Daniel Oliveros A02093272
January 25, 2018

Assignment 1: Algorithm Analysis

Problem #1 Issue with exponential time algorithms:

(a) To solve this operation, we can divide the number of operations by the number of operations per second the Sunway TaihuLight can perform. So:

$$\begin{aligned} Seconds &= \frac{numOps}{psPerSecond} \\ Seconds &= \frac{2^{100}}{1.25*10^{17}} \\ Seconds &= \frac{1.26*10^{30}}{1.25*10^{17}} \\ Seconds &= 1.01 * 10^{13} \end{aligned}$$

And then, to pass this to centuries:

Seconds	Hours	Days	Years	Centuries
$1.02 * 10^{13}$	$2.82 * 10^{9}$	$1.17 * 10^{8}$	322238	3222.38

(b) To solve this operation, we can divide the number of operations by the number of operations per second the Sunway TaihuLight can perform. So:

$$\begin{aligned} Seconds &= \frac{numOps}{psPerSecond} \\ Seconds &= \frac{2^{1000}}{1.25*10^{17}} \\ Seconds &= \frac{1.08*10^{301}}{1.25*10^{17}} \\ Seconds &= 8.57 * 10^{283} \end{aligned}$$

And then, to pass this to centuries:

Seconds	Hours	Days	Years	Centuries
$8.57 * 10^{283}$	$2.38 * 10^{280}$	$9.9 * 10^{278}$	$2.72 * 10^{276}$	$2.72 * 10^{274}$

Problem #2 Items ordered based on asymptotic order:

1. 2^{500}
2. $\log(\log n)^2$
3. $\log n, log_4 n$
4. $\log^3 n$
5. $2^{\log n}$
6. $\sqrt{n}$
7. $n \log n$
8. $n^2 \log^5 n$
9. n^3
10. 2^n
11. $n!$

Problem #3 Indicate whether the following pairs of functions is one of the three cases:
$f(n) = O(g(n)), f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$

(a) $f(n) = 7\log n, g(n) = \log n^3 + 56$. In this case: $f(n) = \Theta(g(n))$

(b) $f(n) = n^2 + n\log^3 n, g(n) = 6n^3 + \log^2 n$. In this case: $f(n) = O(g(n))$

(c) $f(n) = 5^n, g(n) = n^2 2^n$. In this case: $f(n) = \Omega(g(n))$

(d) $f(n) = n\log^2 n, g(n) = \frac{n^2}{\log^3 n}$. In this case: $f(n) = O(g(n))$

(e) $f(n) = \sqrt{n}\log n, g(n) = \log^8 n + 25$. In this case: $f(n) = \Omega(g(n))$

(f) $f(n) = n\log n + 6n, g(n) = n\log_3 n - 8n$. In this case: $f(n) = \Theta(g(n))$

Problem #4 Knapsack factor of 2 approximation.

1. **Algorithm Description:** The idea is simple, go through the array element by element and see if you can use that element for your solution. In order to pick which elements to add to our list we set up some rules depending on the element itself and on our current running sum.

2. **Pseudocode:**
```
def Knapsack(arr, K):
    nums = []
    val = 0
    for cur in arr:
        if(K/2 <= cur <= K):
            return [cur]
        if(cur + val <= K):
            nums.append(cur)
            val += cur
        if(K/2 <= val):
            return nums
    return []
```

3. **Correctness:** Our algorithm is correct because throughout its execution we will never add a number greater than half of K to another number that also falls within this same range, which makes it so we never go over K. Also, whenever we find a number that, on its own, can be a solution for the problem, we are able to return that number by itself.

4. **Time Analysis:** This algorithm goes through each element of the array once, performing operations of only constant complexity every time it goes through. It, therefore, has a runtime of **O(n)**

People I worked with for this assignment:

- Chris Darais