Julian Garcia
CSE122: HW3
2/23/19

1. Show $T(n) = 5n^4 + 6n^2 + 2n + 4$ is $O(n^4)$ using the definition of big O. That is find a c and a n_0.

- $5n^4 + 6n^2 + 2n + 4 \leq 5n^4 + 6n^4 + 2n^4 + 4n^4 = 17n^4$
- $C = 17$, and $n_0 = 1$
- $T(n) = O(n^4)$

2. Show $1^2 + 2^2 + \cdots + n^2$ is $O(n^3)$ using the definition of big O.

- $1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^2 + n(2n+1)}{6} = \frac{2n^3 + 3n^2 + 2n}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{3}n$
- $C = \frac{1}{3}$, and $n_0 = 1$
- By definition of big O, $T(n) = O(n^3)$

3. If $T(n) = O(n\log n)$ what happens to the running time if you double n?

   $T(2n) = 2n\log 2n$

   So essentially the running time will be a little more than double that of just nlogn, since the logn part of the function has much less influence than the n.

4. Find the running time T(n) and the big O of the following:

```
for i = 2 to n - 1
   i = i * i
        break
```

	Cost	# of Times
`for i = 2 to n - 1`	C_1	(n - 1 - 2 +1) + 1 = n - 1
`i = i * i`	C_2	n - 2
`break`	C_3	Loop executes once, n doesn't matter

$T(n) = C_1(n - 1) + C_2(n - 1) + C_3(n - 2)$
$T(n) = C_1 + C_2 + C_3$
$T(n) = O(1)$

5. Find the running time T(n) and the big O of the following:

```
for i = 1 to n
   for j = 1 to i
```

```
        for k = 1 to j
             x = i * j * k
```

	Cost	# of Times
`for i = 1 to n`	C_1	(n - 0 - 1 +1) + 1 = n + 1
`for j = 1 to i`	C_2	1+2+...+(n+1) = n((n+1)/2)
`for k = 1 to j`	C_3	(n+2)(n+1)n/6
`x = i * j * k`	C_4	(n+2)(n+1)n/6

$T(n) = C_1(n + 1) + C_2n((n+1)/2) + C_3(n + 2)(n + 1)n/6 + C_4(n + 2)(n + 1)n/6$
$T(n) = O(n^3)$

6. What is the big O of the term-term by polynomial evaluation?

```
//The  output  of this is p,
//the  polynomial  evaluated  at x.
p = c0
for i = 1 to n
    term = ci
   for j = 1 to i
        term = term *x
   p = p + term
```

	Cost	# of Times
$p = c_0$	C_1	1
`for i = 1 to n`	C_2	(n - 0 - 1 +1) + 1 = n + 1
$term = c_i$	C_3	n
`for j = 1 to i`	C_4	1+2+...+(n+1) = n((n+1)/2)

`term = term *x`	C_5	n((n+1)/2)
`p = p + term`	C_6	n

$T(n) = C_1 + C_2n + C_3n + C_4n((n+1)/2) + C_5n((n+1)/2) + C_6n$

$T(n) = O(n^2)$

7. What is the big O of polynomial evaluation by Horner's Rule

```
//The  output  of this is p,
//the  polynomial  evaluated  at x.
p = 0
for (i = n; i > 0; i--)
     p = x·(p + c_i)
return p + c_0
```

	Cost	# of Times
`p = 0`	C_1	1
`for (i = n; i > 0; i--)`	C_2	(n - 0 - 1 +1) + 1 = n + 1
`p = x·(p + c_i)`	C_3	n
`return p + c_0`	C_4	n

$T(n) = C_1 + C_2(n+1) + C_3(n) + C_4(n)$

$T(n) = O(n)$

8. Which is the more efficient algorithm to evaluate polynomials? Term-by-term or Horner's Rule?

Horner's rule is the more efficient algorithm since it is linear whereas the term-by-term method is quadratic. Codewise, this is explained by the fact that we need two for loops to evaluate polynomials term by term.

Julian Garcia CSE122: HW3
2/23/19

9. Find the T(n) and big O for the best and worst case of the following code. What can you say about the average case?

```
//array indexing begins with 1. length of array is n
for i = 1 to n - 1
   for j = i to n
       if (a[j] > a[i])
           tmp = a[i]
           a[i] = a[j]
           a[j] = tmp
```

	Cost	# of Times
`for i = 1 to n - 1`	C_1	(n - 1 - 1 +1) + 1 = n
`for j = i to n`	C_2	1+2+...+(n+1) = n((n+1)/2)
`if (a[j] > a[i])`	C_3	n((n+1)/2)
`tmp = a[i]`	C_4	n((n+1)/2)
`a[i] = a[j]`	C_5	n((n+1)/2)
`a[j] = tmp`	C_6	n((n+1)/2)

Best Case loop finds that a[j] > a[i]:
$T(n) = C_1n + C_2n((n+1)/2) + C_3n((n+1)/2)$
$O(n^2)$
Worst Case loop has to sort complete list:
$T(n) = C_1n + C_2n((n+1)/2) + C_3n((n+1)/2) + C_4n((n+1)/2) + C_5n((n+1)/2) + C_6n((n+1)/2)$
$O(n^2)$

The average case is still n^2 because, in the worst case and the best case, the biggest n would still be quadratic.

Julian Garcia CSE122: HW3
2/23/19

10. Find T(n) and the big O for the best and worst case of the selection sort algorithm. What can you say about the average case?

```
//assume you are sorting an array of length n. First element has index 1.
//selection sort produces an array of sorted integers in ascending order
for k = 1 to n - 1
  indexOfMin = k
  for i = k + 1 to n
       if (a[i] < a[indexOfMin])
           indexOfMin = i

  if (indexOfMin != k)
      tmp = a[k]
      a[k] = a[indexOfMin]
      a[indexOfMin] = tmp
```

	Cost	# of Times
`for k = 1 to n - 1`	C_1	(n - 1 - 1 +1) + 1 = n
`indexOfMin = k`	C_2	n - 1
`for i = k + 1 to n`	C_3	n((n+1)/2)
`if (a[i] < a[indexOfMin])`	C_4	n((n+1)/2)
`indexOfMin = i`	C_5	n((n+1)/2)
`if (indexOfMin !=k)`	C_6	n - 1
`tmp = a[k]`	C_7	n - 1
`a[k] = a[indexOfMin]`	C_8	n - 1
`a[indexOfMin] = tmp`	C_9	n - 1

Best Case loop finds that the index is sorted, so a [i] is never bigger than a [indexofMin]:

$T(n) = C_1n + C_2(n-1) + C_3n((n+1)/2) + C_4n((n+1)/2) + C_5n((n+1)/2) + C_6n((n+1)/2)$

$O(n^2)$

Worst Case, list is sorted in reverse:

Julian Garcia CSE122: HW3
2/23/19

$T(n) = C_1n + C_2(n-1) + C_3n((n+1)/2) + C_4n((n+1)/2) + C_5n((n+1)/2) + C_6n((n+1)/2) + C_7(n-1) + C_8(n-1) + C_9(n - 1)$

$O(n^2)$

The average case is still n^2 because, in the worst case and in the best case, the biggest n would still be quadratic.

11. Solve (find an explicit formula) the following recurrence relation for the running time T(n) using the substitution method:

$$T(n) = \begin{cases} a & if\ n = 1 \\ T(n/2) + b & if\ n \geq 2 \end{cases}$$

a and b are constants.
What is the big O of the running time? Show your work. Clearly show what T(n) is after k unrollings.

T(1) = a

T(n) = T(n/2) + n
T(n/2) = T(n/4) + b
T(n) = (T(n/4) + b) + b = T(n/4) + 2b
T(n/4) = T(n/8) + b
T(n) = ((T(n/8) + b) + b) + b
T(n) = T(n/8) + 3b
$T(n) = T(n/2^k) + kb$

Assume $n = 2^k$
Then k = log(n)
Substituting, we get: T(n) = a + log(n)
Therefore, T(n) = O(log(n))

12. Solve the following recurrence relation for the running time T(n) using the substitution method:

Julian Garcia CSE122: HW3
2/23/19

$$T(n) = \begin{cases} a & if\ n = 1 \\ 2T(n/2) + b & if\ n \geq 2 \end{cases}$$

a and b are constants.
Show your work. Clearly show what T(n) is after j unrollings.

$T(1) = a$

$T(n) = 2T(n/2) + n$
$T(n/2) = 2T(n/4) + b$
$T(n) = 2(2T(n/4) + b) + b = 4T(n/4) + 3b$
$T(n/4) = 2T(n/8) + b$
$T(n) = 2(2 (2T(n/8) + b) + b) + b$
$T(n) = 8T(n/8) + 7b$
$T(n) = 2^kT(n/2^k) + (2^k - 1)b$

Assume $n = 2^k$
Then $k = \log_2(n)$
Substituting, we get: $T(n) = 2^{\log(n)}a + (2^{\log(n)} - 1)b = 2(n)a + (n - 1)$
Therefore, T(n) = O(n)

13. Show for the following recurrence relation for the running time T(n) is O(n^{k+1}). Use the substitution method:

$$T(n) = \begin{cases} a & if\ n = 1 \\ T(n-i) + n^k & if\ n > 1 \end{cases}$$

a and k are constants.
What is the big O of the running time? Show your work. Clearly show what T(n) is after i Unrollings.

$T(1) = a$

$T(n) = T(n - i) + n^k$
$T(n - i) = T(n - 2i) + n^k$
$T(n) = (T(n - 2i) + n^k) + n^k = T(n - 2i) + 2n^k$
$T(n - 2i) = T(n - 3i) + n^k$
$T(n) = ((T(n - 3i) + n^k) + n^k) + n^k$

Julian Garcia CSE122: HW3
2/23/19

$T(n) = T(n - 3i) + 3n^k$
$T(n) = T(n - ki) + kn^k$

Assume n - ki = 0
Then k = n/i and n = ki
Substituting, we get: $T(n) = T(0) + n/i(n)^k$
Therefore, $T(n) = O(n^{k+1})$

14. Define f(n) = O(g(n)) to mean that $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$

Show that log n = $o(n^{\varepsilon})$ for any $\epsilon > 1$.
Show your work. Hint: use l'Hospital's Rule.

f(n)= log(n)
$g(n) = n^{\varepsilon}$

$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \log(n)/ n^{\varepsilon} = \frac{\infty}{\infty}$
Applying l'Hospital's Rule
$\lim_{n\to+\infty} \frac{f'(n)}{g'(n)} = (1/n)/(\epsilon n^{\varepsilon})$
Plugging in infinity:
$= 0/(\infty) = 0$

15. Assume your machine can do on the order 10^{12} ops per second and each n takes one operation. Given n and the order (big O) of the running time, find how many seconds it would take each algorithm to run on the machine.

n = 10

O(logn) = $3 * 10^{-12}$sec	-Under a second
O(n) = $1*10^{-11}$sec	-Under a second
O(n log n) = $3*10^{-11}$ sec	-Under a second
O(n^2) = $1*10^{-10}$sec	-Under a second
O(n^3) = $1*10^{-9}$ sec	-Under a second
O(2^n) = $1.024*10^{-9}$ sec	-Under a second
O(n!) = 3.6288×10^{-6}sec	-Under a second

n = 10^3

O(logn) = $9 * 10^{-12}$sec	-Under a second
O(n) = $1*10^{-9}$sec	-Under a second

Julian Garcia CSE122: HW3
2/23/19

$O(n \log n) = 9*10^{-9}$sec -Under a second
$O(n^2) = 1*10^{-6}$sec -Under a second
$O(n^3) = 0.001$sec -Under a second
$O(2^n) = 1.07*10^{289}$sec -NO HOPE
O(n!) = 4 × 10^2555sec -NO HOPE

$n = 10^6$
$O(\log n) = 1*10^{-11}$sec -Under a second
$O(n) = 1*10^{-6}$sec -Under a second
O(n log n) = 0.00001sec -Under a second
O(n^2) = 1 sec -Exact second
$O(n^3) = 1*10^6$sec -About 10 days(under 120 days)
O(2^n) = 9 × 10^301017sec -NO HOPE
O(n!) = 8 × 10^5565696 sec -NO HOPE

$n = 10^9$
$O(\log n) = 2 \times 10^{-11}$sec -Under a second
O(n) = 0.001 sec -Under a second
O(n log n) = 0.02 sec -Under a second
$O(n^2) = 1 \times 10^6$ sec -About 10 days(under 120 days)
O(n^3) = 1 × 10^15 sec -Eventually
$O(2^n) = (2^{10^9})/10^{12}$ sec -NO HOPE
O(n!) = 9.9 × 10^8565705510 sec -NO HOPE

$n = 10^{12}$
O(logn) = 3 × 10^-11 sec -Under a second
O(n) = 1 sec -Exact second
O(n log n) = 3 sec -Under an hour
O(n^2) = 1 × 10^12 sec -Eventually
O(n^3) = 1 × 10^24 sec -NO HOPE
$O(2^n) = 2^{10^{12}}/10^{12}$ sec -NO HOPE
O(n!) = 1.4 × 10^11565705518091 sec -NO HOPE

$n = 10^{15}$
O(logn) = 4× 10^-11sec -Under a second
O(n) = 1000sec -Under an hour
O(n log n) = 40000 sec -Under an hour
O(n^2) = 1 × 10^18 sec -NO HOPE
O(n^3) = 1 × 10^33 sec -NO HOPE

Julian Garcia CSE122: HW3
2/23/19

```
O(2^n) = 2^10^15/10^12 sec
O(n!) =  10^10^16 sec
```

-NO HOPE
-NO HOPE

16. Filled over table for readability/convenience.