CSE 122 MIDTERM REVIEW SPRING 2019

NMT CS DEPT.

An algorithm must be seen to be believed.

Donald Knuth

Programming is the art of algorithm design and the craft of debugging errant code.

Ellen Ullman

Useful formulas and facts

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

$$2^0 + 2^1 + \cdots + 2^n = 2^{n+1} - 1$$

$$10^7 seconds \approx 1 year$$

$$2^{10} \approx 10^3$$

(1) To show that $f(n) = O(g(n))$ you only need to find a pair of constants c and n_0 such that $|f(n)| \leq c|g(n)|$ when $n > n_0$.
Find the *big O* of the following by using the definition:

$$T(n) = 4n^3 + 2n^4 - 6n^5 - 4n^2 + 4$$

$$T(n) = 2n^2 + 2n^3 - 8n - 4n^4 + 3$$

$$T(n) = 5n^6 - 7n^4 - 3n^5 - 2n^2 - 12$$

Hint: That is, find a c and an n_0.

(2) You find the running time of the following algorithms are as follows:

$$T(n) = 3n + 6log(n) + 9nlog(n) + n!$$

$$T(n) = 3n + 6log(n)$$

$$T(n) = 3n + 6(n - 1)$$

$$T(n) = 3n + 9nlog(n)$$

What are the respective *big O* of $T(n)$?

(3) Assume you want to compute the summation $\sum_{i=1}^{n} 2^i = 2^{n+1} - 1$, would it be more efficient to calculate the sum (LHS of the equation) or to calculate the formula (RHS of the equation)? Why? What is the *big O* for the LHS and the RHS of the equation?

(4) Using induction, show $2^n < n!$. Make sure you clearly state the base case, the assumption, and what you are trying to prove.

(5) Determine and Solve the recurrence relation for the following code snippets. That is, find a formula for $T(n)$, a base case(s), and determine a *big O*.

```
long power(long x, long y){
        if(n == 0) return 1;
        if(n == 1) return x;
        if((n%2) == 0)
                return power(x, n/2) * power(x, n/2);
        else
                return power(x, n/2) * power(x, n/2) * x;
}
```

```
long power(long x, long y){
        if(n == 0) return 1;
        if(n == 1) return x;
        if((n%2) == 0)
                return power(x*x, n/2);
        else
                return power(x*x, n/2) * x;
}
```

(6) Find the *big O* of the following recurrence relation. Use substitution. **Show work and clearly show what $T(n)$ is after the k-th unrolling**. a and b are constants.
$T(n) = 2T(n-1) + b$
$T(1) = a$

(7) Find $T(n)$ for the best and worst case of the following insertion sort pseudo code. Fill in the table with the cost and count for each line and each case. **Be exact in your counts**.

```
1: for i ← 2 to n do
2:     key ← A[i]
3:     k ← i − 1
4:     while k > 0 and A[k] > key do
5:         A[k + 1] ← A[k]
6:         k ← k − 1
7:     A[k + 1] ← key
```

Line	Cost	Best Case	Worst Case
1			
2			
3			
4			
5			
6			
7			

(8) What is the *big O* for the best case in #7?

(9) What is the *big O* for the worst case in #7?

(10) What can you say about the *big O*'s average case in problem #7?

(11) What type of data produces the best case in problem #7?

(12) What type of data produces the worst case in problem #7?

(13) Order the following functions $2^n, n\,log\,n, n^n, log\,n, n^2, n!, n, n^3$ from least to greatest according to their growth rate.

(14) If you have a sorted array of elements, would you use a linear search or a binary search to find an element in the array? What is the *big O* of the algorithm you chose?

(15) What happens to the running time when you triple the input size of the following $T(n) = O(?)$ algorithm? It increases by a factor of what?

$T(n) = O(log(n))$
$T(n) = O(n)$
$T(n) = O(nlog(n))$
$T(n) = O(n^2)$
$T(n) = O(n!)$
$T(n) = O(2^n)$