

Computer Architecture HW2 Report

Bubble sort & Quick sort By SPIM

EE3 B03901156 Yu Xuan Huang

A. Bubble Sort

Design:

First, I divide the whole program into three parts completed by the respective function, including

- To get array's size and input integers from users.
- To bubble sort the input array.
- To output the sorted array.

The core of the design is according to the c++ pseudo code on the below:

```
1 void BubbleSort::Sort(int* array, int length)
2 {
3     for (int i = length - 1; i > 0; --i)
4         for (int j = 0; j < i; ++j)
5             if (array[j] > array[j + 1])
6                 swap(array[j], array[j + 1]);
7 }
```

And all the required explanations are illustrated as comments in the source code on the below:

```
1 .data
2
3 # define the require data
4 array: .space 1000 #leave enough spave for input data
5 newline: .asciiz "\n"
6 space: .asciiz " "
7 str1: .asciiz "Please enter the number of integers you want to sort: "
8 str2: .asciiz "Enter the integer: "
9 original: .asciiz "\nThe original array: "
10 sort: .asciiz "\nThe sorted array: "
11 switches: .asciiz "\n#switches: "
12
13
14 .text
15
16
17 main:
18     # a. To get array's size and input integers from users.
19     jal getSize
20     jal getInputIntergers
21     jal printOriginal
22     # b. To bubble sort the input array
23     jal bubbleSort
24     # c. To output the sorted array
25     jal printSorted
26     # The number of switches is used for debug
27     jal printNoOfSwitches
28     j exit
29
30 getSize:
31     # print "Enter the integer: " on console
32     li $v0, 4 # print string instruction
33     la $a0, str1 # load addr of str1 for syscall
34     syscall
35     # read the number of integers
36     li $v0, 5 # read integer and save in $v0
37     syscall
```

```

38
39     add $s0, $v0, $zero           # transfer the number into $s0
40     j Out
41
42     # $s0 : the number of integers
43
44     # for(i = 0; i< no of integers; i++){
45     #     A.push(i);
46     # }
47
48
49 getInputIntegers:
50     la $s2, array                 # $s2 = array address
51     add $t0, $zero, $zero         # set index i = 0 ($t0)
52     add $t1, $s2, $zero           # $t1 = array address (tmp)
53
54     # print "Please enter the number of integers you want to sort: " on console
55 prompt: li $v0, 4                 # print string instruction
56         la $a0, str2
57         syscall
58         li $v0, 5                 # read integer and save in $v0
59         syscall
60         sw $v0, ($t1)             # store the input value
61         addi $t1, $t1, 4          # remove the current address to next array cell
62         addi $t0, $t0, 1          # increase index i, i++
63         beq $t0, $s0, Out        # break loop if all the integers are obtained
64         j prompt
65
66     # while(i<numberOfIntegers){
67     #     print(A[i]);
68     #     i++;
69     # }
70
71
72 printInteger:
73     add $t0, $zero, $zero         # set index i = 0 ($t0)
74     add $t1, $a0, $zero          # $t1 = array pointer
75 loop: lw $t2, ($t1)              # load the integer into $t2
76         add $a0, $t2, $zero       # print
77         li $v0, 1
78         syscall
79         li $v0, 4                 # print string instruction
80         la $a0, space             # print space
81         syscall
82         addi $t1, $t1, 4          # increment pointer to next array cell
83         addi $t0, $t0, 1          # increment index i
84         bne $t0, $s0, loop        # if index i < numberOfIntegers
85         j Out
86
87 # s0 : number of integers
88 # s2 : address of first value of array
89 # s3 : save the number of switches
90 # t0 : counter for outer loop
91 # t1 : counter for inner loop
92 # t2 : n-1
93 # t3 :
94 # t4 : n-c-1
95 # t5 : A[d]
96 # t6 : A[d+1]
97
98 #####
99 # void BubbleSort::Sort(int* array, int length) #
100 # { #
101 #     for (int i = length - 1; i > 0; --i) #
102 #         for (int j = 0; j < i; ++j) #
103 #             if (array[j] > array[j + 1]) #
104 #                 swap(array[j], array[j + 1]); #
105 # } #
106 #####
107
108
109 bubbleSort:
110     li $t0, 0                    # t0 = counter for outer loop (initialized:0) = c
111 OuterLoop:

```

```

112         li $t1, 0                # t1 = counter for inner loop (initialized:0) = j
113         add $t2, $zero, $zero
114         li $t2, -1
115         add $t2, $s0, $t2        # t2 = n-1
116         slt $t7, $t0, $t2        # if( t0 < n-1 )
117         beq $t7, $zero, Out
118     InnerLoop:
119         sub $t4, $t2, $t0        # t4 = n - c -1
120         slt $t3, $t1, $t4        # if( t1=j < n-c-1 )
121         beq $t3, $zero, incT1ctr # if( t1 = n-c-1 ), jump out of inner loop
122         sll $t3, $t1, 2
123         add $t3, $t3, $s2
124         lw $t5, 0($t3)           # save A[j]
125         lw $t6, 4($t3)           # save A[j+1]
126         bgt $t6, $t5, no_swap    # if(A[j] <= A[j+1])
127         # if if(A[j] > A[j+1]) :
128         addi $s3, $s3, 1         # update $s3 which tracks number of Swaps used in the sortIntegers funtion
129         sw $t6, 0($t3)           # swap
130         sw $t5, 4($t3)
131     no_swap:
132         add $t1, $t1, 1         #increment counter of inner loop
133         j InnerLoop
134     incT1ctr:
135         add $t0, $t0, 1         # increment counter $t0 for outer loop
136         beq $t3, $zero, OuterLoop
137         j Out
138
139     printSorted:
140         li $v0, 4
141         la $a0, sort
142         syscall
143         add $a0, $s2, $zero      # load the integers into $a0
144         j printInteger
145
146     printOriginal:
147         li $v0, 4
148         la $a0, original
149         syscall
150         add $a0, $s2, $zero      # load the integers into $a0
151         j printInteger
152
153     printNoOfSwitches:
154         li $v0, 4
155         la $a0, switches
156         syscall
157         add $a0, $s3, $zero      # load the integers into $a0
158         li $v0, 1
159         syscall
160         j pOut
161
162     Out:
163         jr $ra                  # return to the original address
164     exit:
165         li $v0, 10
166         syscall

```

Result:

```

Console
Please enter the number of integers you want to sort: 10
Enter the integer: -1
Enter the integer: 3
Enter the integer: -5
Enter the integer: 7
Enter the integer: -9
Enter the integer: 2
Enter the integer: -4
Enter the integer: 6
Enter the integer: -8
Enter the integer: 10

The original list: -1 3 -5 7 -9 2 -4 6 -8 10
The sorted list: -9 -8 -5 -4 -1 2 3 6 7 10
#switches: 20

```

B. Quick Sort

Design:

First, I divide the whole program into three parts completed by the respective function, including

- To get array's size and input integers from users.
- To quick sort the input array.
- To output the sorted array.

```
1  Quicksort(A,p,r)
2  {
3      if(p<r)
4      {
5          q = Partition(A,p,r);
6          quick_sort(A,p,q-1);
7          quick_sort(A,q+1,r);
8      }
9  }
10
11 Partition(A,p,r)
12 {
13     i = p -1 ;
14     for (j = p; j < r; ++j)
15     {
16         if (A[j] <= A[r])
17         {
18             swap(A[i],A[j]);
19         }
20     }
21     i++;
22     swap(A[i],A[r]);
23     return i;
24 }
```

The core of the design is according to the c++ pseudo code on the below:

```
1  .data
2
3  array: .word 0 : 1000
4  comma: .asciiz ", "
5  newline: .asciiz "\n"
6  space: .asciiz " "
7  str1: .asciiz "Please enter the number of integers you want to sort: "
8  str2: .asciiz "Enter the integer: "
9  original: .asciiz "\nThe original array: "
10 sort: .asciiz "\nThe sorted array: "
11 switches: .asciiz "\n#switches: "
12 size: .word 0 #size of the actual array
13
14 .text
15
16 .globl main
17 main:
18     getSize:
19         # print str1 on console
20         li $v0, 4
21         la $a0, str1
22         syscall
23         # read the number of integers
24         li $v0, 5
25         syscall
26         la $t0, size
27         sw $v0, 0($t0)
28         # transfer the number to $t0
29
30     getInputIntegers:
31         la $t0, array
32         lw $t1, size
33         li $t2, 0
34         # $t0 = array address
35         # load size in $t1
36         # set index i = 0 ($t2)
37
38     prompt:
39         bge $t2, $t1, finishReadInput
40         # break loop if all the integers are obtained(while($t2<$t1))
```

```

38         li $v0, 4                                # print string instruction
39         la $a0, str2
40         syscall
41         li $v0, 5                                # read integer and save in $v0
42         syscall
43         sw $v0, 0($t0)                            # store value
44         addi $t0, $t0, 4                          # remove the current address to next array cell
45         addi $t2, $t2, 1                          # increase index i, i++
46         j prompt
47
48     finishReadInput:
49         li $v0, 4
50         la $a0, original
51         syscall
52
53         jal Print
54
55     # $a0 = addr of array
56     # $a1 = zero
57     # $a2 = size -1
58
59     #####
60     # pseudo code for c:
61     #
62     # Quicksort(A,p,r)
63     # {
64     #     if(p<r)
65     #     {
66     #         q = Partition(A,p,r);
67     #         quick_sort(A,p,q-1);
68     #         quick_sort(A,q+1,r);
69     #     }
70     # }
71     #
72     # Partition(A,p,r)
73     # {
74     #     i = p -1 ;
75     #     for (j = p; j < i; ++j)
76     #     {
77     #         if (A[j] <= A[r])
78     #         {
79     #             swap(A[i],A[j]);
80     #         }
81     #     }
82     #     i++;
83     #     swap(A[i+1],A[r]);
84     #     return i;
85     # }
86     #####
87
88     la $a0, array                                # points $a0 = addr of array
89     li $a1, 0                                    # left value
90     lw $a2, size                                 # right value
91     addi $a2, $a2, -1
92     jal QuickSort
93
94     li $v0, 4
95     la $a0, sort
96     syscall
97
98     jal Print
99     j exit
100
101
102 Print:
103
104
105     la $t0, array                                # load the array addr in $t0
106     lw $t1, size                                 # load size to $t1
107     li $t2, 0                                    # load 0 (index) to $t2
108
109     loopOfPrint:
110         bge $t2, $t1, PrintEnd                    # while($t2<$t1)
111         li $v0, 1

```

```

112     lw $a0, 0($t0)
113     syscall
114
115     li $v0, 4                # print string instruction
116     la $a0, space           # print space
117     syscall
118
119     addi $t0, $t0, 4         # increment pointer to next array cell
120     addi $t2, $t2, 1         # increment index
121
122     j loopOfPrint
123
124 PrintEnd:
125     jr $ra
126
127
128 # addr of array : $a0
129 # p: $a1
130 # r: $a2
131 # i: $t2
132 # j: $t3
133 # pivot: $t4
134
135
136
137 swap:
138     sll $t1, $a1, 2          # $t1 = $a1 (t) * 4
139     add $t1, $a0, $t1        # $t1 = addr of arr[t]
140     lw $t0, 0($t1)          # tmp = v[t]
141     lw $t2, 4($t1)          # $t2 = v[t+1]
142     sw $t0, 4($t1)          # v[t+1] = tmp
143     sw $t2, 0($t1)          # v[t] = v[t+1]
144     jr $ra
145
146 partition:
147     add $t1, $a2, $zero      # move $a2 into $t1
148     sll $t1, $t1, 2          # $t1 = $t1 * 4
149     add $t1, $a0, $t1        # $t1 = A[r]'s addr
150     addi $t2, $a1, -1        # i(=$t2) = p(=$a1)-1
151     add $t3, $a1, $zero      # j(=$t3) = p(=$a1)
152     lw $t4, 0($t1)          # pivot = A[r] = $t4
153
154 OutLoop:
155     slt $t0, $t3, $a2        # $t0 = (j<r)
156     beq $t0, $zero, endPartition
157     add $t5, $t3, $zero      # $t5 = copy j
158     sll $t5, $t5, 2          # $t5 = $t5 * 4
159     add $t5, $a0, $t5        # $t5 = addr of A[j]
160     lw $t6, 0($t5)          # $t6 = A[j]
161     slt $t0, $t4, $t6        # if(pivot<A[j])
162     bne $t0, $zero, noSwap
163     addi $t2, $t2, 1          # i++
164     add $t7, $t2, $zero      # $t7 = copy i
165     sll $t7, $t7, 2          # $t7 = $t7*4
166     add $t7, $a0, $t7        # $t7 = addr of A[i]
167     lw $t8, 0($t7)          # $t8 = A[i]
168     sw $t6, 0($t7)          # A[i] = A[j]
169     sw $t8, 0($t5)          # A[j] = A[i]
170     j OutLoop               # j++
171
172 endPartition:
173     addi $t2, $t2, 1          # i++
174     add $t5, $t2, $zero      # $t5 = copy i
175     sll $t5, $t5, 2          # $t5 = $t5*4
176     add $t5, $a0, $t5        # $t5 = addr of A[i+1]
177     lw $t7, 0($t1)          # $t7 = A[r]
178     lw $t6, 0($t5)          # $t6 = A[i+1]
179     sw $t6, 0($t1)          # swap
180     sw $t7, 0($t5)
181     add $v0, $t2, $zero      # $v0 = copy i
182     jr $ra
183
184 QuickSort:
185     slt $t0, $a1, $a2        # $t0 = (p<r)
186     beq $t0, $zero, endQuickSort

```

```

186      addi $sp, $sp, -12                # stack
187      sw $ra, 8($sp)
188      sw $a1, 4($sp)
189      sw $a2, 0($sp)
190      jal partition
191
192      addi $sp, $sp, -4                  # make room in stack for v0
193      sw $v0, 0($sp)
194      add $t1, $v0, $zero
195      addi $t1, $t1, -1                  # q-=1
196      add $a2, $t1, $zero                # $a2 = q
197      jal QuickSort
198
199      lw $v0, 0($sp)                    # restore $v0
200      lw $a2, 4($sp)                    # restore $a2
201      add $t1, $v0, $zero                # $t1 = copy $v0
202      addi $t1, $t1, 1                   # q+=1
203      add $a1, $t1, $zero                # $a1 = q
204      jal QuickSort
205
206      lw $v0, 0($sp)                    # restore $v0
207      lw $a2, 4($sp)                    # restore $a2
208      lw $a1, 8($sp)                    # restore $a1
209      lw $ra, 12($sp)                   # restore $ra
210      addi $sp, $sp, 16                  # return the stack
211 endQuickSort:
212      jr $ra
213
214 exit: li $v0, 10
215      syscall

```

Result:

```

Console
Please enter the number of integers you want to sort: 10
Enter the integer: -1
Enter the integer: 3
Enter the integer: -5
Enter the integer: 7
Enter the integer: -9
Enter the integer: 2
Enter the integer: -4
Enter the integer: 6
Enter the integer: -8
Enter the integer: 10

The original array: -1 3 -5 7 -9 2 -4 6 -8 10
The sorted array: -9 -8 -5 -4 -1 2 3 6 7 10

```