1 Integer Radix Sort (9 marks)

Consider the Radix sort algorithm presented in lectures. As discussed in the second part of lecture 2, it is possible to vary the base used by this algorithm.

In this task you need to use radix sort to sort a given list of integers into ascending numerical order, **using a given base**. To do this, you will write a function `num_rad_sort(nums, b)`.

There will be a **video on moodle** which discusses some concepts related to representing numbers in different bases. If you are confused about any aspect of this task, please watch the video!

1.1 Input

- `nums` is a unsorted list of non-negative integers
- `b` is an integer, with value ≥ 2

1.1.1 FAQ

Q: In what base will the input be given?
A: This question does not really make sense. If the input list were a list of strings such as "14", then you would need to know what base was being used to interpret them. For example, "14" in base ten is different to "14" in base 5. Since the input list is composed of integers, not strings, each value in the input list is a numerical value, which is independent of a base.

Q: When I write a list like `[1,45,173]` is this not in base 10?
A: Python by default assumes that any numerical values written by the programmer are in base 10. This means that if we are writing a list of values into our code, we must write in base 10. We have to use a particular base because we are **representing** the numbers, and when numbers are being represented, they need a base. Once python has read the numbers in, there is no longer a need to represent them to the programmer, so their representation, and therefore their base, is no longer relevant to us.

Another way to think about this is that if Python decided to convert he numbers to a different base once it had read in the contents of the list, it would not affect any of the operations we might ask Python to perform on those numbers.

Q: How can a number not have a base?
A: A base is only relevant when we are talking about the **representation** of a number. The concept of **three**, for example, exists independently of its representation as a word, the digit "3", or the digits "11" (which is three in base 2), or indeed, ". . ." (which is three dots representing the quantity). All of these are just different ways of writing down the same numerical value.

Q: In what base should we give out output?
A: Since the output does not need to be printed, and is not a list of strings, the same answer applies as regarding the input. The output does not have a base.

Q: How does the parameter `b` affect the output?
A: It does not. Varying `b` while keeping `nums` unchanged will result in the same output every time. It does, however, change how the algorithm operates. If you do not correctly use the parameter `b`, you will not receive full marks for this task.

1.2 Output

`num_rad_sort` returns a list of integers. This list will contain exactly the same elements as `nums`, but sorted into ascending numerical order.

Example:

```
nums = [43, 101, 22, 27, 5, 50, 15]
>>>num_rad_sort(nums, 4)
[5, 15, 22, 27, 43, 50, 101]
```

1.3 Complexity

`num_rad_sort` should run in $O((n + b) * log_b M)$ time where

- n is the length of `nums`
- b is the value of `b`
- M is the numerical value of the maximum element of `nums`

2 Timing bases (9 marks))

In this task, you should **re-use** `num_rad_sort`.

We saw that varying the base in Task 1 did not change the output. However, it does have an effect on the **runtime** of radix sort. In this task we will investigate the relationship between the base and the runtime.

You will write a function `base_timer(num_list, base_list)` which you will use to investigate the relationship between the base used and the runtime.

2.1 Input

- `num_list` is a list of non-negative integers
- `base_list` is a list of integers, all with values ≥ 2, sorted ascending

2.2 Output

`base_timer` a list of numbers. Element i in this list is the time taken to run your radix sort from Task 1 on `num_list` using element i from `base_list` as the base.

Since the actual runtimes will vary between students due to differences in implementation and hardware, there are no marks for the exact values of the times you obtain as output. In other words, just because two students obtain different outputs for the same input does **not** mean that one student has an error.

The marks for this task come from the nature of the output, and your explanation of it (details in section 2.3)

2.3 Explanation

For this task, you will run `base_timer` on various inputs, and explain the results.

Insert the following code snippet into your assignment in order to create four lists of data and produce output for them:

```
random.seed("FIT2004S22021")

data1 = [random.randint(0,2**25) for _ in range(2**15)]
data2 = [random.randint(0,2**25) for _ in range(2**16)]

bases1 = [2**i for i in range(1,23)]
bases2 = [2*10**6 + (5*10**5)*i for i in range(1,10)]

y1 = base_timer(data1, bases1)
y2 = base_timer(data2, bases1)
y3 = base_timer(data1, bases2)
y4 = base_timer(data2, bases2)
```

In `explanation.pdf` document, include 2 graphs. One comparing `y1` and `y2`, with `bases1` as the horizontal axis. The other comparing `y3` and `y4`, with `bases2` as the horizontal axis.

For the first graph, you should use a **logarithmic** scale on your horizontal axis, but for the second graph you should use a **linear** scale.

The vertical axis corresponds to the **runtimes**. This axis should **not** use a logarithmic scale.

In the same pdf, answer the following questions:

1. Why do the base/time curves for the first two graphs show a U shape? In other words, why are the times high when the base is low and when the base is high, but low when the base is in between? Justify your answer using the complexity of radix sort.

2. Why are the times for `y2` about twice as long as for `y1` when the base is low? Include a mathematical arguement based on the complexity of radix sort in your answer.

3. Why are the times for `y2` **not** twice as long as for `y1` (and in fact are very close) when the base is high? Include a mathematical arguement based on the complexity of radix sort in your answer.

4. Why are the times for `y3` and `y4` almost the same, despite `data2` having twice as many elements as `data1`? Include a mathematical arguement based on the complexity of radix sort in your answer.

5. Why do the graphs for for `y3` and `y4` show an almost linear shape? Include a mathematical arguement based on the complexity of radix sort in your answer.

2.4 Complexity

The overall complexity for this task is not important.