CSE/IT 122: Homework 4

For problems involving running time, make a table of costs and the number of times each line runs. Have a column for line numbers. You do not need to show the work for the count, but your answer needs to be exact for each line. If a problem involves best case and a worst case scenario, make a column for each case.

Type your answers using Latex, or Word, or similar, and convert the file to a pdf file named `cse122_firstname_lastname_hw4a.pdf`.

Problems

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.
2. Show $1^2 + 2^2 + \cdots + n^2$ is $O(n^3)$ using the definition of Big O.
3. Show $2^{n+1} = O(2^n)$ using the definition of Big O.
4. If $T(n) = O(n \, log \, n)$ what happens to the running time if you double n?
5. 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. Fill in the following table:

n	$O(log \, n)$	$O(n)$	$O(n \, log \, n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$	$O(n!)$
10							
10^3							
10^6							
10^9							
10^{12}							
10^{15}							
10^{18}							
10^{21}							

The log is base 2. Use the approximation $10^3 = 2^{10}$ or $10 = 2^3$ to convert from base 10 to base 2 where necessary. For example, to calculate $10^3 log 10^3$, you do the following:

$$10^3 log_2 10^3 = 10^3(3 log_2 10) = 10^3(3 log_2 2^3) = 10^3(3 \cdot 3 log_2 2) = 9 \cdot 10^3$$

which is close enough to the (approximate) answer of 9965.78 for our purposes. Similarly, to find 2^{10^3} you do the following:

$$2^{10^3} = 2^{10 \cdot 10^2} = (2^{10})^{10^2} = (10^3)^{100} = 10^{300}$$

Where necessary, report coefficients to one digit. This is meant to be a hand exercise to get you familiar with manipulating exponents and using approximations. You shouldn't use a calculator except for factorials, which then you should use WolframAlpha. WolframAlpha will report things in both decimal and power of 10 representations. If there is a decimal representation use that and round to the nearest integer for the coefficient. Other wise report it as a power of 10 representation and for the exponent round to one decimal.

Rather than fill in a table, which can be difficult to format given the size of the exponents and power of 10 representations, "serialize" the table by reporting the table in columnar format:

```
n = 10
O(logn) =
O(n) =
O(n log n) =
O(n^2) =
O(n^3) =
O(2^n) =
O(n!) =

n = 10^3
...
```

For power of ten representations, use `10 ^ 10 ^ 10 ^ 1.5`, which is equivalent to $10^{10^{10^{1.5}}}$

6. For the times you found in the previous problem, and assuming that there are approximately 10^5 seconds in a day, 10^6 seconds in 10 days, 10^7 seconds in 120 days, $3 \cdot 10^7$ seconds in a year, and the known age of the universe is on the order of 10^{17} seconds, for what value of n and big O which algorithms will run under a second? in a hour? in a day? in 10 days? in 120 days? in a year? What algorithms have no hope of ever finishing? Type your answers.

For the next two problems assume you have a real polynomial of the form

$$\sum_{i=0}^{n} c_i x^i = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_2 x^2 + c_1 x + c_0$$

7. Type your answer. Find the order (big O) of the running time of term-term by polynomial evaluation

```
//The output of this is p,
//the polynomial evaluated at x.

p = c_0

for i = 1 to n
   term = c_i

  for j = 1 to i
      term = term * x

  p = p + term
```

8. Type your answer. Find the order (big O) of the running time 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
```

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

10. Find the order (big O) of the running time 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
```

11. Find the order (big O) of the running time 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
```

Submission

Upload your pdf file to Canvas before the due date.