Homework 1 - Introduction to Algorithms

Edwing Alexis Casillas Valencia

September 11, 2025

Abstract

In this homework, we review the Selection algorithm and the complexity calculation

PROBLEM 1

Build a program in C that gets an integer in the shell and reverse the integer. For example, if you get 1234 at the shell you should return 4321. Here, we ask you to do the following:

1) Get the integer number from the shell - and see if the number given is really a integer and it is positive.
2) Look at the integer as a binary
3) Reverse the bits
4) Return the reversed number
    a) You can use the idea of masks and $<<$
5) Explain why it works
6) Calculate its complexity by counting the steps

Answer:

First I request the user to type an integer number into the shell, it doesn't matter that the user enters a negative number at this point.

Once I get the number, I convert this number into it's binary representation, for this, I use a mask of 32 bits (value of an int on C language) with 1 at the first position, then I do a bit to bit AND with the number I got from the user. C automatically storage and interprets the numbers in binary, that's why bit to bit operations are easy to implement. The results at every right shift are storage on an array.

Due I storage the number in 32 bits, the first digit tells me if the number is positive or negative.

- 1 for negative
- 0 for positive

If the number is negative, I just need to apply the two's complement. The way I did it was to find the first 1 of the LSB part, and from there leave all the 0's; on the rest of the number I applied again the same bit to bit operation, but with a NAND, this permits me change the 0's to 1's and vice versa. Once it's done the process, I show the binary representation to the user.

To reverse the number, I need to get the last digit from the given number and storage in the result variable, then multiply it by 10 (for number with +2 digits), and add the following digit applying the same logic.

I use the following steps on the script:

1) Initialize the variable I'll use on 0.
2) Multiply it by 10. Due binary uses multiples of 2, we can separate the *10 multiplication into one *8 and one *2 and apply the addition on them using left shift ($<<3$ and $<<1$). This is because mathematically, a multiplication are series of additions.
3) Get the module of the number dividing by 10.
4) Do the addition of the module with the result number.
5) Get the floor of the given number dividing by 10.
6) Repeat steps 2-5 until the floor is 0.

This permits us reverse the number using binary operations.

$$O(x+6+x+6+8+x+1+6+x+x+1+n+8) \tag{1}$$

$$O(n+5x+36) \tag{2}$$

Were n are the n-cycles and x-constant cycles. Due I'm using all the bits that storage an int on C (32), every x-cycle is going to be executed 32 times. Substituting, we get the following complexity

$$O(n+196) \tag{3}$$

Note: I'm attaching the the C file on the homework assignment

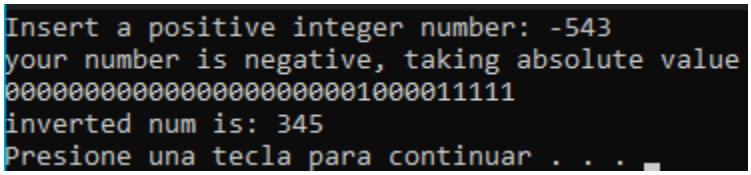

Fig. 1. Shell

PROBLEM 2

- Algorithm steps:
  1) Initialization: The array is conceptually divided into two parts: a sorted subarray (initially empty) and an unsorted subarray (initially the entire array).
  2) Iteration: For each pass through the unsorted subarray (from the first element to the second-to-last):
     a) Find Minimum: Locate the smallest element within the current unsorted subarray.
     b) Swap: Exchange this smallest element with the first element of the unsorted subarray.
     c) Boundary Shift: The boundary between the sorted and unsorted subarrays shifts one position to the right, incorporating the newly sorted element.
  3) Termination: This process continues until the entire array is sorted.

1) Use a single main.c file for this
2) compile this
3) Use the cont method to get me the complexity.

Answer:

First I initialize an array of n elements and I print it on the shell for the user. Once it is done, I calculate the length of the array dividing the bytes used to storage the array by the bytes of the first element.

Then I initialize a counter to travel around a "for" loop, at each loop, the minimum value is going to be updated by the value of the array that we're comparing. Inside that loop I initialize another loop, this one will compare the value of the i+1 position on the array with the minimum value of the array. If the i+1 value is less than the minimum value, it will be now the new minimum value.

Once I have the minimum value of the unsorted array, I storage the first value of it on a temporal variable. Then, I replace the first value of the array with the minimum and on the next position I insert the value of the temporary variable. I repeat this process n times, were n is the length of the array.

Once the array is sorted, I print it on the shell for the user.

The complexity is:

$$O(7 + 1 + n + n * n + 7 + 1 + n + 3) \tag{4}$$

$$O(n^2 + 2n + 19) \tag{5}$$

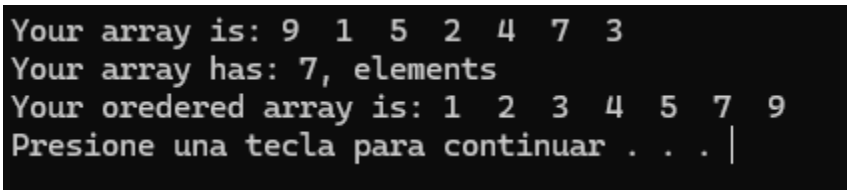

Fig. 2. Selection algorithm

Note: I'm attaching the the C file on the homework assignment