TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HỒ CHÍ MINH

Khoa Khoa học và Kĩ thuật Máy tính

BÁO CÁO

BÀI TẬP LỚN KIẾN TRÚC MÁY TÍNH

Đề tài 6:

Thiết kế chương trình merge sort bằng hợp ngữ assembly MIPS

Danh sách nhóm:

STT	Họ và tên	MSSV	Lớp
1	Đào Xuân Đạt	1911000	L09

I. Tổng quan về hợp ngữ MIPS:

- **MIPS** viết tắt của *Microprocessor without Interlocked Pipeline Stages*, là kiến trúc bộ tập lệnh RISC phát triển bởi MIPS Technologies. Ban đầu kiến trúc MIPS là 32bit, và sau đó là phiên bản 64 bit. Nhiều sửa đổi của MIPS, bao gồm MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32 và MIPS64. Phiên bản hiện tại là MIPS32 và MIPS64.

- Hợp ngữ MIPS là ngôn ngữ có khả năng chuyển đổi 1-1 sang ngôn ngữ máy dành cho các dòng máy sử dụng kiến trúc MIPS.

- Hợp ngữ MIPS có 3 định dạng lệnh: R-Type, I-Type và J-Type

- Các nhóm lệnh trong tập lệnh:

+ Các lệnh số học nguyên (cộng, trừ, or, nor, xor, shift left, shift right)

+ Các lệnh truy xuất dữ liệu từ bộ nhớ (load, store, hỗ trợ dữ liệu như byte, half word, word,…)

+ Nhảy và rẽ nhánh (Jump and Branch)

+ Các lệnh học số thực

+ Các lệnh phụ

II. Khái quát về Merge Sort:

- Trong khoa học máy tính, sắp xếp trộn (merge sort) là một thuật toán sắp xếp để sắp xếp các danh sách (hoặc bất kỳ cấu trúc dữ liệu nào có thể truy cập tuần tự, v.d. luồng tập tin) theo một trật tự nào đó. Nó được xếp vào thể loại sắp xếp so sánh. Thuật toán này là một ví dụ tương đối điển hình của lối thuật toán chia để trị (divide and conquer) do John von Neumann đưa ra lần đầu năm 1945. Một thuật toán chi tiết được Goldstine và Neumann đưa ra năm 1948.

- Merge sort có nhiều loại và biến thể như in-place merge, natural merge,… với các độ phức tạp khác nhau. Trong bài báo cáo này chúng ta sẽ xây dựng standard merge sort có time complexity trung bình là O(nlogn).

- Thuật toán:

+ Chia array các phần tử thành 2 phần [a1,….,idx], [idx,….an] với idx là phần tử thứ arraySize / 2 xong rồi tiến hành trộn chúng lại thành 1 dãy đã được sắp xếp. Ta tiến hành đệ quy chia rồi sắp xếp rồi trộn lại cho từng phần cho tới khi phần cần được sắp xếp chỉ còn 1 phần tử.

+ Giả sử có hai danh sách đã được sắp xếp a[1..m] và b[1..n]. Ta sẽ trộn chúng lại thành 1 danh sách mới c[1…m+n] theo cách sắp xếp sau :

So sánh hai phần tử đứng đầu của hai danh sách, lấy phần tử nhỏ hơn cho vào danh sách mới. Tiếp tục như vậy cho tới khi một trong hai danh sách là rỗng.

Khi một trong hai danh sách là rỗng ta lấy phần còn lại của danh sách kia cho vào cuối danh sách mới.

- Pseudo code:

MergeSort(arr[], l, r)

If r > l

1. Find the middle point to divide the array into two halves:

   middle m = (l+r)/2

2. Call mergeSort for first half:

   Call mergeSort(arr, l, m)

3. Call mergeSort for second half:

   Call mergeSort(arr, m+1, r)

4. Merge the two halves sorted in step 2 and 3:

   Call merge(arr, l, m, r)

III. Tiến hành hiện thực merge sort bằng hợp ngữ MIPS:

- Các loại lệnh sẽ được sử dụng:

ALU: add, addi, sub, srl, sll

Memory: lw, sw, la

Jump: j, jr, jal

Branch: beq, ble, bge, bgt

- Hiện thực:

1. Đầu tiên ta khởi tạo ô trống cho mảng gồm 20 phần tử:

```
.data
    myarray:.space 80                   # create space for aray with 20 word elements
    st:.asciiz "Enter the 20 Elements:\n"
    spa: .asciiz " "                            # space char
    eol: .asciiz "\n"                           # end of line
```

2. Sau đó, ta nhập giá trị cho từng phần tử cho mảng:

```
.text
    addi $v0,$0,4                               # promp array input
    la $a0,st
    syscall
    addi $t0, $0, 0                             # set counter/index to 0
    addi $t1, $0, 80                            # Get array size
    jal inputArray                              # input array
```

- Ta prompt “Enter the 20 elements: “ cho user rồi khởi tạo biến đếm rồi thực hiện hàm inputArray.

- Hàm inputArray sẽ cho user nhập từng phần tử vào mảng.

- Đây là hiện thực của hàm inputArray:

```
inputArray:
    beq $t0, $t1, exit                                  # check loop condition
    addi $v0, $0, 5                             # input from user
    syscall

    sw $v0, myarray($t0)                        # store user's value to array
    add $t0,$t0,4                               # incement counter/index
    j inputArray                                # loop the function

exit:
```

3. Tiến hành merge sort:

```
        la      $a0, myarray            # Load the start address of the array
        add     $t0, $0, $t1            # Load the array length
        add     $a1, $a0, $t0           # Calculate the array end address
        jal     mergesort               # Call the merge sort function
        add     $v0, $0, 10
        syscall                         # We are finished sorting
```

- Hiện thực hàm merge sort:

```
# Recrusive mergesort function
#
# @param $a0 first address of the array
# @param $a1 last address of the array
##
mergesort:

        addi    $sp, $sp, -16           # Adjust stack pointer
        sw      $ra, 0($sp)             # Store the return address on the stack
        sw      $a0, 4($sp)             # Store the array start address on the stack
        sw      $a1, 8($sp)             # Store the array end address on the stack
        sub     $t0, $a1, $a0           # Calculate the difference between the start and end address
                                        # (i.e. number of elements * 4)
        ble     $t0, 4, mergedone       # If the array only contains a single element, just return
        srl     $t0, $t0, 3             # Divide the array size by 8 to half the number of elements
                                        # (shift right 3 bits)
        sll     $t0, $t0, 2             # Multiple that number by 4 to get half of the array size
                                        # (shift left 2 bits)
        add     $a1, $a0, $t0           # Calculate the midpoint address of the array
        sw      $a1, 12($sp)            # Store the array midpoint address on the stack
```

```
        jal     mergesort               # Call recursively on the first half of the array

        lw      $a0, 12($sp)            # Load the midpoint address of the array from the stack
        lw      $a1, 8($sp)             # Load the end address of the array from the stack

        jal     mergesort               # Call recursively on the second half of the array

        lw      $a0, 4($sp)             # Load the array start address from the stack
        lw      $a1, 12($sp)            # Load the array midpoint address from the stack
        lw      $a2, 8($sp)             # Load the array end address from the stack

        jal     merge                   # Merge the two array halves

mergesortend:

        lw      $ra, 0($sp)             # Load the return address from the stack
        addi    $sp, $sp, 16            # Adjust the stack pointer
        jr      $ra                     # Return
```

- Khi đã sắp xếp được 2 nửa của array rồi thì tiến hành trộn chúng lại với nhau bằng hàm merge:

+ Đầu tiên, ta cần lưu lại các address của vị trí đầu, cuối, và giữa của array, sau đó khởi tạo 2 array phụ $s0, $s1:

```
##
# Merge two sorted, adjacent arrays into one using 2 subarrray
#
# @param $a0 First address of first array
# @param $a1 First address of second array
# @param $a2 Last address of second array
##
merge:
        addi    $sp, $sp, -16           # Adjust the stack pointer
        sw      $ra, 0($sp)             # Store the return address on the stack
        sw      $a0, 4($sp)             # Store the start address on the stack
        sw      $a1, 8($sp)             # Store the midpoint address on the stack
        sw      $a2, 12($sp)            # Store the end address on the stack
        sub     $t6, $a1, $a0           # Calculate first half size
        sub     $t7, $a2, $a1           # Calculate second half size

        addi    $v0, $0, 9
        add     $a0, $0, $t6            # Create array[$t6]
        syscall
        add     $s0, $0, $v0            # Move address of newly created array to $s0

        addi    $v0, $0, 9
        add     $a0, $0, $t7            # Create array[$t7]
        syscall
        add     $s1, $0, $v0            # Move address of newly created array to $s1
        lw      $a0, 4($sp)             # Reload start address
```

+ Sau đó, ta thực hiện copy giá trị của 2 nửa array vào 2 array mới tạo:

```
        addi    $t0, $0, 0              # Create index
        la      $s2, ($s0)              # Create working copy of array $s0
        la      $s3, ($s1)              # Create working copy of array $s1
copyData1:                              # Copy data from first half to array $s0
        bge     $t0, $t6, copy1done     # Check loop condition
        lw      $t1, 0($a0)             # Assign element at index $t0 to element in array $s0
        sw      $t1, 0($s2)
        addi    $a0, $a0, 4             # Increase index and both of array's address
        addi    $s2, $s2, 4
        addi    $t0, $t0, 4
        j       copyData1               # Loop again
copy1done:
        lw      $a0, 4($sp)             # Reload the start address of the original array
        addi    $t0, $0, 0              # Reset index to 0
copyData2:                              # Copy data from second half to array $s1
        bge     $t0, $t7, copy2done     # Check loop condition
        lw      $t1, 0($a1)             # Assign element at index $t0 to element in array $s1
        sw      $t1, 0($s3)
        addi    $a1, $a1, 4             # Increase index and both of array's address
        addi    $s3, $s3, 4
        addi    $t0, $t0, 4
        j       copyData2               # Loop again
copy2done:
```

- Sau đó, hàm merge sẽ so sánh hai phần tử đứng đầu của 2 subarray, lấy phần tử nhỏ hơn cho vào array. Tiếp tục như vậy cho tới khi một trong hai danh sách là rỗng.

```
copy2done:
        addi    $t0, $0, 0              # Create index for array $s0
        addi    $t1, $0, 0              # Create index for array $s1
        la      $s2, ($s0)              # Reset to begining of array $s0
        la      $s3, ($s1)              # Reset to begining of array $s1
        lw      $a0, 4($sp)             # Reload the start address of the original array
mergeloop:                              # Merging 2 array $s0, $s1 to original array
        bge     $t0, $t6, remain1       # Loop condition: $t0 < $t6($s0 size) && $t1 < $t7($s1 size)
        bge     $t1, $t7, remain1
        lw      $t2, 0($s2)             # Load value of current element in array $s0 to $t2
        lw      $t3, 0($s3)             # Load value of current element in array $s1 to $t3
        bgt     $t2, $t3, else          # Compare 2 value of 2 subarray, the smaller one will be add to
                                        # original array and continue with the next one in the same subarray
        sw      $t2, 0($a0)             # Store value of current element in array $s0 to the
                                        # current position of original array
        addi    $t0, $t0, 4             # Increment index $t0
        addi    $a0, $a0, 4             # Move onto the next element of the original array
        addi    $s2, $s2, 4             # Move onto the next element of array $s0
        j       mergeloop
else:
        sw      $t3, 0($a0)             # Store value of current element in array $s1 to the
                                        # current position of original array
        addi    $t1, $t1, 4             # Increment index $t1
        addi    $a0, $a0, 4             # Move onto the next element of the original array
        addi    $s3, $s3, 4             # Move onto the next element of array $s0
        j       mergeloop               # Loop again

remain1:                                # Copy the remaining elements of array $s0 to original array
```

+ Sau đó, đưa những phần tử còn lại của subarray vào trong array:

```
remain1:                                # Copy the remaining elements of array $s0 to original array
        bge     $t0, $t6, remain2       # Loop condition
        lw      $t2, 0($s2)             # Assign $s0's current value to current element of original array
        sw      $t2, 0($a0)
        addi    $t0, $t0, 4             # Increase index and move to the next element of both array
        addi    $a0, $a0, 4
        addi    $s2, $s2, 4
        j       remain1                 # Loop again
remain2:                                # Copy the remaining elements of array $s0 to original array
        bge     $t1, $t7, mergedone     # Loop condition
        lw      $t3, 0($s3)             # Assign $s1's current value to current element of original array
        sw      $t3, 0($a0)
        addi    $t1, $t1, 4             # Increase index and move to the next element of both array
        addi    $a0, $a0, 4
        addi    $s3, $s3, 4
        j       remain2
mergedone:
        jal     print                   # Each time a merge finished, print the array
        jr      $ra                     # Return
```

+ Sau khi merge xong, in dãy ra và kết thúc hàm.

Hiện thực hàm in:

```
print:                                  # print the array
        la      $a0, myarray            # load address of array
        addi    $t0, $0, 0              # initialize the starting index
        addi    $t1, $0, 80             # length of array
        jal     prloop
prloop:
        bge     $t0, $t1, prend         # if index >= length, jump to prend
        lw      $a0, myarray($t0)       # print element at index
        li      $v0, 1
        syscall
        la      $a0, spa                # print a space betwwen printing an element
        addi    $v0, $0, 4
        syscall
        addi    $t0, $t0, 4             # incement index
        b       prloop                  # loop
prend:
        la      $a0, eol                # Get to new line
        addi    $v0, $0, 4
        syscall
        lw      $ra, 0($sp)             # Load the return address
        addi    $sp, $sp, 16            # Adjust the stack pointer
        jr      $ra                     # Return
```

5. Kết thúc chương trình.

IV. Tính toán thời gian thực thi chương trình:

- Ta áp dụng công thức tính thời gian thực thi:

Time = Instruction Count × CPI × cycle time

- Giả sử CPI mỗi lệnh = 1, máy tính MIPS có tần số 2GHz. Lúc này:

Cycle time = 1/ClockRate = ½(2x10^9) = 0.5ns

- Sử dụng tool Instruction Counter của MARS, ta sẽ thu được Instruction Count và số lượng các lệnh thuộc 3 định dạng.

- Sử dụng tool Instruction Statistics của MARS, ta sẽ thu được số lượng các loại lệnh đã được thực thi.

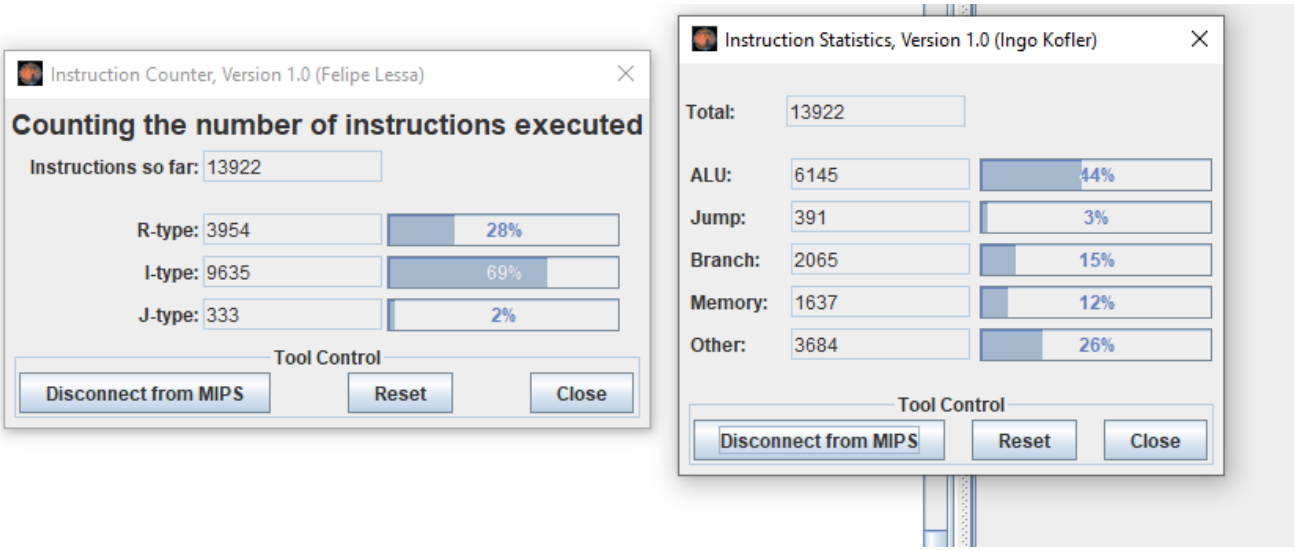
VD: Ta có 1 mảng gồm 20 phần tử ngẫu nhiên được nhập vào và thu được kết quả:

```
Enter the 20 Elements:
12
321
0
1
-32
0
2
-39
14
41
342
-1
41
34
34
21
-9
14
341
143
```

```
12 321 0 1 -32 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
12 321 0 1 -32 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
12 321 0 1 -32 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
12 321 0 1 -32 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
12 321 0 1 -32 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
12 321 0 1 -32 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
12 321 0 -32 1 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
12 321 -32 0 1 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 0 2 -39 14 41 342 -1 41 34 34 21 -9 14 341 143
-32 0 1 12 321 -39 0 2 14 41 342 -1 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 342 -1 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 342 -1 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 342 -1 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 342 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 342 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 342 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 342 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 342 41 34 34 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 342 34 34 41 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 21 -9 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 -9 21 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 -9 21 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 -9 21 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 -9 21 14 341 143
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 -9 21 14 143 341
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 -9 21 14 143 341
-39 -32 0 0 1 2 12 14 41 321 -1 34 34 41 342 -9 14 21 143 341
-39 -32 0 0 1 2 12 14 41 321 -9 -1 14 21 34 34 41 143 341 342
-39 -32 -9 -1 0 0 1 2 12 14 14 21 34 34 41 41 143 321 341 342

-- program is finished running --
```

Sử dụng các tool Intruction Counter và Instruction Statistics, ta thu được:



Như vậy, tổng số Instruction là 13922.

Thời gian thực thi chương trình trên là :

Time = Instruction Count * CPI * cycle time = 13853 * 1 * 0.5ns = 6921ns

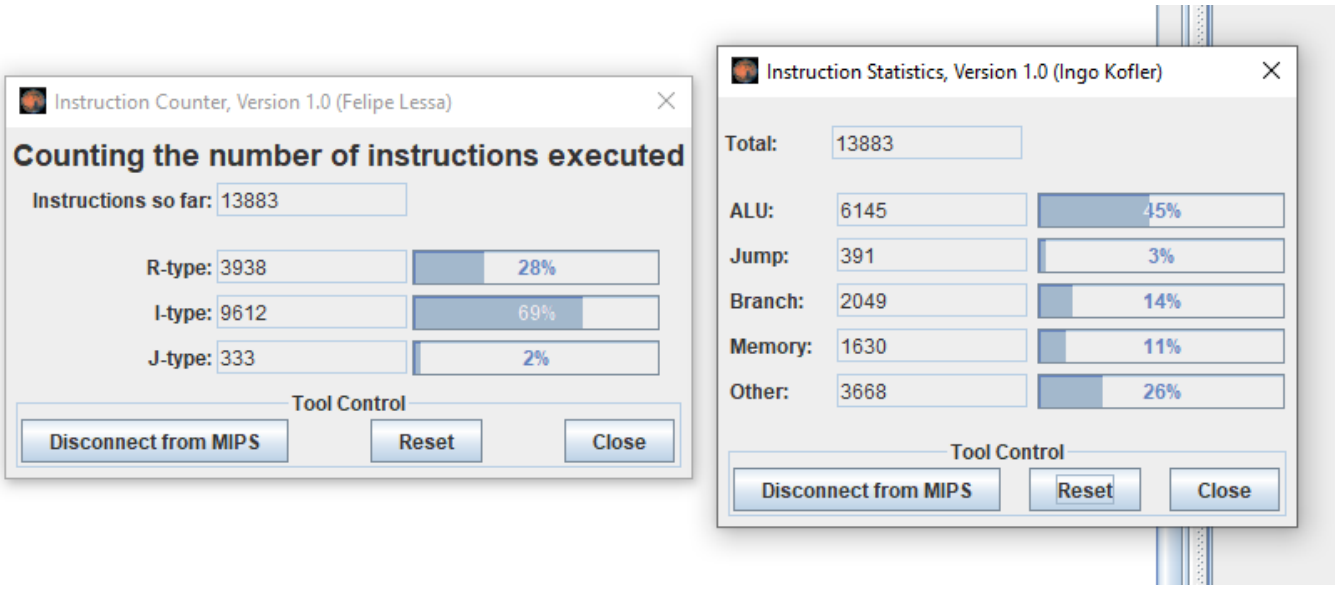
- Sau đây là 1 số test case khác:

VD2:

```
Enter the 20 Elements:
12
-12
0
12
3
1
4
5
2
-10
20
19
47
21
134
-9
12
4
56
98
```

```
-12 -10 -9 0 1 2 3 4 4 5 12 12 12 19 20 21 47 56 98 134

-- program is finished running --
```



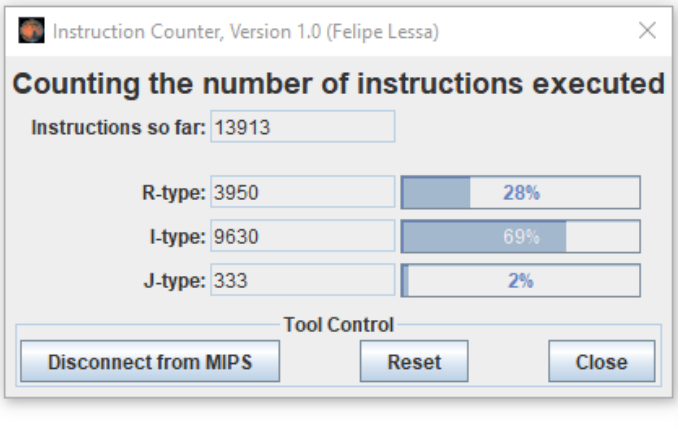
Time execute = 6941.5ns

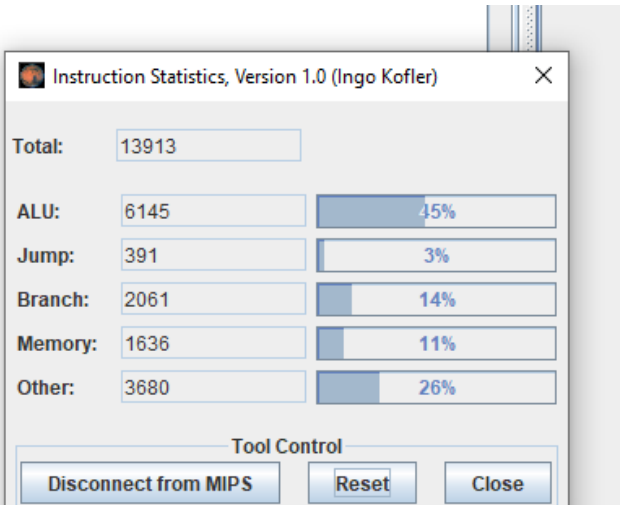
VD3:

```
Enter the 20 Elements:
3
0
1
0
3
2
41
0
3
1
4
5
1
6
7
3
0
1
4
1
```

```
0 0 0 0 1 1 1 1 1 2 3 3 3 3 4 4 5 6 7 41

-- program is finished running --
```



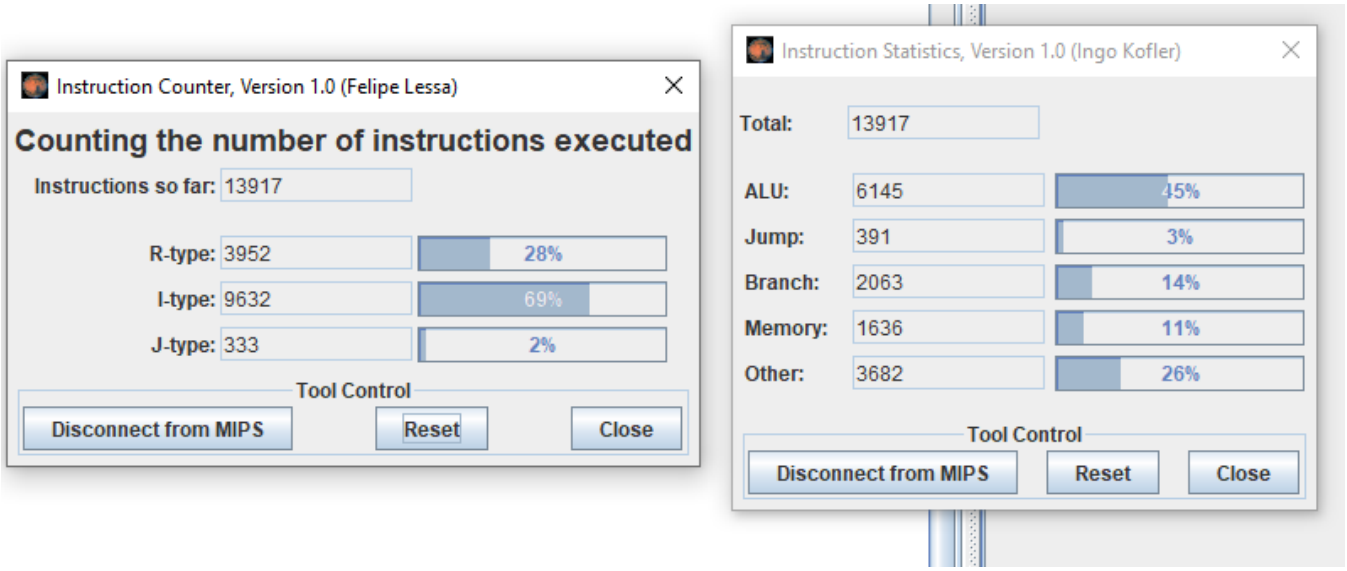


Time execute = 6956.5ns

VD4:

```
Enter the 20 Elements:
-21
-4
-12
-34
-12
-43
-132
-543
-7
-234
-645
-2
-1
54
-82
-8
-20
-40
-9
-12
-645 -543 -234 -132 -82 -43 -40 -34 -21 -20 -12 -12 -12 -9 -8 -7 -4 -2 -1 54

-- program is finished running --
```



Time execute = 6958.5ns

Qua 4 VD trên, có thể thấy được với các dãy số ngẫu nhiên, thuật toán sắp xếp merge sort có run time tương đối ổn định, hầu như không có khác biệt.

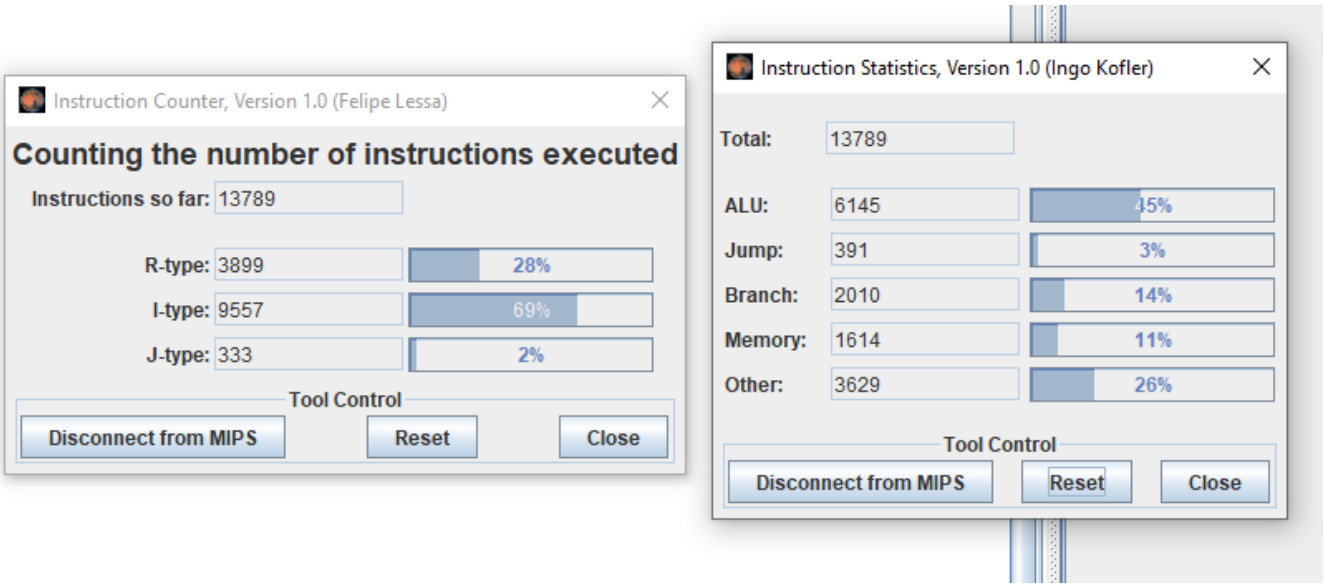
Sau đây, ta sẽ thử 1 vài test case khác đặc biết hơn:

VD5:

```
Enter the 20 Elements:
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

-- program is finished running --
```

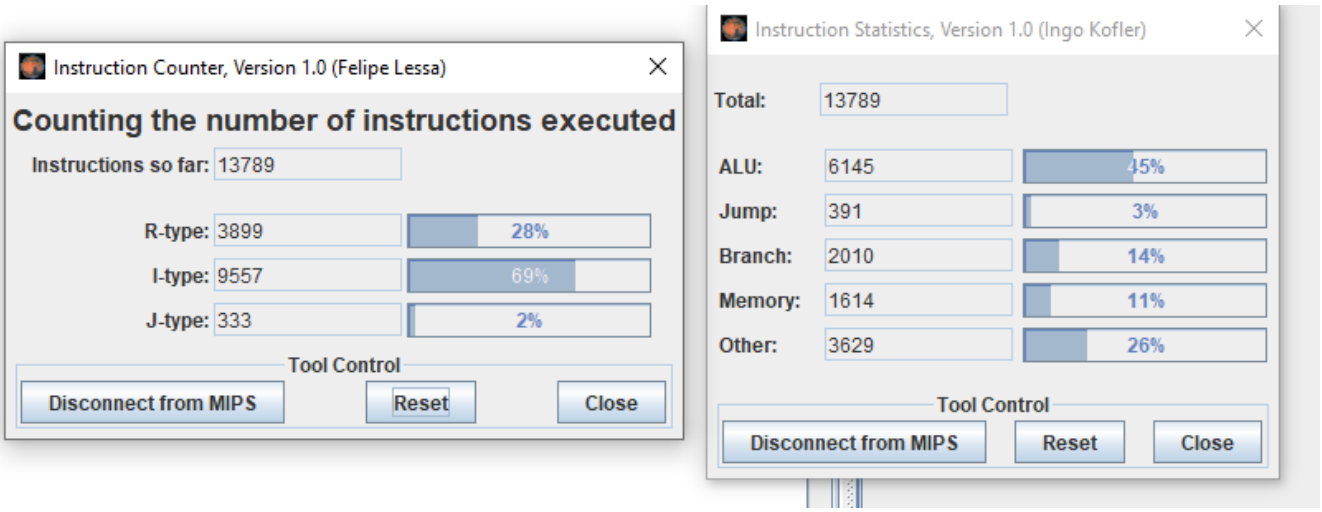


Time execute = 6894.5ns

VD6:

```
Enter the 20 Elements:
-13
-11
-10
-9
0
1
4
14
15
18
19
20
43
53
65
78
79
90
92
100
-13 -11 -10 -9 0 1 4 14 15 18 19 20 43 53 65 78 79 90 92 100

-- program is finished running --
```



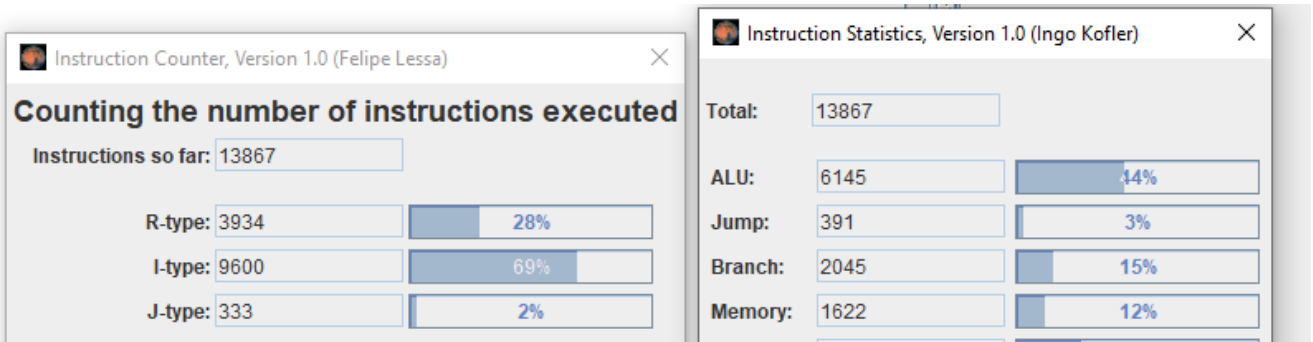
Time execute = 6894.5ns

VD7:

```
Enter the 20 Elements:
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
```

```
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

-- program is finished running --
```



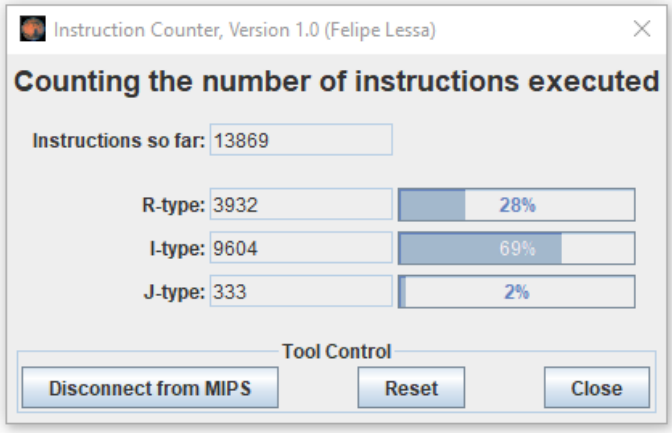
Time execute = 6933.5ns

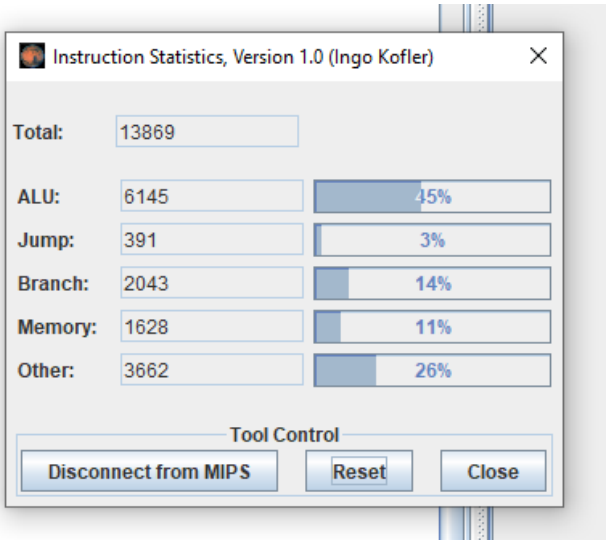
VD8:

```
Enter the 20 Elements:
1
2
3
4
-3
-2
-1
0
-10
-9
-8
-7
7
8
9
10
5
6
7
8
```

```
-10 -9 -8 -7 -3 -2 -1 0 1 2 3 4 5 6 7 7 8 8 9 10

-- program is finished running --
```





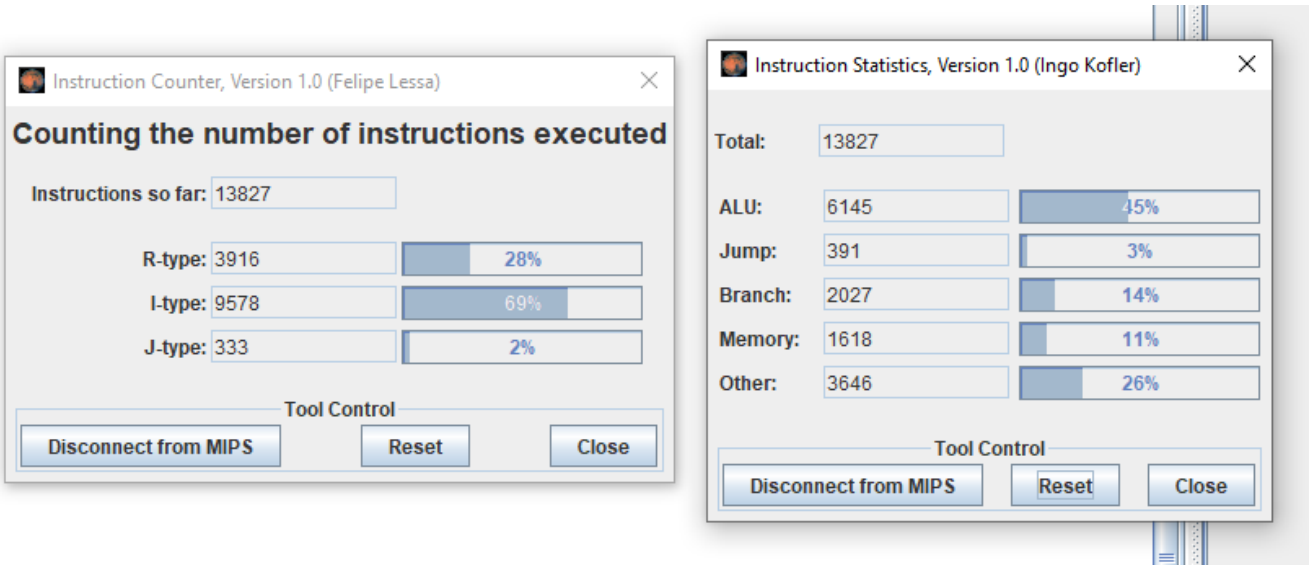
Time execute = 6934.5ns

VD9:

```
Enter the 20 Elements:
1
2
3
4
5
6
7
8
9
10
20
19
18
17
16
15
14
13
12
11
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

-- program is finished running --
```



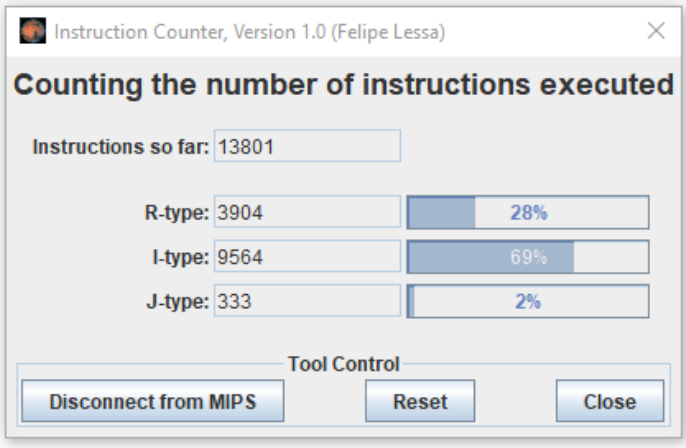
Time execute = 6913.5ns

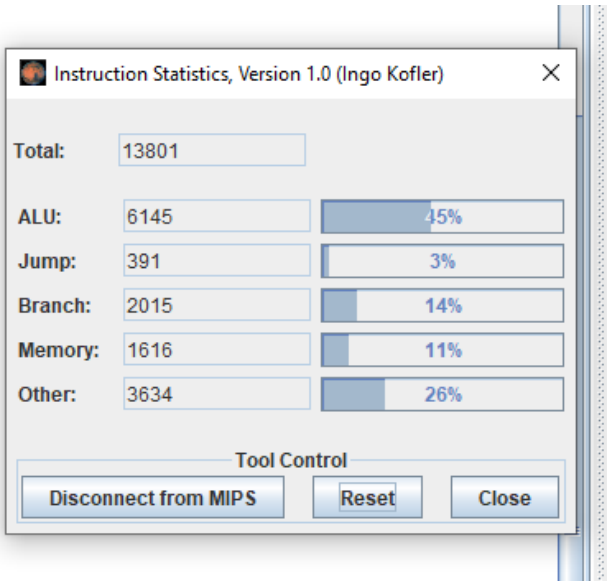
VD10:

```
Enter the 20 Elements:
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2

-- program is finished running --
```





Time execute = 6900.5ns

Qua các ví dụ trên, ta có thể thấy được thời gian thực thi hàm qua các test case hầu như không có khác biệt. Merge sort tuy là 1 trong những giải thuật sắp xếp nhanh nhất nhưng lại không tốt cho các dãy đã được sắp xếp do nó phải đệ quy cho tới khi mảng con còn 1 phần tử. Vì thế nên để khắc phục điểm bất lợi cho merge sort, người ta sẽ sử dụng parallel merge sort(chia mảng ra thành nhiều phần rồi sắp xếp từng mảng đó bằng giải thuật sắp xếp khác rồi merge chúng lại với nhau).

KẾT THÚC BÁO CÁO CỦA NHÓM