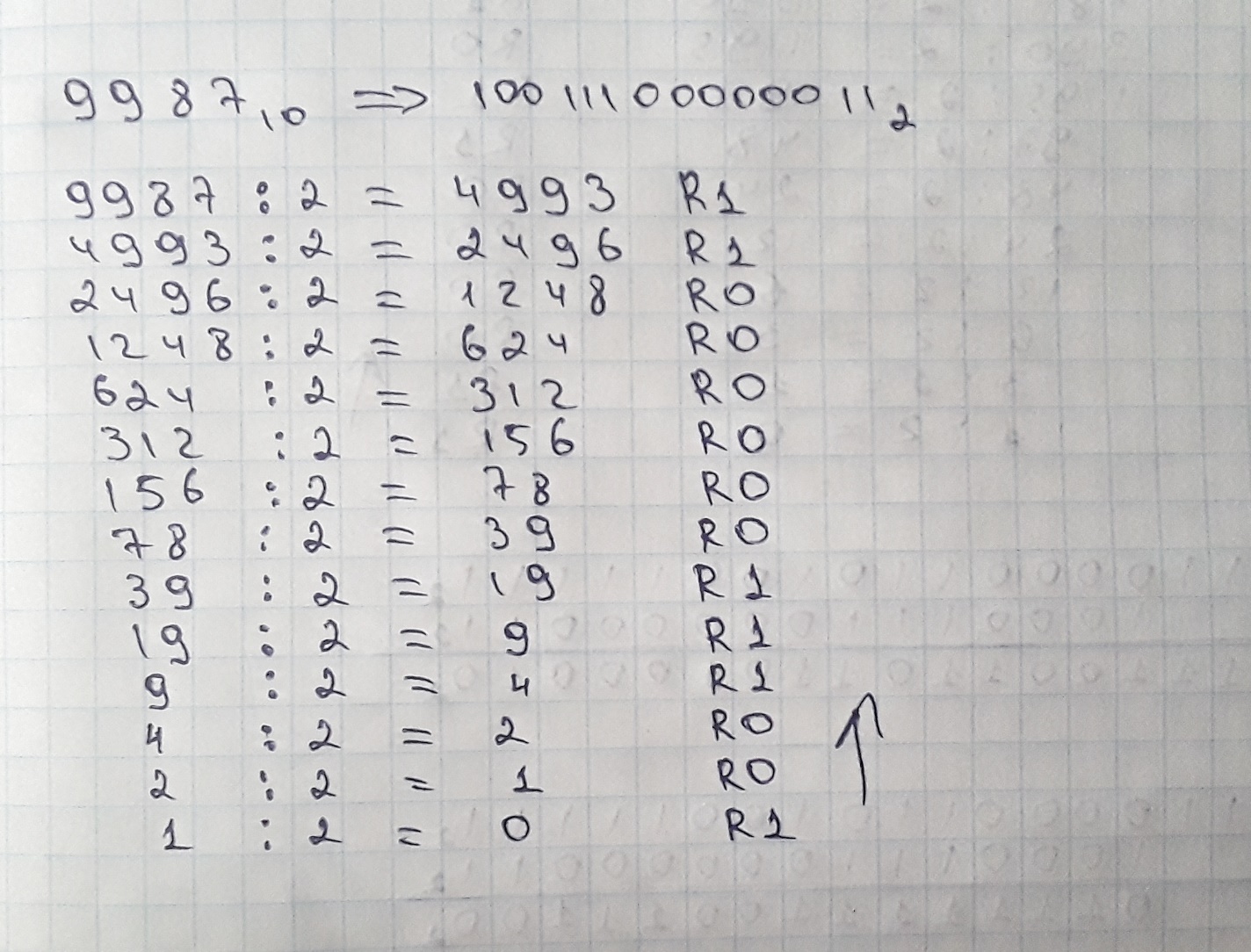
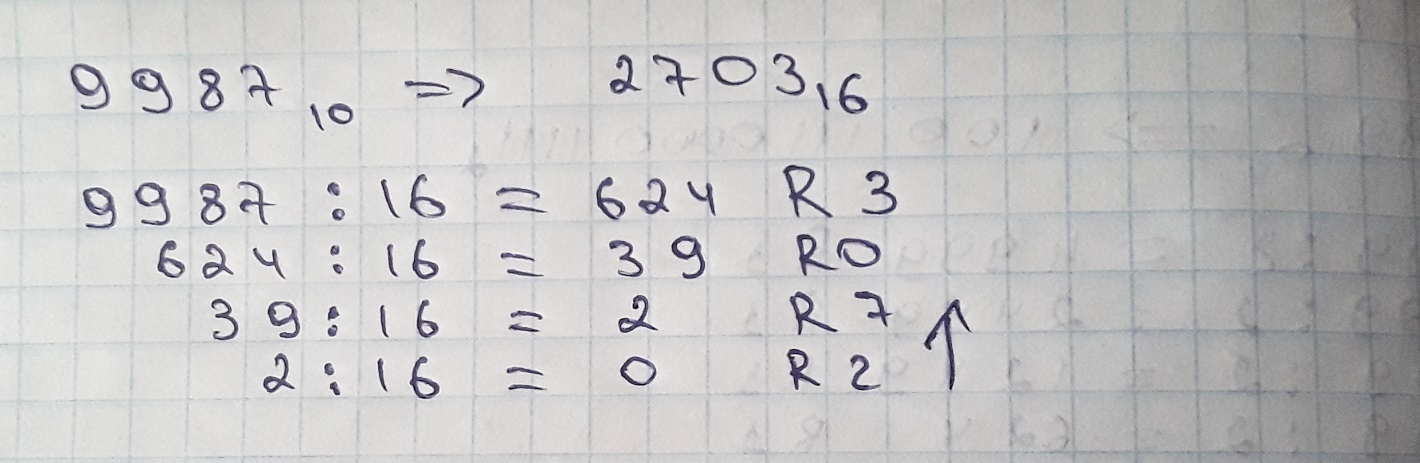
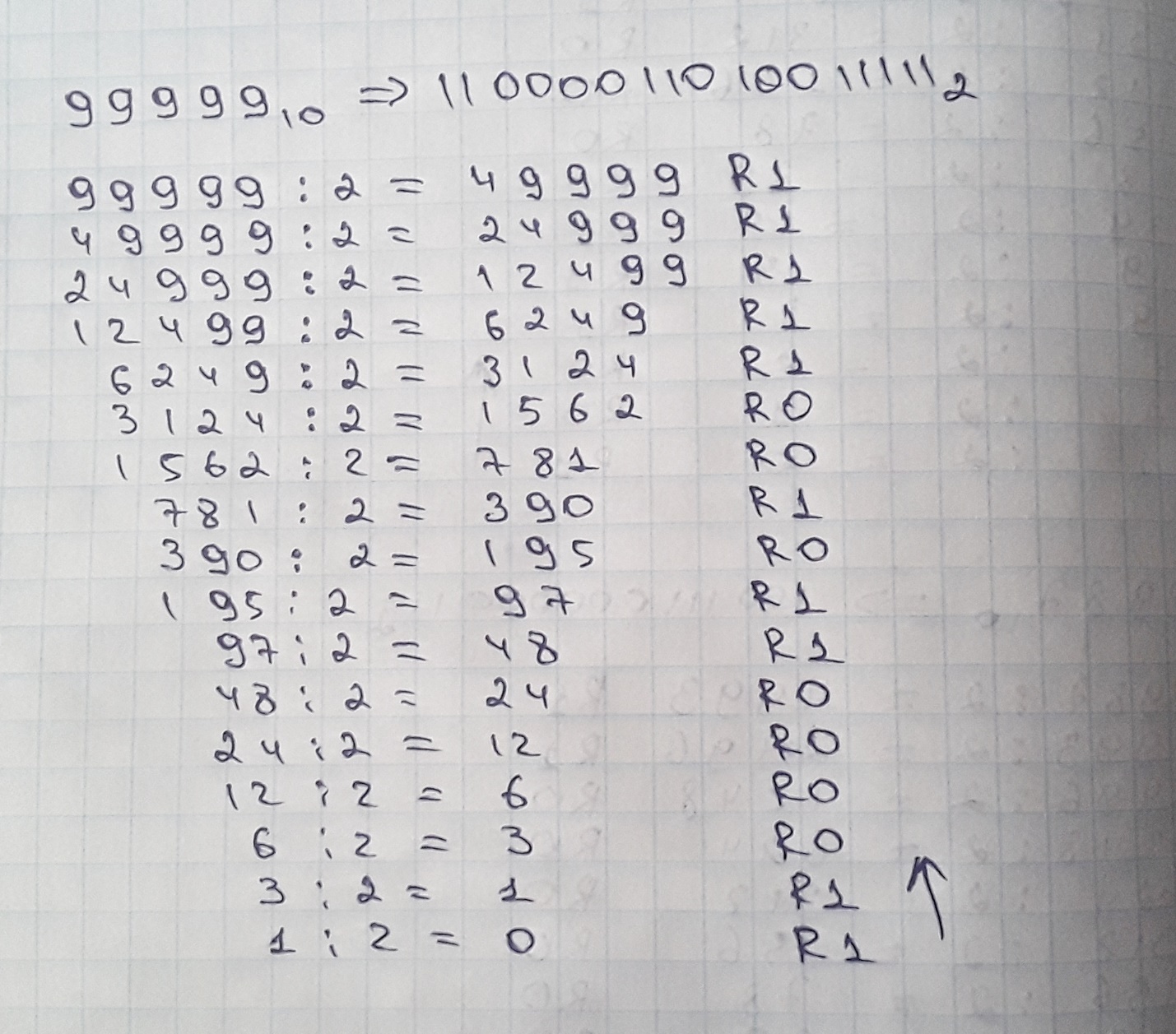
**Task 2**

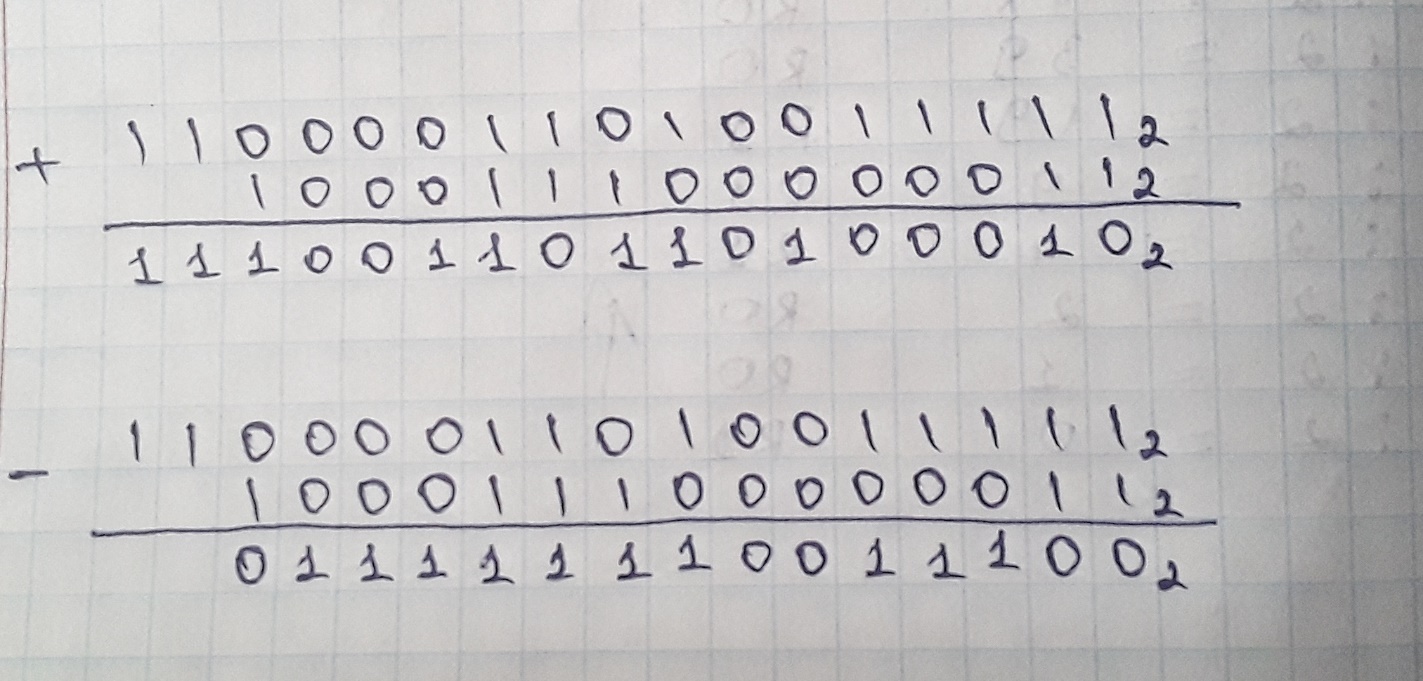
**a)** *Decimal to Binary (R – Remainder):*

*Decimal to Hex:*

**

**b)** *Conversion of 99,999 to binary:*

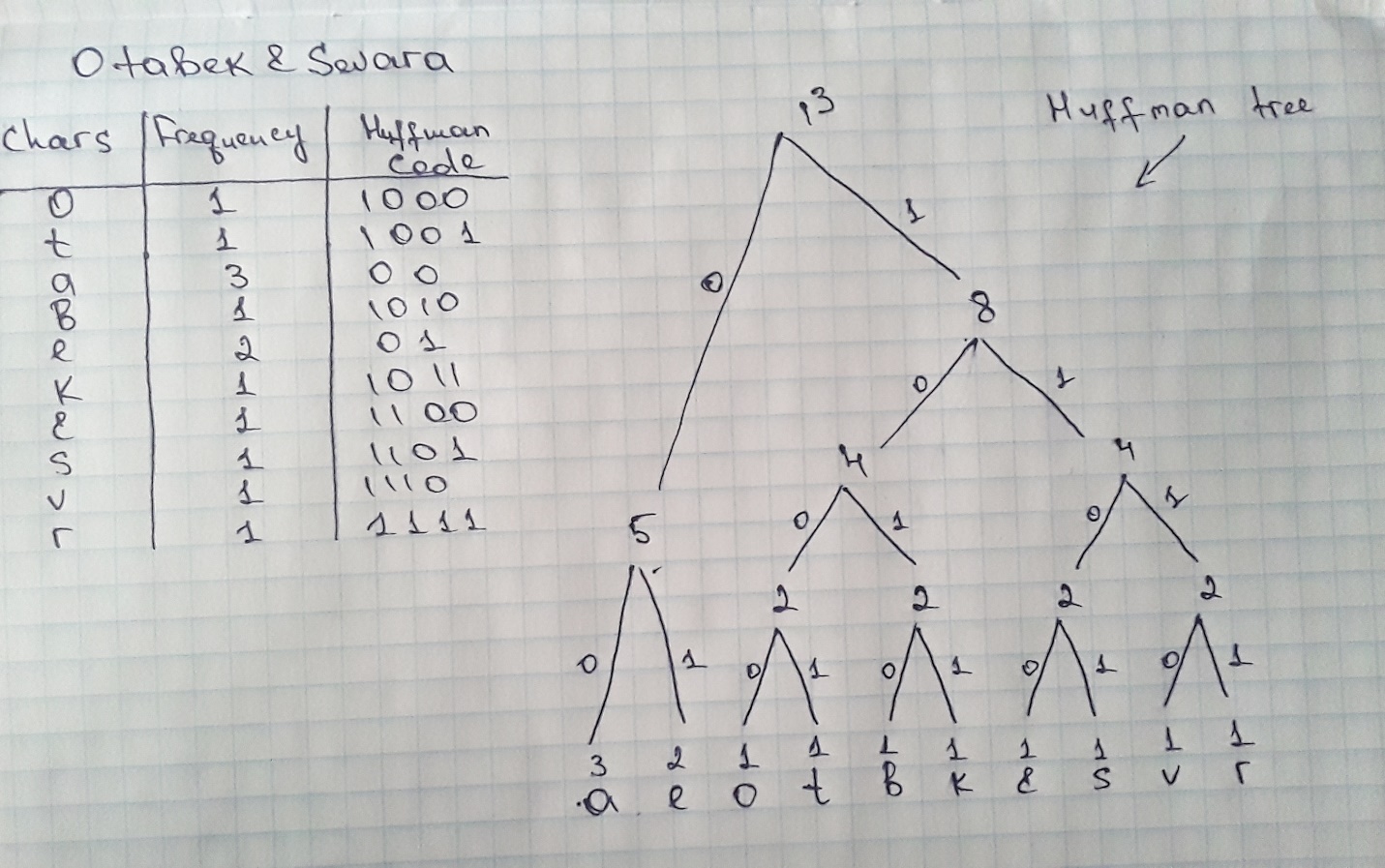
*Addition and subtraction, respectively:*

**

**c)** There are several reasons why using hexadecimal is more efficient than other number bases. The reasons are, firstly, since digits that are more closely look alike to usual base-10 number system used in hexadecimal, it is easier to read the digits at the first glance. Secondly, higher information density, that is in order to show any number between 0 and 255, only 2 digits are required in hexadecimal, and when it comes to do the exact same thing in binary, 8 digits are required (Savas, 2016).

**Task 3**

*Table and Huffman tree, respectively:*



**Task 4**

**Step 1:**

The numbers are 9, 9, 8, 7, 4, 5, 2, 3, 7

In ascending order: 2, 3, 4, 5, 7, 7, 8, 9, 9

The number to look for is 4 (x = 4).

**Step 2:**

* Midpoint is 7 (9 / 2 = 4.5, 5th number is the midpoint)

2, 3, 4, 5, 7, 7, 8, 9, 9

* Since x < midpoint, ignore the range on the right (7 – 9)

2, 3, 4, 5, ~~7, 7, 8, 9, 9~~

* Midpoint is 3 (4 / 2 = 2, 2nd number is the midpoint)

2, 3, 4, 5, ~~7, 7, 8, 9, 9~~

* Since x > midpoint, ignore the range on the left (2 – 3)

~~2, 3,~~ 4, 5, ~~7, 7, 8, 9, 9~~

* Midpoint is 4 (2 / 2 = 1, 1st number is the midpoint)

~~2, 3,~~ 4, 5, ~~7, 7, 8, 9, 9~~

* x is equal to midpoint (5 = 5). The number we are looking for is found.

**Task 5**

**1.** In paged memory technique, main memory is split into parts or chunks of storage called frames. A frame is a physical memory unit with the fixed size, and frames are space of storage for pages. Pages are the sections or parts of a task or process (which is being executed), and depicts a logical memory unit.

The functionality of splitting a task or process into sections eases the difficulty of loading a task from locating an accessible huge section of space to locating sufficient small sections. This way a process or task does not have to be stored contiguously in memory anymore, and it is one of the advantages of paging.

A significant enlargement to the paged memory management is the concept of demand paging that favors from the case that not all sections of a program need to be stored in memory at simultaneously. Since the CPU can access only one page of a task or process at any time, it is not a big deal if the memory is allocated evenly to the other pages of that task.

The demand paging method support the concept of virtual memory, the idea that the size of a program is not restricted. With the other management techniques, the full program needs to be stored into memory as a continuous whole. What demand paging does is to remove that restriction (Dale and Lewis, 2002).

**2. a)** In order to produce a physical address, firstly, the frame number should be found with help of the page in the page-map table. Secondly, frame number should be multiplied by frame size and added to the offset to find the physical address.

The physical address of <2, 85> with the frame size of 1,024:

5\*1024 + 85 = 5,190.

**b)** The physical address of <0, 1026> can not be found since the offset is larger than the frame size (which is 1024).

**Reference list**

Dale, N. and Lewis, J. (2002). *Computer Science Illuminated,* edition. Sudbury, MA: Jones and Barlett Publishers.

Savas, N. (2016). Why do we use hexadecimal? *Medium*. Available from <https://medium.com/@savas/why-do-we-use-hexadecimal-d6d80b56f026> [Accessed 14 January 2021].