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)

BÀI TẬP LỚN --- Đề tài 6

SẮP XẾP CHUỖI – CHO MỘT CHUỖI SỐ NGUYÊN 20 PHẦN TỬ. SỬ DỤNG HỢP NGỮ ASSEMBLY MIPS, VIẾT THỦ TỤC SẮP XẾP CHUỖI ĐÓ THEO THỨ TỰ TĂNG DẦN BẰNG GIẢI THUẬT MERGE SORT.

Giảng viên hướng dẫn: Trần Thanh Bình
Võ Tấn Phương

Nhóm sinh viên thực hiện: Võ Nguyễn Thiện Nhân – 1910409
Lê Ngọc Minh Nhân – 1910402
Nguyễn Duy Khang – 1910238

Tp. Hồ Chí Minh, Tháng 12/2020.

MỤC LỤC

I. Ý TƯỞNG GIẢI THUẬT 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 do John von Neumann đưa ra lần đầu năm 1945. Một thuật toán chi tiết được Goldstine và Neumann đưa ra năm 1948.

1. ***Trộn hai danh sách đã được sắp xếp.***

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

Ví dụ: Cho hai danh sách ***a = (1,3,7,9)***, ***b = (2,6),*** quá trình hòa nhập diễn ra như sau:

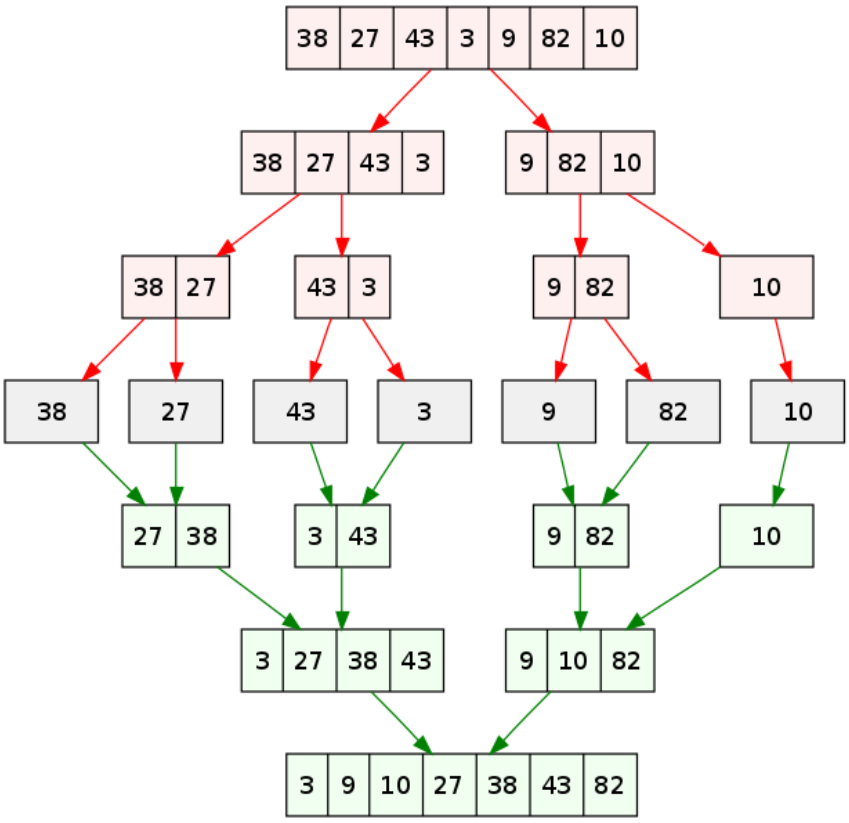
Danh sách a	Danh sách b	So sánh	Danh sách c
1, 3, 7, 9	2, 6	1 < 2	1
3, 7, 9	2, 6	2 < 3	1, 2
3, 7, 9	6	3 < 6	1, 2, 3
7, 9	6	6 < 7	1, 2, 3, 6
7, 9			1, 2, 3 ,6 ,7, 9

2. ***Trộn tại chỗ.***

Giả sử trong danh sách a[1..n] có 2 danh sách con kề nhau a[k_1..k_2] và a[k_2+1..k_3] đã được sắp. Ta áp dụng cách trộn tương tự như trên để trộn hai danh sách con vào một danh sách tạm T[k_1..k_3] rồi trả lại các giá trị của danh sách tạm T về danh sách A. Làm như vậy gọi là trộn tại chỗ.

3. ***Sắp xếp trộn đệ quy.***

Thủ tục đệ quy 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. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.



1-Minh họa giải thuật Merge Sort

II. HIỆN THỰC GIẢI THUẬT MERGE SORT BẰNG HỢP NGỮ MIPS

Để có thể hiện thực được giải thuật Merge Sort, ta sẽ cần hiện thực hai hàm phụ đó là hàm đệ quy ***MergeSort*** để sắp xếp 2 mảng con tương ứng tăng dần và ***Merge*** để trộn 2 mảng con đó lại với nhau thành 1 mảng tăng dần.

Khởi tạo dữ liệu ban đầu, mảng **arr** cần sắp xếp với số phần tử tương ứng (trong trường hợp như đề bài yêu cầu là 20 phần tử), một số ký tự cần thiết phục vụ cho việc in mảng và một mảng phụ tempArr sẽ được sử dụng trong hàm ***Merge*** để lưu tạm mảng con thứ nhất vào.

```
.data
space:          .asciiz "  "
newline:        .asciiz "\n"
given_arr:      .asciiz "Given Array: "
process:        .asciiz "Sorting process: "
prompt:         .asciiz "Sorted Array: "
tempArr:        .word   0:20
arr:            .word   900 900 80 70 45 30 20 25 10 -50 -40 -30 -10 15 0 60 50 5 999 999
size:           .word   20
```

Trong phần chương trình chính (main), thực hiện việc truyền các tham số tương ứng cho hàm ***MergeSort***, lưu các giá trị cần thiết vào stack để phục hồi, đồng thời jump & link đến hàm ***MergeSort***.

```
        add     $a1, $zero, $zero       # a1 = index of start array
        lw      $a2, size
        addi    $a2, $a2, -1            # a2 = index of end array

        la      $a0, arr                        # a0 = address of array

        jal     MergeSort                       # merge sort array
```

Trong hàm ***MergeSort***, các thao tác cần thực hiện:

+ So sánh chỉ số đầu (start) và chỉ số cuối (end). Nếu start = end thì thoát khỏi hàm ***MergeSort***.

```
beq         $a1, $a2, ExitMergeSort                     # if start = end, then Exit MergeSort

addi        $sp, $sp, -16
sw          $ra, 12($sp)                                # save ra on stack
sw          $a1, 8($sp)                         # save start on stack
sw          $a2, 4($sp)                         # save end on stack

add         $s0, $a1, $a2                               # start + end
srl         $s0, $s0, 1                         # mid = (start + end) / 2
```

+ Ngược lại, lưu các giá trị cần tái sử dụng vào stack như: địa chỉ trả về ($ra), chỉ số đầu ($a1), chỉ số cuối ($a2). Cũng như thực hiện thao tác đệ quy trên hai mảng con của mảng hiện tại, mảng con thứ nhất từ phần tử đầu đến phần tử chính giữa, mảng con thứ hai là phần còn lại của mảng hiện tại. Khi đó, phần tử chính giữa được xác định như sau: ***mid = (start + end)/2.***

```
add     $a2, $zero, $s0         # make end = mid for sort half first array
jal     MergeSort

lw      $s0, 0($sp)             # load mid on stack

add     $a1, $zero, $s0         # make start = mid for sort 2nd half array
addi    $a1, $a1, 1             # start = mid + 1
lw      $a2, 4($sp)             # end = end
jal     MergeSort
```

+ Sau khi kết thúc hàm đệ quy của mảng con thứ nhất, ta sẽ load cái giá trị đã lưu vào stack ra để tái sử dụng, đồng thời thay đổi các tham số $a1 thành phần tử kế tiếp của phần tử chính giữa, $a2 tới phần tử cuối của mảng và jump and link tới hàm ***MergeSort*** của mảng con còn lại.

+ Sau khi kết thúc 2 lần đệ quy trên 2 mảng con tương ứng, ta sẽ được 2 mảng con, mỗi mảng đã được sắp xếp tăng dần. Thực hiện việc load dữ liệu đã lưu từ stack ra để tái sử dụng. Sau đó jump and link tới hàm ***Merge*** để thực hiện thao tác trộn 2 mảng con.

```
lw      $a1, 8($sp)             # a1 = load start on stack
lw      $a2, 4($sp)             # a2 = load end on stack
lw      $a3, 0($sp)             # a3 = load mid on stack

jal     Merge                   # combine array
```

Trong hàm ***Merge***, các thao tác cần thực hiện:

+ Khởi tạo các biến đếm chỉ số cho mảng con bên trái (i = $s0), mảng con bên phải (j = $s2) và mảng tạm lưu kết quả (k = $s1).

```
add     $s0, $zero, $a1         # s0 = i = index of array left
add     $s1, $zero, $a1         # s1 = k = start
add     $s2, $zero, $a3         # s2 = j = index of array right (mid)
addi    $s2, $s2, 1             # j = mid + 1
```

+ Kiểm tra điều kiện và so sánh để điền các phần tử của mảng con bên trái và mảng con bên phải vào mảng tạm, nếu arr[i] <= arr[j] thì điền phần tử arr[i]

của mảng con bên trái vào mảng tạm, ngược lại nếu arr[i] > arr[j] thì điền phần tử arr[j] mảng con bên phải vào mảng tạm. Cứ tiếp tục như vậy cho đến khi biến đếm vượt quá số lượng phần tử của mảng con bên trái hoặc mảng con bên phải

```
Comparision:                            # COMPARE array[i] and array[i] (t1 and t3)
        sll     $t0, $s0, 2             # t0 = i*4
        add     $t0, $t0, $a0           # address of array[i]
        lw      $t1, 0($t0)             # t1 = value of array[i]

        add     $t0, $zero, $zero
        sll     $t0, $s2, 2             # t0 = j*4
        add     $t0, $t0, $a0           # address of array[j]
        lw      $t3, 0($t0)             # t3 = value of array[j]

        sub     $t6, $t3, $t1           # if array[j] < array[i]
        bltz    $t6, FillArrayRight

FillArrayLeft:
        la      $t2, tempArr            # load address of temp array
        sll     $t0, $s1, 2             # t0 = index of next element of array * 4 = k*4
        add     $t2, $t2, $t0           # address of next element of temp array
        sw      $t1, 0($t2)             # temp[k] = array[i]
```

+ Nếu điều kiện phía trên không thỏa thì chúng ta sẽ tiếp tục điền các phần tử còn lại của mảng con bên trái hoặc mảng con bên phải vào mảng tạm.

```
CheckCondition:
        sub     $t6, $s0, $a3           # if i > mid, exit fill array left
        bgtz    $t6, FillRestArrayRight

        sub     $t6, $s2, $a2           # if j > end, exit fill array right
        bgtz    $t6, FillRestArrayLeft
```

```
FillRestArrayLeft:
            sub         $t6, $s0, $a3                   # if i > mid (fill completed), then set variable
            bgtz        $t6, SetVariable

            la          $t0, tempArr                    # load address of temp array
            sll         $t1, $s1, 2                     # t1 = k*4
            add         $t0, $t0, $t1                   # t0 = address of next element of temp array

            sll         $t2, $s0, 2                     # i*4
            add         $t2, $t2, $a0                   # t2 = address of array[i]
            lw          $t1, 0($t2)                     # t1 = value of array[i]

            sw          $t1, 0($t0)                     # store value of array[i] to temp array

            addi        $s1, $s1, 1                     # k++
            addi        $s0, $s0, 1                     # i++
            j           FillRestArrayLeft

FillRestArrayRight:
            sub         $t6, $s1, $a2                   # if k > end (fill completed), then set variable
```

+ Sau khi đã nhập 2 mảng con vào mảng tạm, tiến hành lặp để in và lưu mảng tạm vào đúng vị trí trong mảng kết quả (địa chỉ tại $a0). Khởi tạo biến đếm $t0 trên mảng tạm, khi đếm qua vị trí cuối cùng của mảng tạm ($t0 > $t1) thì dừng hàm này lại. Để lưu, load giá trị tại vị trí $t0 của mảng tạm vào $t4, sau đó store giá trị này vào vị trí tương ứng trong mảng kết quả và in nó ra màn hình. Tăng biến đếm $t0 và tiếp tục lặp.

```
SetVariable:
        add     $t0, $a1, $zero             # t0 = current
        addi    $t1, $a2, 0                 # t1 = end
        la      $t2, tempArr

SortEachMerge:
        sub     $t6, $t0, $t1               # if current > end, then Sort End
        bgtz    $t6, EndSortEachMerge

        sll     $t3, $t0, 2                 # current*4
        add     $t3, $t3, $t2               # t3 = address of current element of temp array
        lw      $t4, 0($t3)                 # t4 = value of current element of temp array

        sll     $t3, $t0, 2                 # current*4
        add     $t3, $t3, $a0               # t3 = address of next element of array
        sw      $t4, 0($t3)                 # store: value of next element of temp array = value of
                                            #       current element of temp array

        add     $t5, $zero, $a0             # t5 = temp address for a0
                                            # a0 now is being used for print

        add     $a0, $zero, $t4             # print array each merge
                                            # t4 = value need to printed

        li      $v0, 1
        syscall
        la      $a0, space
        li      $v0, 4
        syscall

        add     $a0, $zero, $t5             # return value for a0
```

+ Đã sáp nhập xong, trở về vị trí sau ***Merge*** trong hàm ***MergeSort***.

```
        add     $t5, $zero, $a0             # t5 = a0 for print newline after each merge

        la      $a0, newline
        li      $v0, 4
        syscall

        add     $a0, $zero, $t5
        jr      $ra
```

Sau các lần đệ quy của ***MergeSort***, in mảng kết quả cuối cùng đã sắp xếp ra màn hình. Kết thúc chương trình.

III. KẾT QUẢ MỘT SỐ TESTCASE.

 # TESTCASE 1

```
arr:    .word    900 900 80 70 45 30 20 25 10 -50 -40 -30 -10 15 0 60 50 5 999 999
```

 # RESULT:

```
Given Array:
900  900  80  70  45  30  20  25  10  -50  -40  -30  -10  15  0  60  50  5  999  999
Sorting process:
900  900
80  900  900
45  70
45  70  80  900  900
20  30
20  25  30
-50  10
-50  10  20  25  30
-50  10  20  25  30  45  70  80  900  900
-40  -30
-40  -30  -10
0  15
-40  -30  -10  0  15
50  60
5  50  60
999  999
5  50  60  999  999
-40  -30  -10  0  5  15  50  60  999  999
-50  -40  -30  -10  0  5  10  15  20  25  30  45  50  60  70  80  900  900  999  999
Sorted Array:
-50  -40  -30  -10  0  5  10  15  20  25  30  45  50  60  70  80  900  900  999  999
-- program is finished running --
```

 # TESTCASE 2

```
arr:    .word  950 800 749 655 589 456 321 258 147 98 88 79 65 54 49 38 22 11 5
0
```

#RESULT:

```
Given Array:
950  800  749  655  589  456  321  258  147  98  88  79  65  54  49  38  22  11  5  0
Sorting process:
800  950
749  800  950
589  655
589  655  749  800  950
321  456
258  321  456
98  147
98  147  258  321  456
98  147  258  321  456  589  655  749  800  950
79  88
65  79  88
49  54
49  54  65  79  88
22  38
11  22  38
0  5
0  5  11  22  38
0  5  11  22  38  49  54  65  79  88
0  5  11  22  38  49  54  65  79  88  98  147  258  321  456  589  655  749  800  950
Sorted Array:
0  5  11  22  38  49  54  65  79  88  98  147  258  321  456  589  655  749  800  950
-- program is finished running --
```

 # TESTCASE 3

```
arr:    .word  696 325 235 123 95 80 55 55 41 0 956 895 753 652 145 58 49 5 2 2
```

#RESULT:

```
Given Array:
696  325  235  123  95  80  55  55  41  0  956  895  753  652  145  58  49  5  2  2
Sorting process:
325  696
235  325  696
95  123
95  123  235  325  696
55  80
55  55  80
0  41
0  41  55  55  80
0  41  55  55  80  95  123  235  325  696
895  956
753  895  956
145  652
145  652  753  895  956
49  58
5  49  58
2  2
2  2  5  49  58
2  2  5  49  58  145  652  753  895  956
0  2  2  5  41  49  55  55  58  80  95  123  145  235  325  652  696  753  895  956
Sorted Array:
0  2  2  5  41  49  55  55  58  80  95  123  145  235  325  652  696  753  895  956
-- program is finished running --
```

 # TESTCASE 4

```
arr:    .word  -900 999 255 25 -555 89 78 0 1 -88 7 56 78 -100 -56 56 -112 562 123
-1000
```

#RESULT

```
Given Array:
-900 999 255 25 -555 89 78 0 1 -88 7 56 78 -100 -56 56 -112 562 123 -
1000
Sorting process:
-900 999
-900 255 999
-555 25
-900 -555 25 255 999
78 89
0 78 89
-88 1
-88 0 1 78 89
-900 -555 -88 0 1 25 78 89 255 999
7 56
7 56 78
-100 -56
-100 -56 7 56 78
-112 56
-112 56 562
-1000 123
-1000 -112 56 123 562
-1000 -112 -100 -56 7 56 56 78 123 562
-1000 -900 -555 -112 -100 -88 -56 0 1 7 25 56 56 78 78 89 123 255 562
999
Sorted Array:
-1000 -900 -555 -112 -100 -88 -56 0 1 7 25 56 56 78 78 89 123 255 562
999
-- program is finished running --
```

 # TESTCASE 5

```
arr:    .word  -89 -999 -1 -56 -888 -23 -556 -102 -2 -45 -785 -12 -785 -65 -483 -102
-852 -25 -45 -899
```

```
# RESULT
Given Array:
-89 -999 -1 -56 -888 -23 -556 -102 -2 -45 -785 -12 -785 -65 -483 -102 -
852 -25 -45 -899
Sorting process:
-999 -89
-999 -89 -1
-888 -56
-999 -888 -89 -56 -1
-556 -23
-556 -102 -23
-45 -2
-556 -102 -45 -23 -2
-999 -888 -556 -102 -89 -56 -45 -23 -2 -1
-785 -12
-785 -785 -12
-483 -65
-785 -785 -483 -65 -12
-852 -102
-852 -102 -25
-899 -45
-899 -852 -102 -45 -25
-899 -852 -785 -785 -483 -102 -65 -45 -25 -12
-999 -899 -888 -852 -785 -785 -556 -483 -102 -102 -89 -65 -56 -45 -45 -
25 -23 -12 -2 -1
Sorted Array:
-999 -899 -888 -852 -785 -785 -556 -483 -102 -102 -89 -65 -56 -45 -45 -
25 -23 -12 -2 -1
-- program is finished running --
```

 # TESTCASE 6

```
arr:    .word  9 9 2 3 3 3 7 4 5 6 6 6 0 0 1 2 3 10 15 20
```

```
# RESULT
Given Array:
9  9  2  3  3  3  7  4  5  6  6  6  0  0  1  2  3  10  15  20
Sorting process:
9  9
2  9  9
3  3
2  3  3  9  9
3  7
3  4  7
5  6
3  4  5  6  7
2  3  3  3  4  5  6  7  9  9
6  6
0  6  6
0  1
0  0  1  6  6
2  3
2  3  10
15  20
2  3  10  15  20
0  0  1  2  3  6  6  10  15  20
0  0  1  2  2  3  3  3  3  4  5  6  6  6  7  9  9  10  15  20
Sorted Array:
0  0  1  2  2  3  3  3  3  4  5  6  6  6  7  9  9  10  15  20
-- program is finished running --
```

 # TESTCASE 7

```
arr:    .word  413 -430 -903 -922 -835 -362 -101 -87 449 -888 -473 770 -969 547
201 299 -303 516 926 52
```

```
# RESULT
Given Array:
413  -430  -903  -922  -835  -362  -101  -87  449  -888  -473  770  -969  547  201
299  -303  516  926  52
Sorting process:
-430  413
-903  -430  413
-922  -835
-922  -903  -835  -430  413
-362  -101
-362  -101  -87
-888  449
-888  -362  -101  -87  449
-922  -903  -888  -835  -430  -362  -101  -87  413  449
-473  770
-969  -473  770
201  547
-969  -473  201  547  770
-303  299
-303  299  516
52  926
-303  52  299  516  926
-969  -473  -303  52  201  299  516  547  770  926
-969  -922  -903  -888  -835  -473  -430  -362  -303  -101  -87  52  201  299  413
449  516  547  770  926
Sorted Array:
-969  -922  -903  -888  -835  -473  -430  -362  -303  -101  -87  52  201  299  413
449  516  547  770  926
-- program is finished running --
```

 # TESTCASE 8

```
arr:    .word  268 -536 -193 -726 -34 274 -265 409 -468 -757 866 271 -137 -826
274 -213 -828 -508 -508 -151
```

```
# RESULT
Given Array:
268 -536 -193 -726 -34 274 -265 409 -468 -757 866 271 -137 -826 274 -
213 -828 -508 -508 -151
Sorting process:
-536  268
-536  -193  268
-726  -34
-726  -536  -193  -34  268
-265  274
-265  274  409
-757  -468
-757  -468  -265  274  409
-757  -726  -536  -468  -265  -193  -34  268  274  409
271  866
-137  271  866
-826  274
-826  -137  271  274  866
-828  -213
-828  -508  -213
-508  -151
-828  -508  -508  -213  -151
-828  -826  -508  -508  -213  -151  -137  271  274  866
-828  -826  -757  -726  -536  -508  -508  -468  -265  -213  -193  -151  -137  -34  268
271  274  274  409  866
Sorted Array:
-828  -826  -757  -726  -536  -508  -508  -468  -265  -213  -193  -151  -137  -34  268
271  274  274  409  866
-- program is finished running --
```

 # TESTCASE 9

```
arr:    .word  99 98 40 6 4 74 12 68 98 91 43 80 72 2 39 73 14 1 93 56
```

```
# RESULT
Given Array:
99  98  40  6  4  74  12  68  98  91  43  80  72  2  39  73  14  1  93  56
Sorting process:
98  99
40  98  99
4  6
4  6  40  98  99
12  74
12  68  74
91  98
12  68  74  91  98
4  6  12  40  68  74  91  98  98  99
43  80
43  72  80
2  39
2  39  43  72  80
14  73
1  14  73
56  93
1  14  56  73  93
1  2  14  39  43  56  72  73  80  93
1  2  4  6  12  14  39  40  43  56  68  72  73  74  80  91  93  98  98  99
Sorted Array:
1  2  4  6  12  14  39  40  43  56  68  72  73  74  80  91  93  98  98  99
-- program is finished running --
```

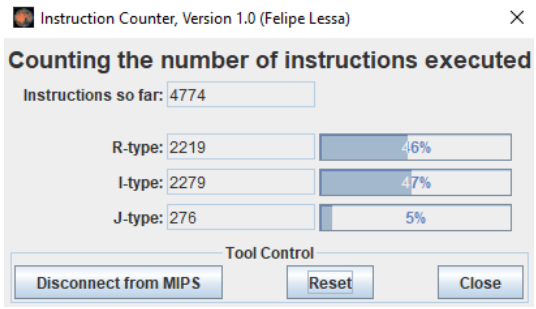
 # TESTCASE 10

```
arr:    .word  -21 -90 58 -164 88 82 -47 -77 -39 -61 -39 -59 -43 83 -129 -100 -22 -129 88 -90
```

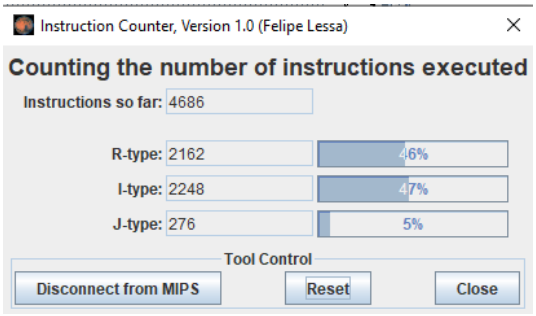
```
Given Array:
-21 -90 58 -164 88 82 -47 -77 -39 -61 -39 -59 -43 83 -129 -100 -22 -129 88 -90
Sorting process:
-90 -21
-90 -21 58
-164 88
-164 -90 -21 58 88
-47 82
-77 -47 82
-61 -39
-77 -61 -47 -39 82
-164 -90 -77 -61 -47 -39 -21 58 82 88
-59 -39
-59 -43 -39
-129 83
-129 -59 -43 -39 83
-100 -22
-129 -100 -22
-90 88
-129 -100 -90 -22 88
-129 -129 -100 -90 -59 -43 -39 -22 83 88
-164 -129 -129 -100 -90 -90 -77 -61 -59 -47 -43 -39 -39 -22 -21 58 82 83 88 88
Sorted Array:
-164 -129 -129 -100 -90 -90 -77 -61 -59 -47 -43 -39 -39 -22 -21 58 82 83 88 88
-- program is finished running --
```

IV. THỐNG KÊ CÁC LỆNH VÀ TÍNH TOÁN THỜI GIAN CHẠY CỦA CHƯƠNG TRÌNH.

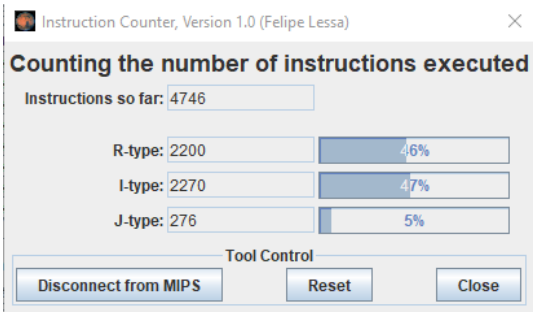
a. Thống kê các lệnh.



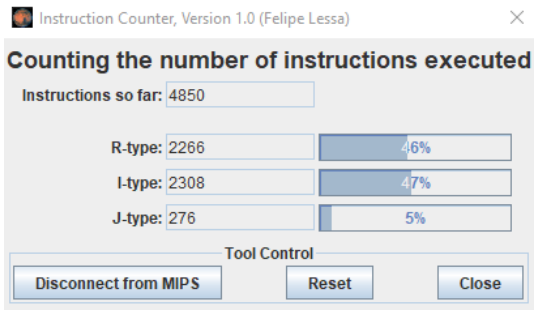
2- ***Testcase 1***: *900 900 80 70 45 30 20 25 10 -50 -40 -30 -10 15 0 60 50 5 999 999*



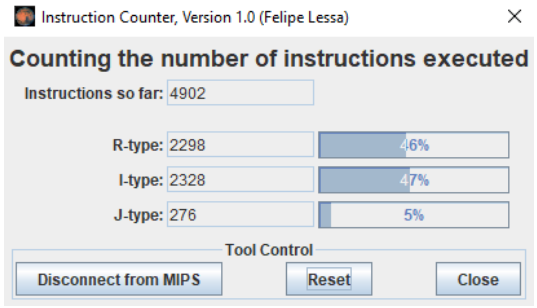
3- ***Testcase 2***: *950 800 749 655 589 456 321 258 147 98 88 79 65 54 49 38 22 11 5 0*



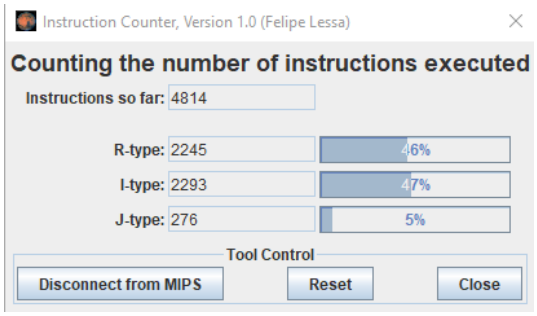
4- ***Testcase 3***: *696 325 235 123 95 80 55 55 41 0 956 895 753 652 145 58 49 5 2 2*



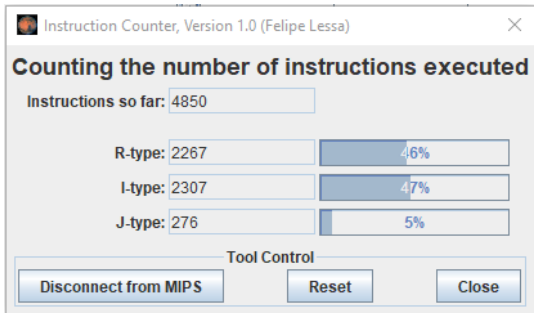
5- ***Testcase 4***: *-900 999 255 25 -555 89 78 0 1 -88 7 56 78 -100 -56 56 -112 562 123 -1000*



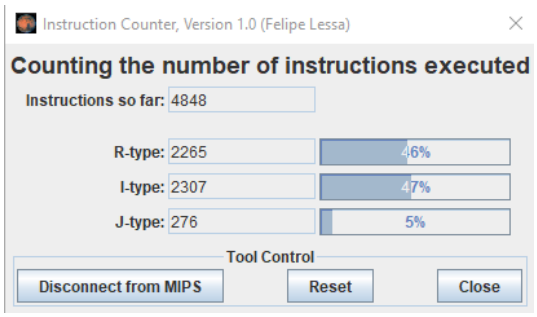
6- ***Testcase 5***: *-89 -999 -1 -56 -888 -23 -556 -102 -2 -45 -785 -12 -785 -65 -483 -102 -852 -25 -45 -899*



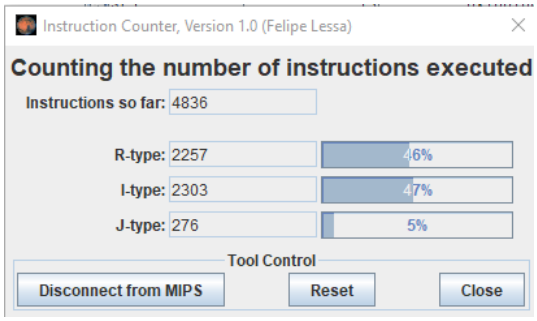
7- ***Testcase 6***: *9 9 2 3 3 3 7 4 5 6 6 6 0 0 1 2 3 10 15 20*



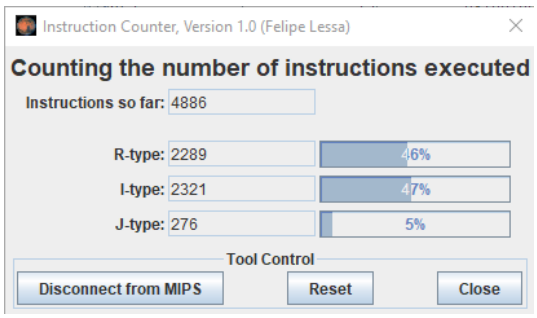
8- ***Testcase 7***: *413 -430 -903 -922 -835 -362 -101 -87 449 -888 -473 770 -969 547 201 299 -303 516 926 52*



9- ***Testcase 8***: *268 -536 -193 -726 -34 274 -265 409 -468 -757 866 271 -137 -826 274 -213 -828 -508 -508 -151*



10- ***Testcase 9***: *99 98 40 6 4 74 12 68 98 91 43 80 72 2 39 73 14 1 93 56*



11- ***Testcase 10***: *-21 -90 58 -164 88 82 -47 -77 -39 -61 -39 -59 -43 83 -129 -100 -22 -129 88 -90*

b. *Tính toán thời gian chạy của chương trình.*

Mỗi lệnh trong chương trình có CPI = 1. Thời gian thực thi của chương trình được tính bằng:

$$t = \frac{(CPI \times n)}{f}$$

Với n là tổng số lệnh đã được thực hiện của chương trình và f là tần số máy.

Testcase	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Thời gian t (ns)	2.387	2.343	2.373	2.425	2.451	2.407	2.425	2.424	2.418	2.443

Table 1- Thời gian thực thi ứng với các testcase

Tham khảo

[1]. Phạm Quốc Cường (2019) Kiến trúc máy tính, NXB: Đại học Quốc gia TP.HCM.

[2]. https://www.geeksforgeeks.org/merge-sort/

----ஜ۩۞۩ஜ----