FIT1045/FIT1053 Algorithmic Problem Solving – Tutorial 07.

Objectives

After this tutorial, you should be able to:

- Work with poly-logarithmic complexity functions.
- Understand how to sort a list using mergesort.
- Analyse Brute-Force GCD Algorithm.
- Apply the divide and conquer strategy to solve computational problems.

Prepared Question: Solving Algorithm 'x'

Below is an algorithm called 'x'. Given the list `lst = [-9, -3, 5, 19, -1, 3, 11, 17]`. What is the returned from the function when `lst[:len(lst)//2]` and `lst[len(lst)//2::]` are invoked as `'a'` and `'b'`.
What is the reason why we need to return `a[i::]` + `b[j::]` on top of `ret_val` at the end of the code?

```
def x(a, b):
    ret_val = []
    n1 = len(a)
    n2 = len(b)
    i = 0
    j = 0

    while i < n1 and j < n2:
        if a[i] > b[j]:
            ret_val.append(b[j])
            j = j + 1
        else:
            ret_val.append(a[i])
            i = i + 1
    return ret_val + a[i::] + b[j::]
```

Workbook

§9 and §10.1

While your tutors mark the prepared question, attempt the above Workbook sections. Your tutors will go through some of these questions with the class.

Warm-up

Do not solve these questions by running in Python.

Poly-logarithmic bounds: For each of the following concrete time complexity give the tightest bound in terms of a simple poly-logarithmic function $n^c log^d n$ using big-Oh notation. That is, determine the smallest pair[1] c and d such that $T(n) \in \mathrm{O}\big(n^c log^d n\big)$. Here log refers to the logarithm of base 2.

- $T(n) = \log(10n + 100)$
- $T(n) = \log_{1.5} n$

Invariant of Binary Search: Below is an implementation of the Binary Search algorithm. Identify the loop invariant, loop exit condition, and post-condition.

- ```
  def binary_search(v,seq):
      a = 0
      b = len(seq) - 1
      while a <= b:
          c = (a + b) // 2
          if seq[c] == v:
              return c
          elif seq[c] > v:
              b = c - 1
          else:
              a = c + 1
      return None
  ```

Linear Time Merging: Below are sorted sub-lists. Merge the sub-lists together so they are sorted and count the number of comparisons that happen between the lists.

- `[1, 2, 5, 8] and [2, 7]`
- `[1, 3], [2, 4] and [1, 9], [2, 5]`

Merge Sort: Practice sorting the following lists using *merge sort*. Starting from 'n' chunks with 1 element record the sequence of merged lists.

- `[3, 1, 6, 8, 11, 7, 14, 12]`
- $[P, Y, T, H, O, N]$

[1] Here, we use lexicographic order to determine what is a smaller pair. That is, we consider $(c, d) = (1, 2)$ to be smaller than $(c', d') = (2, 1)$.
  ```

Analysis of Brute Force GCD Algorithm

Recall the brute force algorithm for finding the greatest common divisor of two integers, given in Lecture 11:

```
def gcd_brute_force(a, b):
    """ Input : integers a and b such that not a==b==0
        Output: the greatest common divisor of a and b
    """
    x = min(a, b)
    while not (a % x == 0 and b % x == 0):
        x = x - 1
    return x
```

1. Trace through the algorithm for the following input: `gcd_brute_force(10,8)`. How many iterations of the loop does it take?
2. What is the loop invariant of this algorithm? What is the loop exit condition? How do these two things together give a post-condition that shows the algorithm's correctness?
3. Analyse the computational complexity of this algorithm. What is the worst-case complexity, in big-O notation? What is the best case? Consider the input size as `n`, where `n == abs(a) + abs(b)`.

EXTENSION: Quick Sort (Non-Compulsory, FIT1053 students should attempt)

Quick Sort is another type of Divide and Conquer algorithm. It works by using a pivot selection strategy in order to partition the list into sub-lists and place the pivot in its correct position. Then it continues on using the same pivot strategy in order to further partition the remaining sub-lists until the list is completely sorted.*

In small groups, practice applying the pivot function to the list, always using the first element of the subsection as the pivot. Show what the list looks like after each subsection has had the pivot function applied. See what happens if you choose an element other than the first for the pivot (e.g. middle-most/right most).

- $[5, 7, 9, 1, 6, 9, 5, 3]$
- $[2, 6, 2, 6, 5, 1]$
- $[3, 1, 2, 7, 9, 1, 5, 2]$
- $[P, Y, T, H, O, N]$

EXTENSION: Closing down a post office (Non-Compulsory, FIT1053 students should attempt)

Consider Task 2 "Closing down a post office" from Workshop 7. Come up with a simple (non-divide-and-conquer) algorithm for this task and discuss its computational complexity. What could be a divide-and-conquer strategy for this problem? Could this lead to an algorithm with lower time complexity than the simple algorithm? Can you identify a critical sub-problem that you would need to be able to solve in linear time (similar to merge in Merge Sort)?