Assignment 1

Due: January 16, 2020

Kanyid, Bradon
bradon.kanyid@pdx.edu

Reimer, Daniel
daniel.reimer@pdx.edu

Problem 1: Asymptotic Analysis Practice

(a) [5 points] Prove or disprove that $\log_k n \in O(\lg n)$ for any $k > 1$. (Note that lg refers to $\log_2$)

Solution

Another way of stating $\log_k n \in O(\lg n)$ for any $k > 1$ is:

$$\begin{aligned} \log_k n &\leq \lg n \text{for any} k > 1 \\ \frac{\lg n}{\lg k} &\leq \frac{\lg n}{\lg 2} \end{aligned}$$

Since $\lg 2 = 1$,

$$\begin{aligned} \frac{\lg n}{\lg k} &\leq \lg n \\ \left(\frac{1}{\lg k}\right) \lg n &\leq \lg n \\ \left(\frac{1}{\lg k}\right) &\leq 1 \text{for any} k > 1 \end{aligned}$$

(b) [5 points] The following recurrence relation solves to $O(n \lg^2 n)$. Prove this by substition. Do not use the Master method.

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \lg n \\ T(1) &= 0 \end{aligned}$$

Solution

$$T(n) = 2T\left(\frac{n}{2}\right) + n\lg n$$
$$T(\frac{n}{2}) = 2[2T\left(\frac{n}{4}\right) + \frac{n}{2}\lg\frac{n}{2}] + n\lg n$$
$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + k\left(n\lg n\right)$$

Assume that:

$$T\left(\frac{n}{k}\right) = T\left(1\right)$$

That implies:

$$\frac{n}{2^k} = 1$$
$$n = 2^k$$
$$k = \lg n$$

Substituting $k = \lg n$

$$T(n) = 2^{\lg n} T\left(\frac{n}{2^{\lg n}}\right) + \left(\lg n\right)\left(n\lg n\right)$$

Simplifying...

$$T(n) = nT\left(1\right) + n\lg^2 n$$
$$= O(n\lg^2 n)$$

(c) [5 points] Suppose that $f(n)$ and $g(n)$ are non-negative functions. Prove or disprove the following: if $f(n) \in O(g(n))$ then $2^{f(n)} \in O(2^{g(n)})$.

Solution

Similar to Problem 1a, another way of stating $f(n) \in O(g(n))$ is:

$$f(n) \leq g(n)$$

To prove that $2^{f(n)} \in O(2^{g(n)})$, then, one must only state in the same way that:

$$2^{f(n)} \leq 2^{g(n)}$$

Taking the lg of each side, the inequality holds,

$$\lg\left(2^{f(n)}\right) \leq \lg\left(2^{g(n)}\right)$$
$$f(n) \leq g(n)$$

Thus,

$$2^{f(n)} \in O(2^{g(n)})$$

Problem 2: Peak-finding Given a set of real numbers stored in an array A find the index of a *Peak*, where a *Peak* is defined as an element that is larger or equal to the both the elements on its sides. (Note: you only need to return *a* peak, not the highest one.)

Example array:

-2	6	-1	4	9	-5	5

Assuming the array one based it has peaks at indices $\{2, 5, 7\}$.

(a) [5 points] Give a linear time algorithm to solve this problem. (This should be obvious)

(b) [10 points] Give a $O(\log n)$ time algorithm to solve this problem.

Solution

To solve this problem, the algorithm starts in the middle of the array. It then finds the biggest number between the left, right, and the middle. If the middle is the biggest, then that is the peak. Otherwise, it selects the middle towards the side of the bigger number. The process is repeated over again until peak is found. This algorithm is $O(\log n)$ because each iteration, the possible selection of numbers is halved. Therefore logarithmic.

```
peakFinding(xs) {
  let middle = len(xs) / 2
  let left   = 0
  let right  = len(xs)-1
  while(true) {
    if(xs[middle-1] > xs[middle+1] > xs) {
      old_middle=middle
      middle=(left + right) / 2
      right=old_middle
    }
    elseif(xs[middle+1] > xs[middle-1] > xs) {
      old_middle=middle
      middle=(left + right) / 2
      left=old_middle
    }
    else {
      return middle
    }
  }
}
```

(c) [10 points] What if instead of a simple array you are given a square matrix, where a *Peak* is now defined as an element larger or equal to its four neighbours. Give a $O(n \log n)$ solution to this variant of the problem.

Solution

The maximum value will always be a peak. In this problem, the square matrix can be represented as a one dimensional array. The data is sorted and a maximum could be retrieved in constant time. Once the maximum is found, the index can be found in constant time by mapping the one-dimensional index back to two-dimensional. Since the sorting algorithm used is $O(n \log n)$ and the other operations are all in constant time, the overall complexity is $O(n \log n)$.

```
  2dPeakFinding(xs) {
    let m = square matrix dimension
    let sortedXs = mergeSort(xs)
    let max = max(sortedXs)
    return (indexOf(max) / m, indexOf(max) % m)  # (x,y)
}
```