DEPARTMENT OF MATHEMATICS
AND COMPUTER SCIENCE

Computer Architecture Project

Gabriel Howard Jadderson : gajad16@student.sdu.dk
<EMPTY> : <EMPTY>

DM548
November 10, 2017

Contents

List of Figures

1 Introduction and Design

Our task is to write a sorting program in assembly

```
i ← 1
while i < length(A)
    x ← A[i]
    j ← i - 1
    while j >= 0 and A[j] > x
        A[j+1] ← A[j]
        j ← j - 1
    end while
    A[j+1] ← x
    i ← i + 1
end while
```

Figure 1: psuedocode of insertionsort

2 Testing

Our unsorted numbers are generated in the following way

```
1 for i in {1..X}; do echo $RANDOM; done > test.dat
```

where x denotes the amount of numbers we want to have generated. we've created 6 data-sets: 100 1.000 5.000 10.000 25.000 50.000 furthermore, each data-set was generated 10 times in order to achieve 10 different test-candidates for each data-set. Our test show that the program has passed all 6 data-set tests and completed all 60 tests as expected.
The following is the output of our test data

$test - candidate$	1	2	3	4	5	6	7	8	9	10
100_{time}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
100_{mcips}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
1.000_{time}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
1.000_{mcips}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
5.000_{time}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
5.000_{mcips}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
10.000_{time}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
10.000_{mcips}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
25.000_{time}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
25.000_{mcips}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
50.000_{time}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}
50.000_{mcips}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}	0.003_{sec}

3 Conclusion

We present an assembly program which utilizes insertionsort to sort up to 16-bit numbers. From the testing results we can safely assume that the program works as expected and behaves accordingly, based on the times we can easily see that the algorithm runs in $O(n^2)$ time. our challenges were to implement memory correctly and to access them correctly in the in the heap, and unfortunately the lack of resources online on assembly didn't help either.