Nguyen Quy Duc – A0244126M – T17

Programming Assignment 1

Task 1 – Mastering Master Theorem

1. **Give as many creative suggestions to Steven to improve exactly this task for S2 AY2023/2024, e.g., how to make the task covers more variations of Divide and Conquer recurrences**

- One approach that I thought of when doing this task is to put some test cases that cannot be applied Master Theorem. The mission of the students is to detect which case cannot be applied and cout the result

2. **What are your learning points about Master theorem after going through this version of task A?**

- My learning point after going task A is:
  3 cases of Master Theorem can be differentiated at one glance by comparing logb(a) and the power of x in f(x). Before this, I was quite confused and not too sure when applying the theorem by hand ☹. But after divide the function into smaller cases, it helps me to do by hand faster.

3. **What are Master theorem pros/potential-automation?**
   (PS: do you double check your Written Assignment 1 answers using this code?)

- Pros of Master theorem:
  - Time efficiency: Using Master Theorem can quickly find the complexity of the algo without having to analysing recursively
  - Consistency: Because Master Theorem is divided into different cases, knowing which case to apply the equation is enough to get the correct solution
- Potential-automation:
  - Because Master Theorem is equation, then it's quite straight forward to automate the process of using Master Theorem to analyze the time complexity

4. **What are Master theorem cons/limitations?**

- Cons:
  - Master Theorem can only be applied to specific cases
  - Sometimes, it's not really efficient in practice because there might be different size subproblems, different number of subproblems, etc.

Task 2 – Linear-time Sort

1. **This task B1 is generally about beating the lower bound of Ω(n log n) of comparison-based sorting. We have tested (as best as we can) that no form of O(T C × n log n) comparison-based sorting can successfully pass the time limit on our secret test cases. Can you give an explanation on why is that so (or surprise us by writing a code that runs in O(T C × n log n) but gets Accepted by the judging server)?**

- I think the reason for this is the constraints of elements given is mod of C. So all the elements should be smaller than C (except for A). Therefore, the time complexity accepted in the best case will be some form of N + C. So as long as the test cases have N, C such that logN > 1 + C/N (NlogN > N + C), it's guaranteed that an algo O(nlogn) cannot pass the test cases.

2. **What are the simplifying constraints that you can take advantage of to design a sorting algorithm that is not comparing integers?**

- Some simplifying constraints:
    - Comparison-based but comparing the hash values of the elements or comparing based on the result return by the comparing function (comparing two objects for example)
    - Use non-comparison algorithm like radix sort.

3. **Explain your solution in details, show/prove that it is correct, and analyze its time complexity.**

- For task B1, I used radix sort with base r = 1000. This base is obtained by trials and errors. And I think for B1, a simple counting can work as well.
- Correctness:
    - The correctness of the also is guaranteed by the correctness of counting sort in each iteration of radix sort (which has been proved in lecture 04).
- Time complexity:
    - O(N log_r(max element)) = O(N log1000(max(A, C)))

4. **First, explain what is the main difference between task B1 and B2!**

- The main difference is resulted in the constraints of A, B and C now can be up to 10^9, but N < 10^6. Therefore, it's guaranteed that counting sort cannot pass the time limit in this case because when N is small together with when C is large, counting sort will perform way worse than comparison based sorting algorithm.

5. **Explain your solution in details, show/prove that it is correct, and analyze its time complexity.**
   If your B2 solution is identical with B1, you can just say so; Otherwise, explain the new solution.

- For my solution for B2, I still used radix sort but rather to improve the counting sort time complexity in each iteration. Instead of using frequency array and an output array (which require 4 loops), I used array of queues to keep track of the elements (which only require 2 loops).
- The correctness of the algo is still guaranteed because it's the same algo but different implementation
- Time complexity:
    - O(N log_r(max element)) = O(N log1000(max(A, C)) (same as B1)