Hunter McMahon

CIS 313 Written Assignment 2

1. determine & explain the run times of the following code;

```
sum = 0
for i = 1 to n*n            ] - O(n^2)
    for j = 1 to i*i*i      ] O(n^3)
        sum++
```

- The Runtime is $\Theta(n^5)$ as we have a nested loop w/ runtime $O(n^3)$ within a loop of runtime $O(n^2)$ thus the upper bound is their product which is $O(n^5)$. from this, we know that the run time in $\Theta(n)$ form would be about the same as it would have to be $\leq$ the upper bound so we can generalize that its about = to the upper bound and say the runtime is $\Theta(n5)$

and

```
sum = 0
for i = 1 to n*n    ] - θ(n^2)
    j = i
    while j >= 1    ]
        sum++       ]
        j = j/5     ] θ(log_5(n))
```

The Run time of the code is $\Theta(n^2 \log_5 n)$. This is as we have a nested loop thus meaning that the total rutime is the product of the two loops runtimes, in this case, the inner loop has a runtime of $\Theta(\log_5(n))$ because of the maintenance line $j = j/5$ and ~~becaus~~ because of the run condition of $j >= 1$. meanwhile the outer loop iterates n^2 times thus iterating the inner loop by n^2 times giving us $\Theta(n^2 \log_5(n))$. Additionally, there are no factors within the loops that would cause this runtime to change thus we can confidently asume that in this case, the upper bound equals the tight bound

2. What is the runtime of the following code which multiplies two $n \times n$ matrices, A & B, and stores the result in C? explain

$$
\begin{array}{l}
\text{for } i=1 \text{ to } n \quad - O(n) \\
\quad \text{for } j=1 \text{ to } n \ \{ \quad - O(n) \\
\qquad C[i,j]=0 \\
\qquad \text{for } k=1 \text{ to } n \quad - O(n) \\
\qquad\quad C[i,j] = C[i,j] + A[i,k] \cdot B[k,j] \\
\qquad \}
\end{array}
\quad - \Theta(n^2) \quad \Theta(n^3)
$$

$= O(n) \cdot O(n) \cdot O(n) = O(n^3)$

The runtime of the code is $\theta(n^3)$. This is as we have a double nested loop thus the total run time is equal to the product of each loops runtime. Additionally, since every loop iterates strictly from 1 to n, their upper and lower bounds are $\equiv$ therefore they each have a runtime of $\theta(n)$ giving us a total runtime of $\theta(n) \cdot \theta(n) \cdot \theta(n) = \theta(n^3)$.

3. TB exercisize 2.1-2; Perform a Loop invariant of the following:
└Show that it sums #'s A[1:n]

```
SUM-ArrAY(A, n)
    sum = 0
    for i = 1 to n
        sum += A[i]
    return sum
```

Initialization:
We know that if there is nothing in the array, then its sum is = 0 as the number 0 represents an empty quanity. Thus it can be used to represent the sum of the empty array as it has no values in it. Also if $n = 0$, the Loop cannot execute as $i = 1 > 0 = n$

Maitenence:
Let A be the set of x elements such that $A = [a_1, a_2, \dots a_x]$
Then the summation of the elements of A for n index's would be

$$m = \text{sum}(A) = \sum_{j=1}^{n} A[j]$$

m can be rewritten as:

$$m = \sum_{j=1}^{n-1} A[j] + A[n] \quad \text{(def of summation)}$$

Therefore, the sum of n elements in Array A equals the summation of n-1 elements in A plus A[n] which is exactly what our for loop does repeatedly, it adds the element A[i] to the sum of all previous elements until it reaches n

Termination:
The for loop will only iterate if $n \geq 1$ as $i = 1$. Therefore if $n \leq 0$ then the loop does not iterate and the sum remains at 0. If there is an valid n however, the loop will only iterate up to that n value by nature of a for loop and will result in the $\text{sum} = \sum_{j=1}^{n-1} A[j] + A[n]$

* i Prefer to use +=, *= etc.

4. T-B Problem 2-3 Horners Rule

given Coef. $a_0, a_1, a_2, \dots a_n$ as $P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$

Horners Rule says to evaluate this as: $P(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + x a_n)))$
implementation given coefficents $a_0, a_1, a_2, \dots, a_n$ in array $A[0:n]$ and an value x;

```
0 Horner(A, n, x)
1     p = 0
2     for i = n down to 0
3         p = A[i] + x·p
4.    return p
```

A. Write the runtime in terms of Θ notation

The runtime of Horner is $\Theta(n)$

b. Write Pseudo-code to implement the naive polynomial evaluation algorithm, What is its run time compared to horner

```
naive(A, n, x)
    p = 0
    for i to n
        power = 1
        for j = 1 to i-1
            power *= x
        p += A[i] · power
    return p
```

The run time of the naive is slower at $\Theta(n^2)$ compared to horners $\Theta(n)$

C. Considez the Loop invariant for horner @ the start of each iteration in Lines 2-3 $p = \sum_{k=0}^{n-(i+1)} A[k+i+1] \cdot x^k$ * interpret summation w/ No terms as $= 0$

Use a Loop invariant proof to show that ~~~~ $p = \sum_{k=0}^{n} A[k] \cdot x^k$ at termination

- Initialization:
    - We know that the summation is 0 at the start as It has no terms

Maitenence:

We have $P = \sum_{k=0}^{n-(i+1)} A[k+i+1] \cdot x^k$ and are itterating

down to zero Thus $i = i - 1$

We want

$$\sum_{k=0}^{n-((i-1)+1)} A[k+(i-1)+1]\,x^k = \sum_{k=0}^{n-i} a[k+i]\,x^k$$

$$=$$

$$\sum_{k=0}^{n-i\cancel{+1}\cancel{-1}} A[k+i\cancel{-1}\cancel{+1}]\,x^k$$

$$\sum_{k=0}^{n-i} A[k+i] = \sum_{k=0}^{n-i} a[k+i]\,x^k$$

Termination

- At the end of iteration, $i=-1$ as we have $i=n\ (=n-1)$

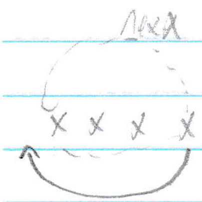
- Therefore we have:

$$\sum_{k=0}^{n-(-1+1)} a[k-1+1]\cdot x^k$$

$$=$$

$$\sum_{k=0}^{n} a[k]\,x^k$$

5. TB 10.2.5 give an $\Theta(n)$-time non-recursive function that reverses a singly linked list; it should use no more than constant storage




```
List_Reverse (Self)
    Old_head = Self.head
    X = true
    While X = true
        if Self.Tail == Old_head;
            X = False
            Break
        else
            Self.Tail.Next = Self.head
            Self.head.Previous = Self.tail
            Self.head = Self.tail
            Self.tail = Self.head.Previous
            Self.tail.next = none
            Self.head.previous = none
```

* this function assumes its a Procedure within an Linked List Object

6. Implement a stack using a single queue; you are given queue Q that has the methods;
Q.size() - returns queue size at any point
Q.enqueue(x)
Q.dequeue()

- Use these to create stack S w/ push(x) & pop methods
- What is the runtime of the two methods

```
Class S:
    def __init__(self)
        self.__queue = Q()
```
Time Complex = $\Theta(1)$

```
    def push(self, x)
        self.__queue.enqueue(x)
```
Time Complexity = $\Theta(1)$

```
    def pop(self)
        val_to_pop = none
        new_queue = Q()
        for i in range(self.__queue.size())
            if i == self.__queue.size():
                val_to_pop = self.queue.dequeue()
            else:
                new_queue.enqueue(self.__queue.dequeue())
        self.__queue = new_queue
        return val_to_pop
```
Time Complexity = $\Theta(n)$

$\sum_{k=0}^{n} i$