IDMA 2026
Problem set 1

Abdulkareem Al-Rifai, kbt185

16-02-26

Contents

1 Different representations of integers

1.a Write the binary number $(110)_2$ in decimal notation

Binary notation refers to the number system of base 2, where as the decimal notation refers to base 10. I will start by expanding $(110)_2 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 6$, now I will try to write the number 6 in decimal notation. To do that I will use the equation

$$m = q_0 \cdot b + r_0 ...$$

$$...m = q_n \cdot b + r_n$$

where : m : is the number we are trying to rewrite, b : the base, we continue running the recursion process until $q_n = 0$

$$6 = 0 \cdot 10 + 6$$

since $q_0 = 0$ we stop and now we have

$$(110)_2 = (6)_{10}$$

1.b Write the decimal number 110 in binary notation

Following the steps of the previous question, we have

$$(110)_{10} = 1 \cdot 10^2 + 1 \cdot 10^1 + 0 \cdot 10^0 = 110$$

to convert the number 110 to base 2 we use the equation :

$$110 = 55 \cdot 2 + 0$$

$$55 = 27 \cdot 2 + 1$$

$$27 = 13 \cdot 2 + 1$$

$$13 = 6 \cdot 2 + 1$$

$$6 = 3 * 2 + 0$$

$$3 = 1 \cdot 2 + 1$$

$$1 = 0 \cdot 2 + 1$$

Then we have that

$$(110)_{10} = (1101110)_2$$

1.c Write the octal number $(110)_8$ in decimal notation

$$(110)_8 = 1 \cdot 8^2 + 1 \cdot 8^1 + 0 \cdot 8^0 = 64 + 8 + 0 = 72$$

$$72 = 7 \cdot 10 + 2$$

$$7 = 0 \cdot 10 + 7$$

hence

$$(110)_8 = (72)_{10}$$

2 Find the greates common divisor using the algorithm we learned, show the steps then express d as linear combination of m and n

To solve this problem we will use the following equation, which was explained in problem 1

$$m = q * n + r \tag{1}$$

2.a m=38, n=14

$$38 = q_0 \cdot 14 + r \rightarrow q = 2, r = 10 \tag{2}$$
$$14 = q_1 \cdot 10 + r \rightarrow q_1 = 1, r = 4 \tag{3}$$
$$10 = q_3 \cdot 4 + r \rightarrow q_3 = 2, r = 2 \tag{4}$$
$$4 = q_4 * 2 + r \rightarrow q_4 = 2, r = 0 \tag{5}$$

The gcd is the last remainder that is not 0. To write it in linear combination we have to solve for the following :

$$2 = x \cdot 14 + y \cdot 38 \ldots \tag{6}$$

2.b m=117, n=69

$$117 = q_0 * 69 + r \tag{7}$$

3 Consider the following algorithm

```
j := 1
while (j<=n) {
        A[j] := 0
        for i := j downto 1 {
                A[j] := A[j]+i*i
        }
        j := j+1
        }
```

3.a Explain what the algorithm does, and what are the number stored and computed in A

Let start with the easier part, what are the values stored in array A, from j to n. The values stored in A are the sum of squares of j downto 1, meaning that for A[2], A[2] is equal to $2^2 + 1^2 = 5$.

j is an index variable showing our position in the array. This algorithm has 2 loops, the outer while-loop runs n times, going through every element in the array. The inner for-loop, that also run n times, is used to update the values of the array from j to n. Initially A[j:n] = 0, a condition/ command specified at the begining of the outer-loop. The inner for-loop takes A[j] and iterates throught it i times, until i hits 1. At each iteration in the for loop, the value of A[j] is updated, then the updated value is used for the next iteration, once i hits 1, the for-loop terminates, and j is updated to j+1. To reduce abstraction, suppose that n=3 and execute the algorithm at j=3 manually. j=3

$$A[3] = 0 \rightarrow j \leq n(true, continue) \tag{8}$$

we enter for-loop

$$A[3] = 0 + 3 * 3 = 9 \rightarrow A[3'] = A[3] + 2 * 2 = 13 \rightarrow A[3''] = A[3'] + 1 * 1 = 13 + 1 = 14 \tag{9}$$

$$i = 1 \textit{(for-oop terminates)} \rightarrow j = 3 + 1 = 4 \rightarrow 4 \leq 3 \textit{(j larger than, outer-loop terminates)} \tag{10}$$

3.b Provide an asymptotic analysis of the running time as function of the array size n

We can analyse the running time of the algorithm by looking at the inner and outer loop. Assuming that every operation takes a constant time c, we can avoid detailed analysis and look at the recursive operations. The outer-loop will run n-times, and for each j in the while loop run, the for loop runs j times. This means that if j = 3, the for loop runs 3 times, j=n, the for-loop with run n-times.The complexity of this such a relationship is n*n = n^2 . Asymptotic analysis refers to the behavior of the algorithm as n grows to infinity.

$$O(n^2) \tag{11}$$

3.c Improve the code to run faster while retaining the same functionality, analyse the time complexity

This algorithm should run faster considering that it only has one loop that runs n times, rather than a nested loop running.

```
A[1:n]
for i = 2 up to n :
        if i <= n :
                A[i] = i*i +A[i-1]
        i = i+1
```

4

```
1 j := n
2 good := TRUE
3 while (j >= 1 and good) {
4    s := A[j]
5    for i := j - 1 downto 1 {
6        s := s + A[i]
7    }
8    s := s / j
9    if (s > B[j]) {
10        good := FALSE
11    }
12    else {
13        j := j - 1
    }
}
14 return good
```

4.a Explain what the algorithm does, when does it return TRUE or FALSE and why

Let divide the code into smaller pieces then explain each part individually. First we pay attention that array B and A have the same length n.

Lines 1- 6 : the length of the arrays is assigned to the variable j, and the variable good is assigned a boolean value : TRUE. The outer while loop keeps running until $j < 1$ or good =false. Variable s is used to save the sum of all the elements in array A. This happens using the inner for loop that iterates through each element in A and sum them, then updat the value of s to include the sum before it.

Lines 8 : After computing the sum of array A, line 8 divide the sum by j : len.(A) which is the average of the values in the array.

Line 9-13: In this if-else statement, we compare the average of all the elements in array A with the last element in array B, if s is larger than B[j] good becomes false and the code returns FALSE. if that is not the case, the else statement is run, j beomes n-1, and the while loop runs one more time computing a new average (the new average is the average of n-1 elements in A)

If s (for j=n) is larger than B[j], then the algorithm returns TRUE. The function returns false if all s' (average) [for j=n to 1] are smaller than B[j:1]. Remember that B[j-1] is compared with average.A[1:j-1] and B[j-2] against averae.A[1:j-2] and so on.

4.b Provide an asymptotic analysis of the running time as function of the array size n

The worst case scenario can be charectrized by the situatuation where every element in B[1:n] is larger than every average computed for A[n:1] (I hope the notation here makes sense). In that scenario, the outter while-loop will run n times, the inner for-loop will run n times. The if-else statement is also run n times, but it is not really a computation , but just a comparison step which takes a constant time, hence it is ignored. Therefor we can say that the Big O for this algorithm is

$$O(n^2) \tag{12}$$

4.c Improve the code to run faster while retatining the same functionality. Analyse it.

The straight forward way to improving an algorithm is by reducing the number of loops (recursive functions), and splitting the problem into smaller sub-problems that takes constant time to solve.

We could split each array in half, compute the averages of the 4 sub-arrays then compare them.

.....