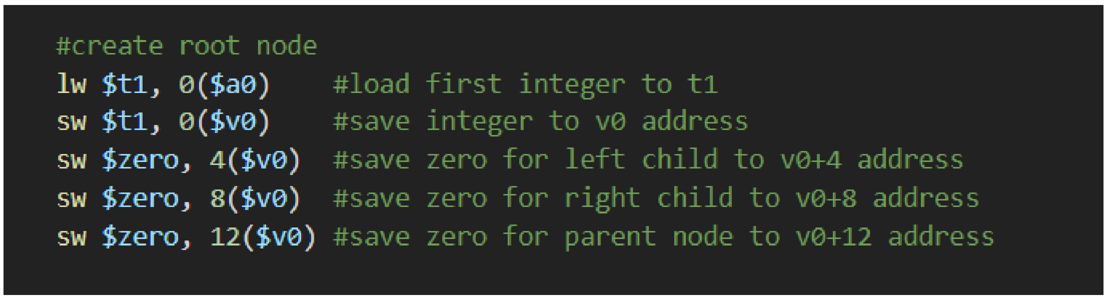
# REPORT

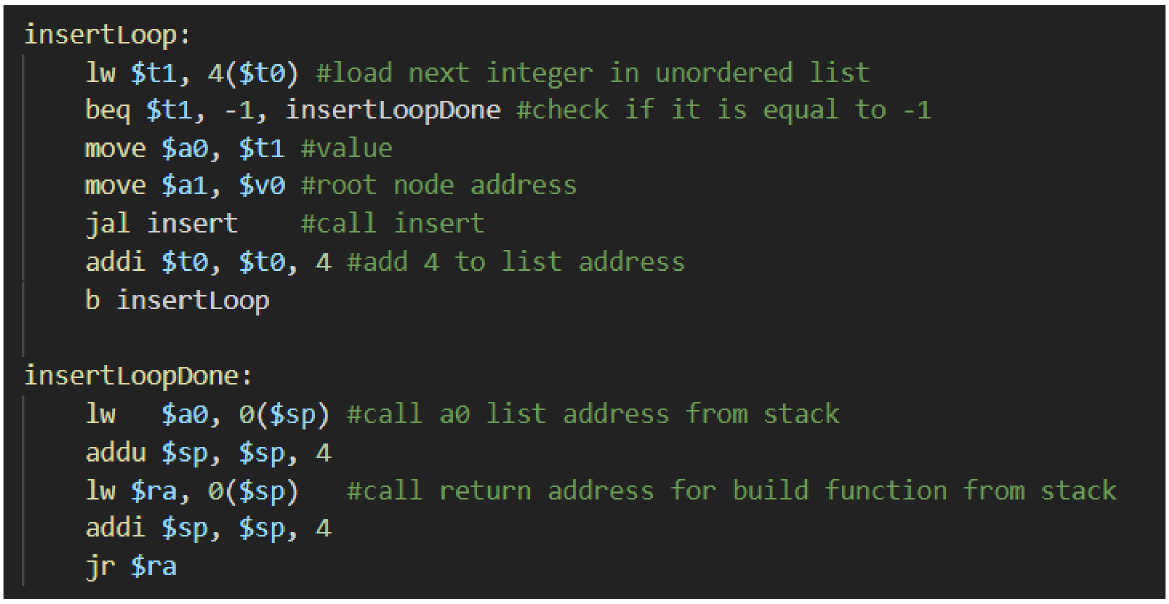
## Implementation Details

First, I created root node in build function, and put the value in list to the created address.



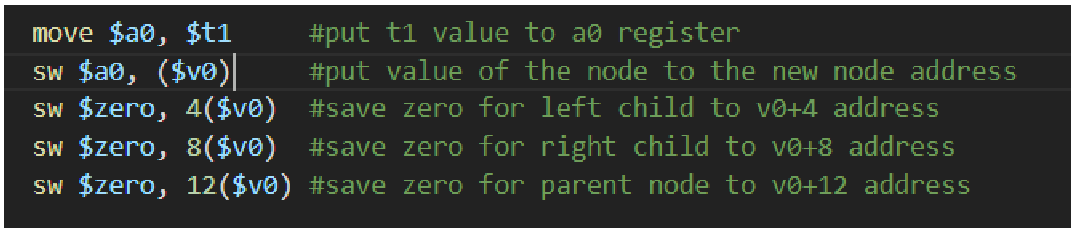
```
#create root node
lw $t1, 0($a0)    #load first integer to t1
sw $t1, 0($v0)    #save integer to v0 address
sw $zero, 4($v0)  #save zero for left child to v0+4 address
sw $zero, 8($v0)  #save zero for right child to v0+8 address
sw $zero, 12($v0) #save zero for parent node to v0+12 address
```

Then, in a loop, I inserted new values from list. I checked if the value is equal -1, if it is equal, then I stopped the loop. I called insert function for every values.



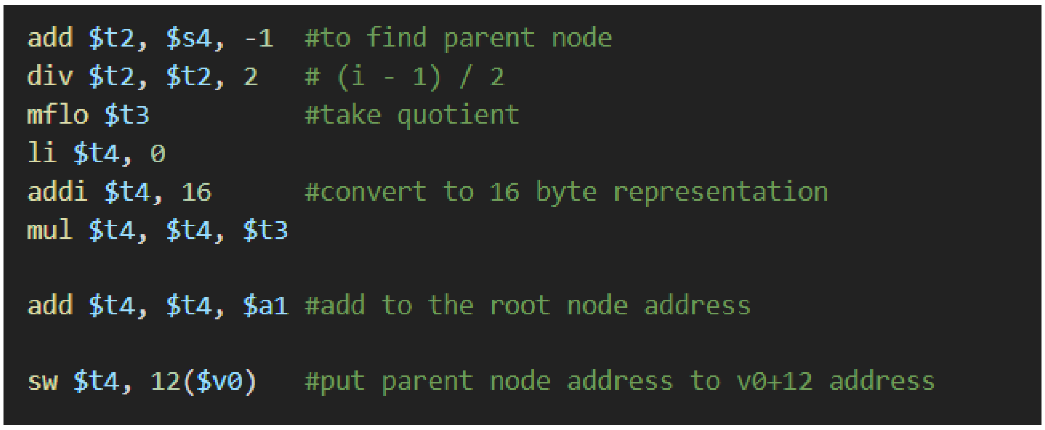
```
insertLoop:
    lw $t1, 4($t0) #load next integer in unordered list
    beq $t1, -1, insertLoopDone #check if it is equal to -1
    move $a0, $t1 #value
    move $a1, $v0 #root node address
    jal insert    #call insert
    addi $t0, $t0, 4 #add 4 to list address
    b insertLoop

insertLoopDone:
    lw   $a0, 0($sp) #call a0 list address from stack
    addu $sp, $sp, 4
    lw $ra, 0($sp)   #call return address for build function from stack
    addi $sp, $sp, 4
    jr $ra
```

In insert function, I created new node and put the value from a0 register to memory address of the new node.



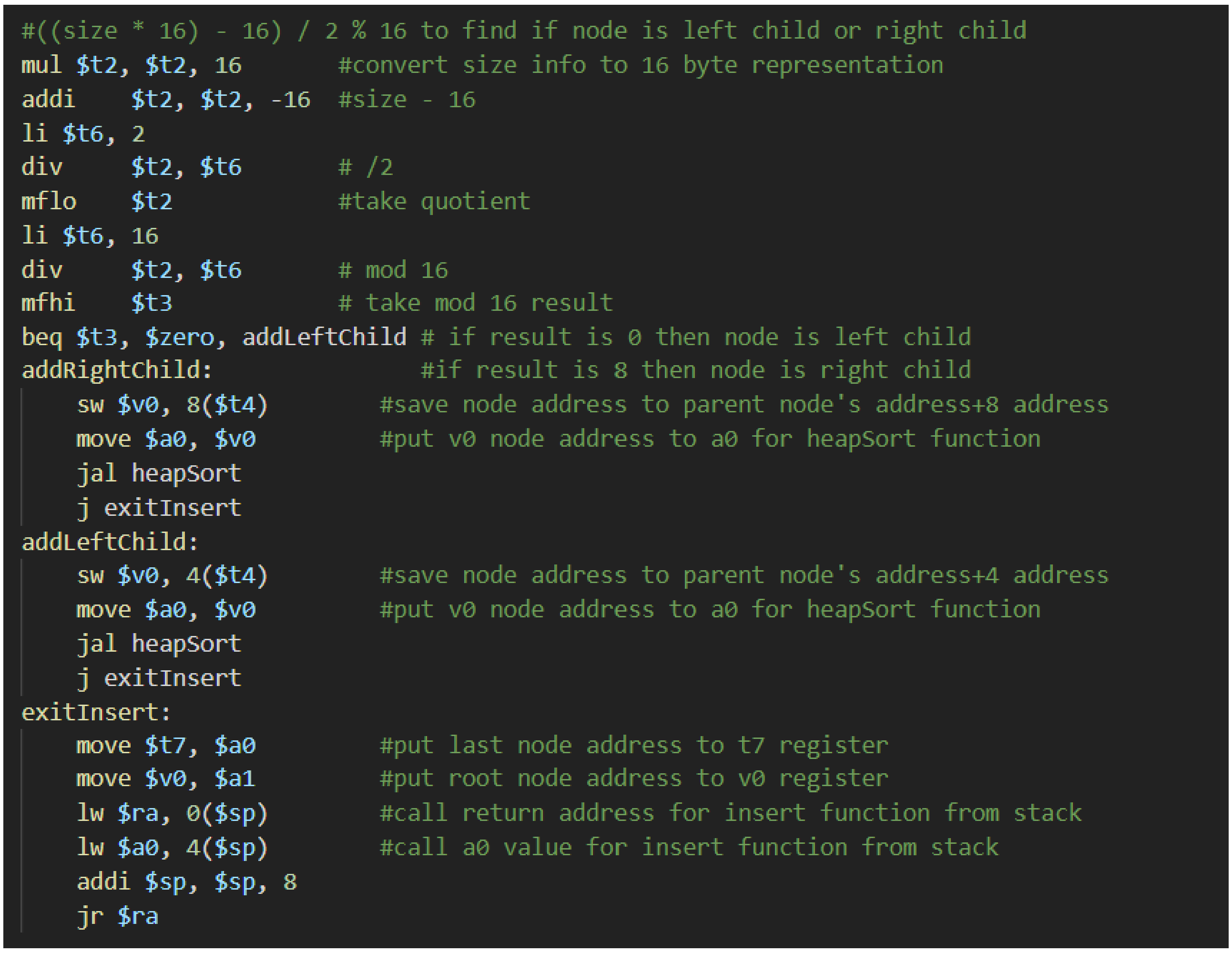
```
move $a0, $t1     #put t1 value to a0 register
sw $a0, ($v0)     #put value of the node to the new node address
sw $zero, 4($v0)  #save zero for left child to v0+4 address
sw $zero, 8($v0)  #save zero for right child to v0+8 address
sw $zero, 12($v0) #save zero for parent node to v0+12 address
```

After that, I did some calculations to find the parent node of this new node. I used "(i – 1) / 2" method to find parent. I multiplied the result of this method by 16 and I added this result to the root node address so I found parent node address. Then I stored parent node address to node  address + 12



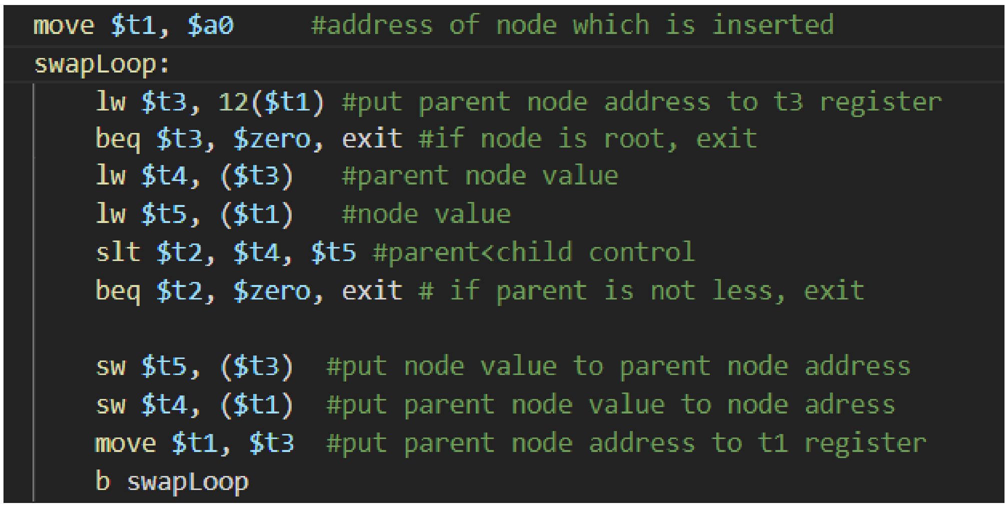
```
add $t2, $s4, -1  #to find parent node
div $t2, $t2, 2   # (i - 1) / 2
mflo $t3          #take quotient
li $t4, 0
addi $t4, 16      #convert to 16 byte representation
mul $t4, $t4, $t3

add $t4, $t4, $a1 #add to the root node address

sw $t4, 12($v0)   #put parent node address to v0+12 address
```

Then, to find if the new node is right or left child, there is another calculation. Formula is (((size * 16) -16) / 2) % 16 . First to represent the size in 16 byte, multiplied by 16,  then subtract 16 and divide by 2. If the mod 16 of result is equal to 8 then new node is right child, if it is equal to 0 then it is left child. I added new node address to parent node's address +4 or +8. The I called heapSort function. I stored last inserted node address in t7 register.



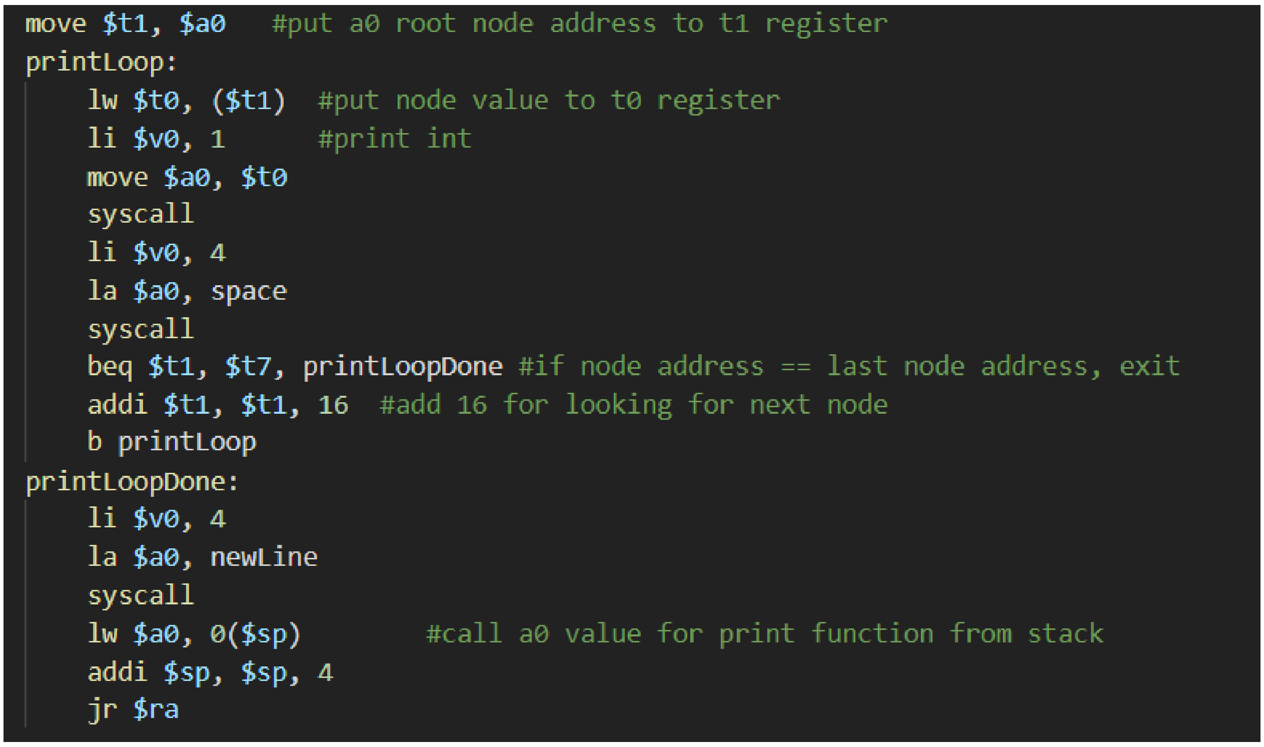
```
#((size * 16) - 16) / 2 % 16 to find if node is left child or right child
mul $t2, $t2, 16       #convert size info to 16 byte representation
addi    $t2, $t2, -16  #size - 16
li $t6, 2
div     $t2, $t6       # /2
mflo    $t2            #take quotient
li $t6, 16
div     $t2, $t6       # mod 16
mfhi    $t3            # take mod 16 result
beq $t3, $zero, addLeftChild # if result is 0 then node is left child
addRightChild:               #if result is 8 then node is right child
    sw $v0, 8($t4)        #save node address to parent node's address+8 address
    move $a0, $v0         #put v0 node address to a0 for heapSort function
    jal heapSort
    j exitInsert
addLeftChild:
    sw $v0, 4($t4)        #save node address to parent node's address+4 address
    move $a0, $v0         #put v0 node address to a0 for heapSort function
    jal heapSort
    j exitInsert
exitInsert:
    move $t7, $a0         #put last node address to t7 register
    move $v0, $a1         #put root node address to v0 register
    lw $ra, 0($sp)        #call return address for insert function from stack
    lw $a0, 4($sp)        #call a0 value for insert function from stack
    addi $sp, $sp, 8
    jr $ra
```

In heapSort function, I swapped parent node’s value and inserted node’s value. I checked if parent node address of inserted node is equal to zero, if it is then it must be root, I exited from the loop. Or parent node’s value is greater than inserted node’s value then exited from the loop.



```
move $t1, $a0     #address of node which is inserted
swapLoop:
    lw $t3, 12($t1) #put parent node address to t3 register
    beq $t3, $zero, exit #if node is root, exit
    lw $t4, ($t3)   #parent node value
    lw $t5, ($t1)   #node value
    slt $t2, $t4, $t5 #parent<child control
    beq $t2, $zero, exit # if parent is not less, exit

    sw $t5, ($t3)  #put node value to parent node address
    sw $t4, ($t1)  #put parent node value to node adress
    move $t1, $t3  #put parent node address to t1 register
    b swapLoop
```

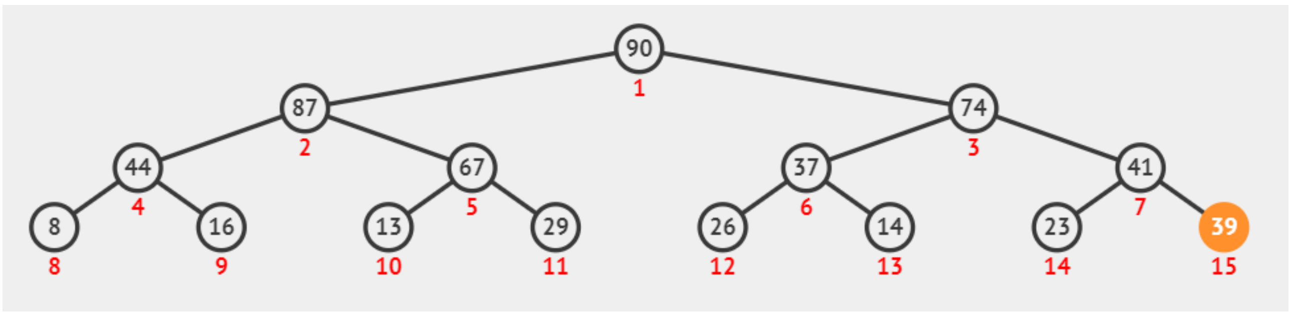
In print function, I printed root node and in loop I added 16 to root node address so I printed every node in breadth-first traversal. I checked if node address is equal to last node’s address, if it is then I exited from loop.



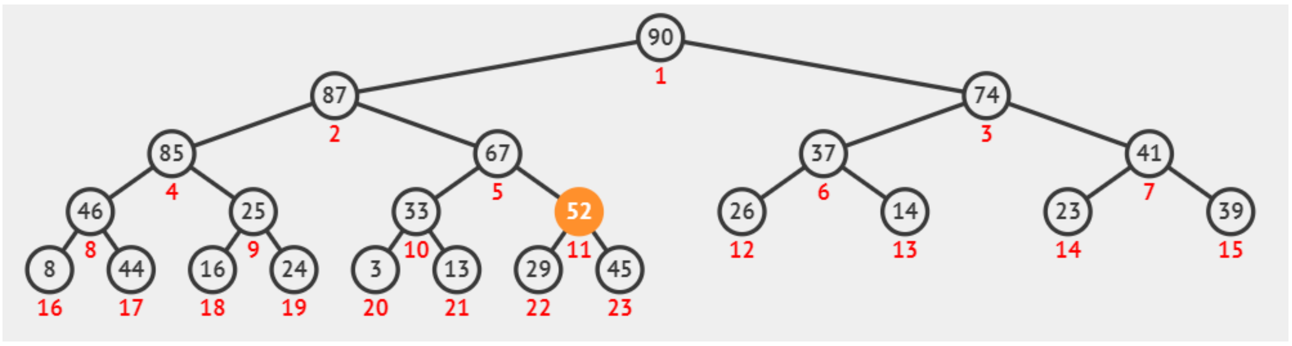
```
move $t1, $a0   #put a0 root node address to t1 register
printLoop:
    lw $t0, ($t1)  #put node value to t0 register
    li $v0, 1      #print int
    move $a0, $t0
    syscall
    li $v0, 4
    la $a0, space
    syscall
    beq $t1, $t7, printLoopDone #if node address == last node address, exit
    addi $t1, $t1, 16  #add 16 for looking for next node
    b printLoop
printLoopDone:
    li $v0, 4
    la $a0, newLine
    syscall
    lw $a0, 0($sp)        #call a0 value for print function from stack
    addi $sp, $sp, 4
    jr $ra
```

**Result**

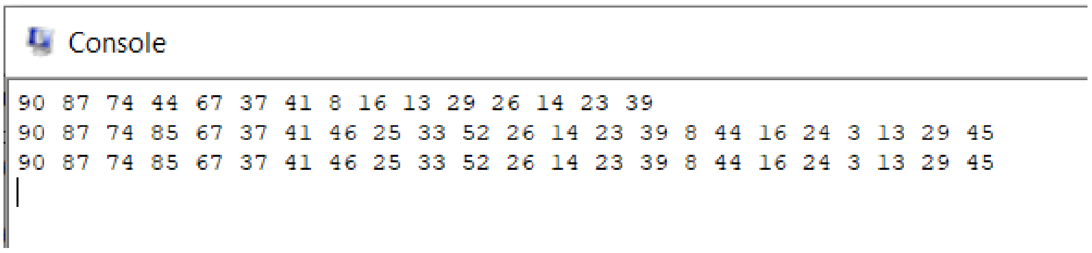
The tree have to be like this after build:



The tree have to be like this after insertions:



Output from program:



```
90 87 74 44 67 37 41 8 16 13 29 26 14 23 39
90 87 74 85 67 37 41 46 25 33 52 26 14 23 39 8 44 16 24 3 13 29 45
90 87 74 85 67 37 41 46 25 33 52 26 14 23 39 8 44 16 24 3 13 29 45
```

**Onur Cihangir**
**250201049**