COMP160: Algorithms, **Homework 2**

- To obtain full credit, you must justify your answers. When describing an algorithm, do not forget to **state any assumptions that you make**, analyze its running time and explain why the algorithm is correct.
- Although not specifically stated, You can assume that we look for algorithms that are as fast as possible, and bounds as tight as possible.
- You may discuss these problems with others, but remember to write the answers on your own. In case of doubt, cite any source you used to do the assignment.
- Remember to submit each question in a separate page.

1. In class, we learned how to select the k-th smallest element using groups of 5.
   (a) What would happen to the runtime if we instead make groups of g, for some constant $g \geq 6$? Give the explicit recursion and resulting runtime. As always, justify your answer.
   (b) **Challenge** (not needed for full credit). Consider two very different group sizes (say, $g = 10$ and $g = 10^6$). Because both are constants, the asymptotic runtime for the Selection algorithm will be the same for both values of g. However, in practice we may see a significant difference in the runtimes. Which value of g do you think will ultimately lead to a faster algorithm? A small value of g or a large one? Justify briefly your answer.
2. Let's give a formal proof that RadixSort works and give a bound on its runtime. For this exercise you need not justify anything that relates to CountingSort (just state the result).
   (a) We start with correctness:

   Lemma 1. RadixSort *will properly sort any n natural numbers.*

   Prove this statement. You can use any strategy you want, but we recommend using induction on ℓ (the number of digits that each value has). You can assume that correctness of CountingSort has been shown (no need to prove yourself).
   (b) Let's focus now on runtime:

   Lemma 2. RadixSort *runs in $O(\ell(n+d))$ time, where d is the radix (the number of digits in the base) and ℓ is the maximum length of the n numbers.*

   Give a brief justification of this statement.
3. We want to sort n integers in the range 0 to n^k, where k is a positive constant, as fast as possible. The integers have length ℓ, and the radix (base) is d. Do not assume these are $O(1)$.
   (a) Say we use standard counting sort. What is the time complexity of this algorithm?

(b) We now consider using RadixSort instead. The function depends on two parameters ℓ and d. Find an explicit function of ℓ as a function of d (it will also depend on n and k), giving a one-sentence justification for your formula. What happens to ℓ as d increases? For example, how does ℓ change if we go from d to d^2? For this last question only you can ignore minor adjustments like floors, ceilings, or ± 1.

(c) Say that $d = 2$. What is the asymptotic runtime?

(d) Now, we want to strategically select a value for d which will reduce the **asymptotic** runtime of RadixSort. What do you think would be the best value of d? What would be the runtime? Justify your answer. For this question you can assume that the number comes in the desired base (don't worry about conversions). Also assume that digit operations take constant time (regardless of the value of d).

(e) (Optional challenge, food for thought.) Now let's focus on the changing the base step. Say that a number comes expressed in base 2 (say, 1010010101110), and you want to change it to your new base d from part (d) above. How much time do you need to do the conversion? You can assume that a division operation takes constant time (even though this is not necessarily correct).