</b>	<b>2</b>
1.1	Giới thiệu . . . . .	2
1.2	Thuật toán . . . . .	2
1.2.1	Thuật toán chung . . . . .	2
1.2.2	Thuật toán Gộp hai nửa mảng đã sắp xếp . . . . .	2
1.3	Minh họa thuật toán . . . . .	2
1.4	Đánh giá . . . . .	3
<b>2</b>	<b>Hiện thực bằng MIPS</b>	<b>4</b>
2.1	File .BIN . . . . .	4
2.2	Hiện thực . . . . .	4
2.3	Thống kê các lệnh . . . . .	5
2.4	Thời gian chạy . . . . .	6
2.5	Kết quả . . . . .	7
	<b>Tài liệu tham khảo</b>	<b>8</b>

# 1 Giải thuật Merge sort

## 1.1 Giới thiệu

Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp.

## 1.2 Thuật toán

### 1.2.1 Thuật toán chung

Ta có các bước đệ quy sau để thực hiện hàm *mergeSort*

Bước 1 Nếu  $left > right$ , kết thúc thuật toán

Bước 2 Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa:  $mid = (left + right)/2$

Bước 3 Gọi đệ quy hàm *mergeSort* cho nửa đầu tiên: *mergeSort*(*arr*, *left*, *mid*)

Bước 4 Gọi đệ quy hàm *mergeSort* cho nửa thứ hai: *mergeSort*(*arr*, *mid* + 1, *right*)

Bước 5 Gộp 2 nửa mảng đã sắp xếp ở (2) và (3): *mergeTwoSortedArray*(*arr*, *left*, *mid*, *right*)

Bước 6 Kết thúc thuật toán

### 1.2.2 Thuật toán Gộp hai nửa mảng đã sắp xếp

Ở thuật toán trên, ta có đề cập đến hàm *mergeTwoSortedArray* để gộp hai nửa mảng đã được sắp xếp với nhau. Sau đây, ta nêu chi tiết hơn về thuật toán này.

Bước 1 Tạo hai mảng phụ (*subArray1* và *subArray2*) và mảng kết quả *resultArray*.

Bước 2 Sao chép phần tử của nửa mảng đầu vào *subArray1*, nửa mảng sau vào *subArray2*.

Bước 3 Khởi tạo biến  $indexOfSubArray1 = 0$ ,  $indexOfSubArray2 = 0$  và  $indexOfResultArray = 0$

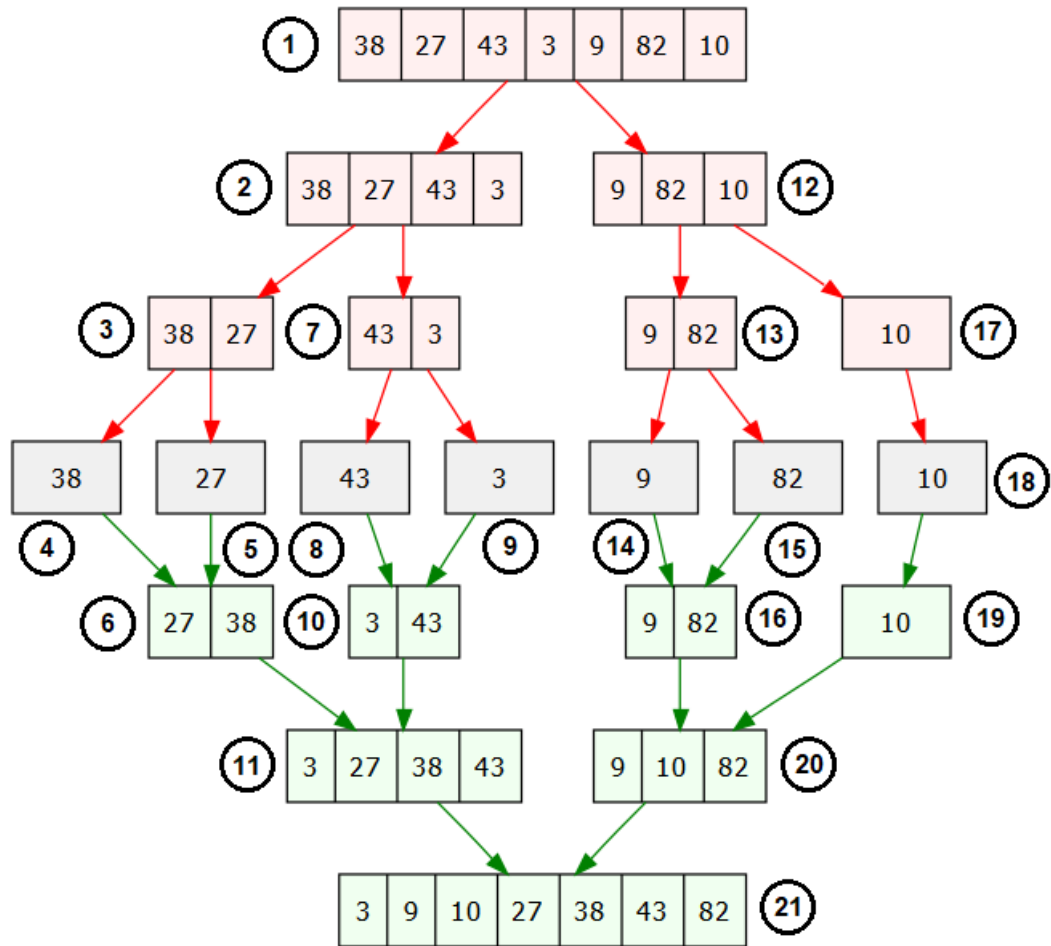
Bước 4 Gộp hai mảng phụ trên bằng cách duyệt một vòng lặp so sánh *subArray1*[*indexOfSubArray1*] và *subArray2*[*indexOfSubArray2*], sau đó sao chép phần tử nhỏ hơn vào *resultArray*, tăng biến đếm của index tương ứng, nếu index không còn thỏa mãn thì thoát khỏi vòng lặp.

Bước 5 Nếu còn lại phần tử nào chưa được duyệt ở mảng phụ *subArray1* thì sao chép vào *resultArray*. Tương tự với *subArray2*.

Bước 6 Kết thúc. Lưu ý xóa vùng nhớ nếu đã có cấp phát.

## 1.3 Minh họa thuật toán

Với thuật toán đệ quy như trên, thì các bước thực hiện thuật toán sẽ tương tự như sau. Tức là mảng sẽ sắp xếp xong một phần, mới bắt đầu sắp xếp phần khác, chứ không phải sắp xếp hai phần một lượt.



Hình 1: Các bước thực hiện thuật toán trên thực tế

## 1.4 Đánh giá

Độ phức tạp thuật toán:

- Trường hợp tốt:  $O(n \log(n))$
- Trung bình:  $O(n \log(n))$
- Trường hợp xấu:  $O(n \log(n))$

Không gian bộ nhớ sử dụng:  $O(n)$

Thuật toán có tính ổn định.

## 2 Hiện thực bằng MIPS

*Bài toán:* Viết chương trình sắp xếp dãy số nguyên có 15 phần tử nhập từ bàn phím dùng giải thuật Merge sort. Yêu cầu xuất dãy ra màn hình mỗi bước. Dữ liệu đầu vào đọc từ file lưu trữ dạng nhị phân trên đĩa INT15.BIN (15 phần tử  $\times$  4 bytes = 60 bytes)

### 2.1 File .BIN

File .BIN là một tập tin mà lưu trữ dữ liệu trong một định dạng nhị phân. File này khác với một tập tin dựa trên văn bản, có thể được chỉnh sửa trong một trình soạn thảo văn bản. File .BIN có thể được tạo ra bởi một loạt các chương trình khác nhau nhưng thông thường không thể chỉnh sửa bằng tay.

Do đó để giải quyết bài toán này, nhóm đã thực hiện viết một chương trình bằng *Mars MIPS* để tạo một file .BIN theo yêu cầu. Sau khi chạy chương trình xong, ta đã có một file chứa 15 dữ liệu, có tên là **INT15.BIN**. *Chương trình nhóm viết dùng để tạo file* : <https://rgl3s.com/4z78gw>

Các lệnh syscall liên quan đến file:

Chức năng	\$v0	Tham số	Kết quả
Mở file	13	\$a0 = địa chỉ của chuỗi chứa tên tệp \$a1 = flags \$a2 = mode	\$v0 chứa bộ mô tả tệp (giá trị âm nếu có lỗi).
Đọc file	14	\$a0 = bộ mô tả tệp \$a1 = bộ đệm chứa dữ liệu của toàn bộ file \$a2 = số ký tự để đọc	\$v0 chứa số ký tự đã đọc (giá trị 0 nếu là cuối tệp, giá trị âm nếu lỗi).
Ghi file	15	\$a0 = bộ mô tả tệp \$a1 = bộ đệm chứa dữ liệu của file đầu ra \$a2 = số ký tự để ghi	\$v0 chứa số ký tự đã ghi (giá trị 0 nếu là cuối tệp, giá trị âm nếu lỗi).
Đóng file	16	\$a0 = bộ mô tả tệp	

Bảng 1: Các lệnh syscall liên quan đến file

### 2.2 Hiện thực

Trước hết, ta khởi tạo trước trường *.data* như sau:

Các biến	Công dụng
fileName	Chuỗi chứa địa chỉ của file
input_str, output_str, readFile_str	Chứa các chuỗi để xuất
comma, space, endl	Các dấu phẩy, khoảng trắng, xuống dòng
openBracket, closeBracket	Các dấu mở ngoặc, đóng ngoặc
arr	Chứa địa chỉ của mảng chính
aux_arr	Chứa địa chỉ của mảng hỗ trợ để in sau các bước

Bảng 2: Trường *.data*

Vì bài toán là sắp xếp mảng có 15 phần tử, nên ta khai báo cố định *arr* là mảng chứa 60 bytes.

Với chương trình này, ở *main* chúng ta có 3 bước chính:

Bước 1 Đọc file `INT15.BIN` đã tạo từ trước đó. Lưu ý khi đọc file, chúng ta phải viết đúng đường dẫn đến file đó ở `fileName` trên trường `.data`.

- Mở file bằng lệnh syscall `13`
- Đọc file bằng lệnh syscall `14`
- Xuất ra dữ liệu ban đầu
- Đóng file bằng lệnh syscall `16`

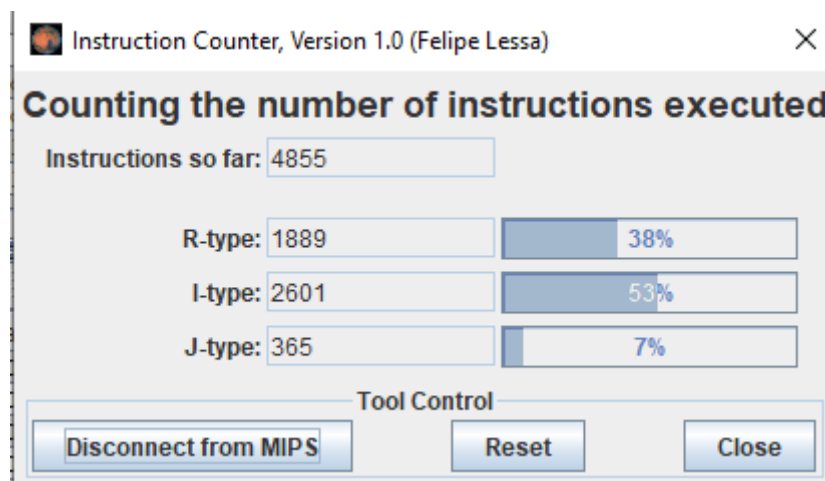
Bước 2 Sắp xếp theo *Merge sort*. Chúng ta cũng vừa sắp xếp, vừa in ra các bước thực hiện.

- Hàm `mergeSort` là hàm không lá, vì nó phải gọi hàm `mergeTwoSortedArray`.
- Hàm `mergeTwoSortedArray` là hàm lá, vì nó không gọi hàm nào khác.

Bước 3 Xuất ra màn hình kết quả cuối cùng.

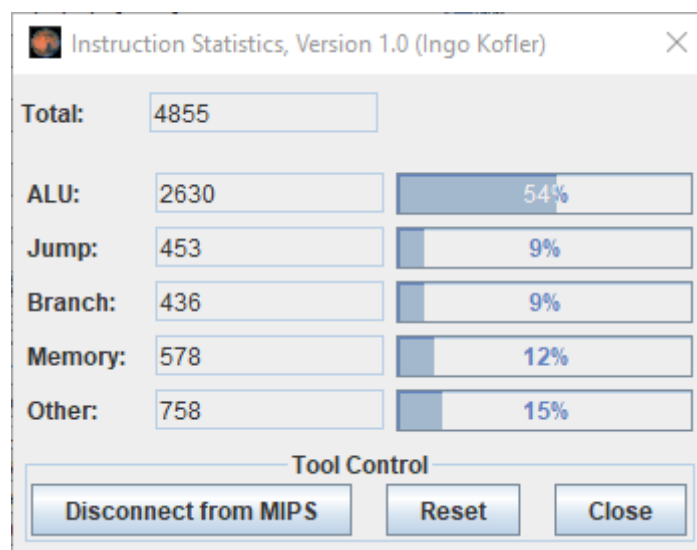
## 2.3 Thống kê các lệnh

MARS MIPS đã cung cấp cho chúng ta sẵn công cụ để thống kê số lượng lệnh. Ta chọn *Tools* -> *Instruction Counter* -> *Connect to MIPS*, sau đó chạy chương trình, ta liền được kết quả thống kê số lượng lệnh theo kiểu *R-type*, *I-type* và *J-type*.



Hình 2: Thống kê các lệnh

Một kiểu thống kê khác, ta chọn *Tools* -> *Instruction Statistics* -> *Connect to MIPS*.



Hình 3: Thống kê các lệnh

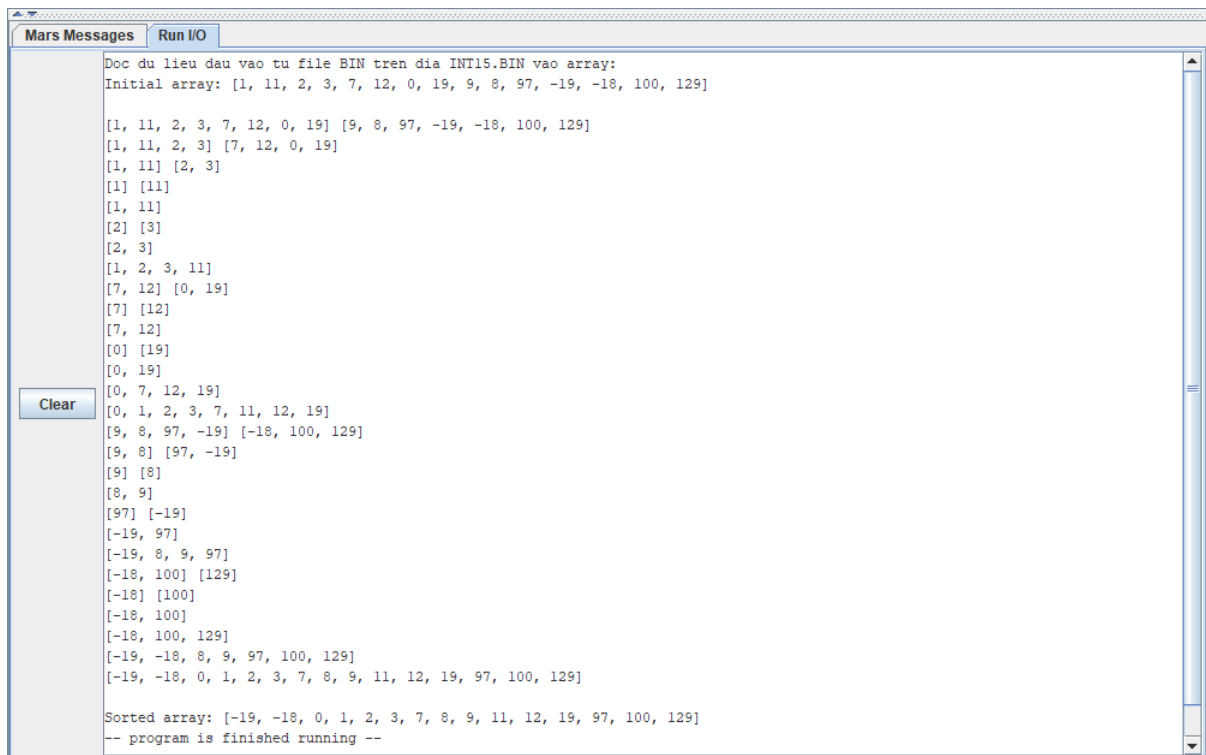
## 2.4 Thời gian chạy

Giả sử Clock Rate = 1 GHz. Chúng ta sẽ tính thời gian chạy chương trình dựa trên số lượng lệnh vừa thống kê ở trên. Khi  $CPI = 1$ , vậy ta sẽ tính được thời gian chạy như sau

$$\text{CPU time} = \text{Instruction Count} \times \text{CPI} \times \frac{1}{\text{Clock Rate}}$$

$$\text{CPU time} = 4855 \times 1 \times \frac{1}{1 \times 10^9} = 4855 \times 10^{-9} (s) = 4855 (ns)$$

## 2.5 Kết quả



```
Doc du lieu dau vao tu file BIN tren dia INT15.BIN vao array:
Initial array: [1, 11, 2, 3, 7, 12, 0, 19, 9, 8, 97, -19, -18, 100, 129]

[1, 11, 2, 3, 7, 12, 0, 19] [9, 8, 97, -19, -18, 100, 129]
[1, 11, 2, 3] [7, 12, 0, 19]
[1, 11] [2, 3]
[1] [11]
[1, 11]
[2] [3]
[2, 3]
[1, 2, 3, 11]
[7, 12] [0, 19]
[7] [12]
[7, 12]
[0] [19]
[0, 19]
[0, 7, 12, 19]
[0, 1, 2, 3, 7, 11, 12, 19]
[9, 8, 97, -19] [-18, 100, 129]
[9, 8] [97, -19]
[9] [8]
[8, 9]
[97] [-19]
[-19, 97]
[-19, 8, 9, 97]
[-18, 100] [129]
[-18] [100]
[-18, 100]
[-18, 100, 129]
[-19, -18, 8, 9, 97, 100, 129]
[-19, -18, 0, 1, 2, 3, 7, 8, 9, 11, 12, 19, 97, 100, 129]

Sorted array: [-19, -18, 0, 1, 2, 3, 7, 8, 9, 11, 12, 19, 97, 100, 129]
-- program is finished running --
```

Hình 4: Kết quả chạy chương trình

Nhóm đã in ra dãy số sau mỗi bước thực hiện. Ta thấy rằng các bước thực hiện này giống như với hình ảnh minh họa ở phần trên.





## Tài liệu tham khảo

- [1] David A Patterson và John L.Hennessy. *Computer Organization and Design: The Hardware/Software Interface, Fifth Edition*. Morgan Kaufmann Publishers, 2017.
- [2] Phạm Quốc Cường. *Kiến trúc máy tính*. Nhà xuất bản Đại học Quốc gia TP HCM.
- [3] Clifford A. Shaffer. *Data Structures and Algorithm Analysis*. Dover Publications, 2013.