BÁO CÁO BÀI TẬP LỚN
MÔN: KIẾN TRÚC MÁY TÍNH

Problem 5: Quick Sort algorithm

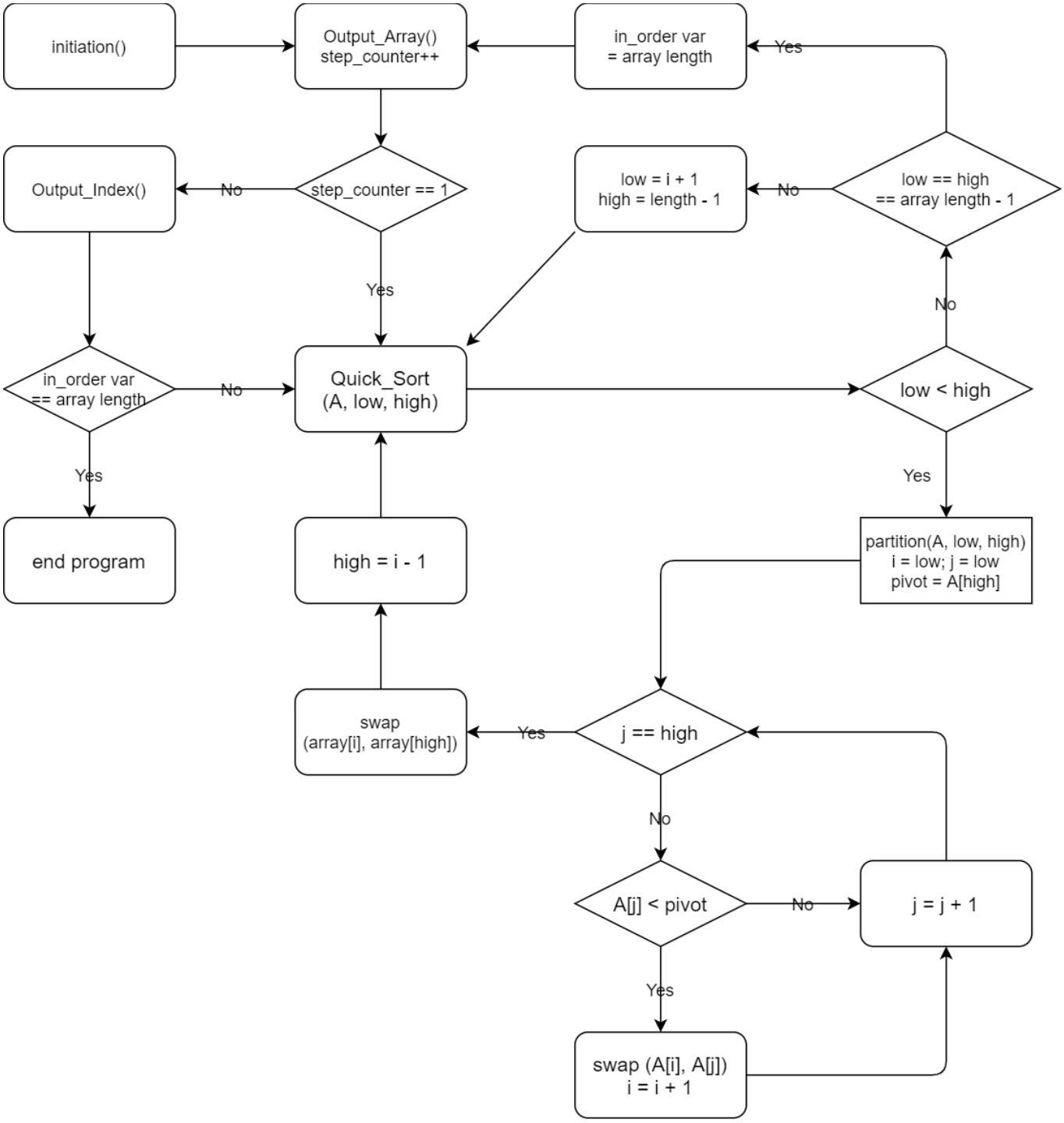
MSSV	Họ tên
1449529	Phạm Bùi Hải Thanh

A. Giải thuật Quick Sort theo lược đồ Lomuto:

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p - 1)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo
    for j := lo to hi do
        if A[j] < pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```

B. Lưu đồ triển khai giải thuật theo lược đồ Lomuto:



C. Thuyết minh phần coding:

- Số dòng lệnh (kể cả dòng trống): 177
- Loại lệnh: R-type, I-type, J-type
- Vì đề bài yêu cầu xuất ra từng bước trong quá trình demo, nên trong output() loop (xuất ra từng phần tử của chuỗi số) sinh viên đã thêm 3 dòng lệnh, 1 label để kiểm tra chuỗi đã đúng thứ tự từ bé đến lớn chưa. Code cụ thể:

```
#$t5 = array[i+1]; $t6 = array[i];

#trước khi xuất chuỗi số chương trình khởi tạo $t7=0

#khi array[i+1]>=array[i], tăng $t7 thêm 1 đơn vị

bgt $t6, $t5, next    #if array[i] > array[i+1]

addi $t7, $t7, 1      #check whether array is in order

next:

move $t6, $t5         #nạp phần tử vừa xuất vào $t6
```

4 dòng lệnh & label này giúp cho chương trình dừng ngay lập tức khi chuỗi đã sắp xếp đúng thứ tự, giảm số lần chạy có thể đến 80%.

Ví dụ chuỗi số input:

12, 15, 10, 5, 7, 3, 2, 1, 31, 46, 172, 208, 13, 93, 65, 112, 1449529, 17, 92, 0

Nếu không có 4 dòng lệnh này, số lần chạy là 66.

Nếu có 4 dòng lệnh này, số lần chạy là 28.

Hơn nữa, theo lý thuyết nếu không xuất ra chuỗi số theo từng bước chương trình chạy, thì với chuỗi số n phần tử chương trình sẽ chạy tối đa $O(n^2)$ lần khi chuỗi đã theo đúng thứ tự hoặc các phần tử bằng nhau. 4 dòng lệnh này sẽ khiến chương trình chỉ chạy 1 lần khi chuỗi có 2 đặc điểm trên.

- Hàm Quick_Sort():
- $s0 lưu địa chỉ chuỗi số input. Hàm Quick_Sort() sẽ nhận 2 biến $s1, $s2 lưu vị trí đầu (low) và cuối (high) của đoạn chuỗi hàm Quick_Sort() nhận để sắp xếp. Ban đầu $s1 = 0; $s2 = array_length – 1. Thanh ghi $a1 lưu chiều dài chuỗi.

```
#initiation

li $s1, 0          #s1 = start
```

```
subi $s2, $a1, 1 #s2 = end
```

- Khi $s1 < $s2, hàm Quick_Sort() sẽ gọi hàm partition().

```
quickSort:
```

```
……
jal partition
```

- Hàm partition() có 2 tham số $t1, $t2. Khi được gọi, hàm partition() sẽ nạp:

```
move $t1, $s1          #i = low
move $t2, $s1          #j = low
```

- Hàm partition() nhận phần tử chuỗi ở vị trí $s2 là pivot

```
sll $t4, $s2, 2
add $t4, $t4, $s0
lw $s3, 0($t4)         #pivot = array[high]
```

- Trong vòng lặp for(), biến $t1 sẽ tăng một đơn vị nếu phần tử chuỗi ở vị trí $t2 < pivot, đồng thời hàm swap() sẽ được gọi để hoán vị phần tử array[$t1] & array[$t2]. Sau mỗi lần gọi loop for() thì $t2 tăng một đơn vị. Khi $t2 == $s2, thoát vòng for(), gọi swap() hoán vị array[$t1] & array[$t2], trả giá trị $t1 (vị trí của pivot trong chuỗi) về caller Quick_Sort().

```
#partition():
move $t1, $s1          #i = low
move $t2, $s1          #j = low
……        #pivot = array[high]
```

```
forLoop:
```

```
beq $t2, $s2, end4     #if j == high -> exit for() loop
……        #$t6 = array[j]
```

```
if:
bge $t6, $s3, endIf   #if array[j] >= pivot, skip swap()
j swap   #if array[j] < pivot -> swap array[i] & array[j]
```

```
cont4Loop:
addi $t1, $t1, 1
endIf:
addi $t2, $t2, 1
j forLoop
swap:
……
blt $t2, $s2, cont4Loop   #continue for() loop
beq $t2, $s2, contEnd4    #end for() loop
end4:
j swap
contEnd4:
……                #initiation for output()
j showStep        #jump to output()
continue:         #xuất các chỉ số $s1, $s2, pivot, vị trí pivot
```

- Hàm partition() trả về giá trị $t1. Hàm Quick_Sort() sẽ liên tục gọi đệ quy hàm Quick_Sort() với $s2 = $t1 – 1 (Left Wing). Giá trị $s2 sẽ giảm dần. Mỗi lần hàm partition() chạy xong sẽ gọi hàm output() để xuất chuỗi ở lần chạy đó.

```
leftWing:
subi $s2, $t1, 1
jal quickSort
```

- Khi $s2 == $s1, các phần tử chuỗi bên trái phần tử array[$t1] đã theo đúng thứ tự. Hàm Quick_Sort() (Right Wing) sẽ được gọi.

```
bge $s1, $s2, rightWing
```

- Hàm Quick_Sort() (Right Wing) kiểm tra nếu $s1 != $s2 != array_length -1 thì sẽ gán $s1 = $t1 + 1; $s2 = array_length -1, gọi đệ quy Quick_Sort() cho nhánh bên phải array[$t1] chưa đúng thứ tự.

```
rightWing:

addi $s1, $t1, 1

subi $s2, $a1, 1

j quickSort
```

- Nếu $s1 == $s2 == array_length – 1, cờ báo hiệu chuỗi đã theo đúng thứ tự được bật, caller gọi hàm xuất chuỗi & xuất các chỉ số $s1, $s2, pivot, vị trí pivot trong chuỗi.
- Sau khi xuất các chỉ số, nếu cờ báo chuỗi đã đúng thứ tự được bật ($t7 == array_length), chương trình lập tức dừng lại.

```
beq $t7, $a1, end#end sorting right after the array is in order

jr $ra            #end of partition()

end:

li $v0, 10

syscall
```

- **10 test cases trong chương trình:**

a) Chuỗi số theo thứ tự ngẫu nhiên, chỉ có số không âm, số lần chạy: 28

12, 15, 10, 5, 7, 3, 2, 1, 31, 46, 172, 208, 13, 93, 65, 112, 1449529, 17, 92, 0

b) Chuỗi số thứ tự ngẫu nhiên có số âm, dương, zero. Số lần chạy: 25

1312, 219, -86, -5080, -38, 1413, 1990, -1612, -19, -50, 3, 0, 24, 1449529, 91, 260, 1, 20, 13, -160

c) Chuỗi số câu b) có phần tử lớn nhất ở cuối. Số lần chạy: 20

1413, 1990, -1612, -19, -50, 91, 1312, 219, -86, -38, 260, 1, 20, 13, -160, 3, 0, 24, -5080, 1449529

d) Chuỗi số câu b) có phần tử bé nhất ở cuối. Số lần chạy: 33

-38, 3, 1413, -160, 260, 13, -19, 1990, 24, -1612, 219, 0, 1, 20, -50, 91, -86, 1312, 1449529, -5080

e) Chuỗi số câu b) có phần tử bé nhất ở đầu. Số lần chạy: 31

-5080, 1449529, -86, -50, 91, 260, -1612, -19, 3, 0, 24, -38, 1413, 1990, 1, 20, 13, -160, 1312, 219

f) Chuỗi số có các phần tử bằng nhau. Số lần chạy: 1

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

g) Chuỗi số input đã đúng thứ tự từ bé đến lớn. Số lần chạy: 1

-5080, -1612, -160, -86, -50, -38, -19, 0, 1, 3, 13, 20, 24, 91, 219, 260, 1312, 1413, 1990, 1449529

h) Chuỗi số input đã gần đúng thứ tự. Số lần chạy: 2

-5080, 1312, -160, -86, -50, -38, -19, 0, 1, 3, 13, 20, 24, 91, 219, 260, 1449529, -1612, 1990, 1413

i) Chuỗi số input theo thứ tự từ lớn đến bé. Số lần chạy: 55

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

f) Chuỗi số input đã gần đúng thứ tự. Số lần chạy: 90

-160, -5080, -86, -50, -38, -19, 0, 1, 3, 13, -1612, 20, 24, 91, 219, 260, 1312, 1413, 1990, 1449529

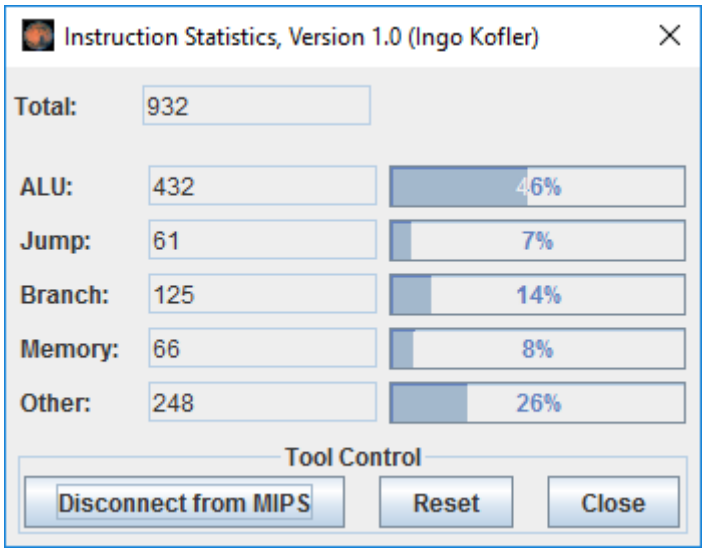
- Theo lý thuyết, số lần chạy tối đa (khi không kiểm tra chuỗi đã đúng thứ tự hay chưa sau mỗi lần chạy hàm partition()) đối với chuỗi input 20 phần tử là $20^2 = 400$

D. Tính thời gian chạy của chương trình:

Theo công thức:

$$\text{Time} = \text{Instruction Count} \times \text{CPI} \times \text{cycle time}$$

- Với mỗi chuỗi input khác nhau số lượng lệnh chạy sẽ khác nhau. Sinh viên dựa vào chức năng thống kê của chương trình giả lập để biết Instruction Count cụ thể.
- Ví dụ với chuỗi các phần tử bằng nhau, chương trình chạy 1 lần:



Số lượng lệnh: 932

Với giả thiết CPI = 1, clock rate = 2 GHz

Time = 932 * 1 / 2000000000 = 466 ns