Nora Sample Midterm

November 1, 2020 9:55 AM

1. What is the best asymptotic ("big-O") characterization of the following function:
$$f(n) = (14log n)^2 + \log(3^n)$$
   a) $O(3^n)$
   b) $O(n^2)$
   c) $O(n)$
   d) $O(2^n)$
   e) $O(log n)$

$f(n) = 14 \log n \cdot 14 \log n + n \log 3$

so we comparing $\log n$ and $n$

$n > \log n$ so $O(n)$

2. Give the best asymptotic ("big-Oh") characterization of the worst case *and* the best case time complexities of the algorithm `DoAgain(A, n)`.

```
Algorithm DoAgain(A,n)
Input: Array A storing integers and of size n > 1.
    sum ← 0
    for c ← 0 to n^2 do
        if A[0] < 0 then
            for k ← 0 to n − 1 do
                sum ← sum + c · A[k]
```

   (a) Best case $O(n)$ and worst case $O(n^2)$
   (b) Best case $O(n)$ and worst case $O(n^3)$
   (c) Best case $O(n^2)$ and worst case $O(n^3)$
   (d) Best case $O(n)$ and worst case $O(n^4)$
   (e) Best case $O(n^3)$ and worst case $O(n^4)$

Best case: 1st element is positive => outer loop runs $n^2$ times.
so $O(n^2)$

Worst case: outer loop runs $n^2$ times, inner runs $n-1$ times so $O(n^3)$

3. You have implemented the queue with a linked list, keeping track of a front node and a rear node with two reference variables. Which of these reference variables will change during an insertion into a NONEMPTY queue?
   a) Neither changes
   b) Only front changes.
   c) Only rear changes.
   d) Both change.

Queue is FiFo structure so new elements will be inserted at the end.
We would need to change both only if its an empty queue

4. Here is an incorrect kind of pseudo code a student provided for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:

```
declare a character stack
while ( more input is available)
{
   read a character
   if ( the character is a '(' )
      push it on the stack
   else if ( the character is a ')' and the stack is not empty )
      pop a character off the stack
   else
      print "unbalanced" and exit
}
print "balanced"
```

Which of these unbalanced sequences does the above code think is balanced?

   a) ((()
   b) ())(()
   c) ((())))
   d) ((())()

This is like the example Hanna showed in class. This algorithm is incorrect because it is not checking if the stack is empty or not. It will always return balanced
if 1st character is '(' we push to stack, none of the elses will trigger, so if we only have '(' to check. It returns balanced. So if we try running option A = "(((()"
0 = '(', push to stack, check next
1 = '(', push to stack, check next
2 = '(', push to stack check next (now stack is '(' '(' '(')
3 = ')', stack not empty remove last pushed '('
4 = ')', stack not empty remove last pushed '(' No more input
break loop, print balanced
option B = skip straight to unbalanced 1st input is empty.
C, D will pop non existing elements

5. Consider the algorithm *Multiply(A,n)* below. *A* is an array of size *n* storing integer values. What is the best characterization of the best and worst case asymptotic time complexity of the following algorithm?

```
Algorithm Multiply(A,n)
    result = 0
    for i = 0 to n/2 do
        j = 0
        while A[j] < 0 and j < n/2 do
            result = result+A[i]*A[j]
            j = j + 1
    return result
```

   a) Best case $O(1)$, worst case $O(n)$.
   b) Best case $O(log n)$, worst case $O(n)$.
   c) Best case $O(n)$, worst case $O(n)$.
   d) Best case $O(1)$, worst case $O(n^2)$.
   e) Best case $O(n)$, worst case $O(n^2)$.

Best case: outer loop runs $\frac{n}{2}$ times $O(n)$
Inner loop runs only once
so $O(n)$

Worst case: outer loop runs $\frac{n}{2}$ times $O(n)$
inner loop runs $\frac{n}{2}$ times $O(n)$
so $O(n^2)$

6. Consider the following pairs of functions: $f(n), g(n)$. For which pair, the functions are such that $f(n)$ *is* $O(g(n))$ *and* $g(n)$ ***is not*** $O(f(n))$?

   (a) $f(n) = n^2$, $g(n) = 2n^2 + 7$

$f(n)$ is $O(g(n))$ and $g(n)$ is not $O(f(n))$ would just change our definition to be strict inequality so $f(n) > g(n)$
a) $n^2 > n^2$ no
b) $\log n > 2 \log n$ no

6. Consider the following pairs of functions: $f(n), g(n)$. For which pair, the functions are such that $f(n)$ *is* $O(g(n))$ *and* $g(n)$ ***is not*** $O(f(n))$?

(a) $f(n) = n^2, g(n) = 2n^2 + 7$
(b) $f(n) = \log n, g(n) = \log(n^2)$
(c) $f(n) = 17, g(n) = n + 1$
(d) $f(n) = n, g(n) = 3000n + 1$
(e) $f(n) = \log n, g(n) = 1/n$

our definition to be strict inequality so $f(n) > g(n)$
a) $n^2 > n^2$ no
b) $\log n > 2\log n$ no
c) $1 > n$ no
d) $n > n$ no
e) $\log n > \frac{1}{n}$ yes

7. What is the best asymptotic ("big-O") characterization of the following function:
$f(n) = 6n\ log\ log(n^2) + 2nlog^2n + nlogn$

a) $O(nlog^2n)$
b) $O(nlogn)$
c) $O(n^2logn)$
d) $O(n^3logn)$
e) $O(nlog(n^2))$

$2n\log^2 n$ grows the fastest so thats our most dominant term, then $O(n\log^2 n)$

8. Suppose we have a circular array implementation of the queue class, with ten items in the queue stored at data [2] through data [11]. The current capacity is 42. Where does the insert method place the new entry in the array?

a) data[1]
b) data[2]
c) data[11]
d) data[12]

queue adds at end cuz FIFO

**B.1 [33 Points]** A palindrome is a string that reads the same forward and backward, capitalization and space are ignored. For example *deed, go dog, level*.... are palindromes.

a) **[10 Points]** Write an iterative algorithm in pseudo code that tests whether a string is a palindrome.

```
Algorithm palindromeItr (A)
Input : A is a string to be checked
output: True if A is a palindrome.
Arr ← A.toCharArray
  for each element in Arr do: //start from end (length-1) like for(int i= A.length-1; i>=0; i--){
      add element to new array B at index Element 1
new ← B.toString
return A.equals(new)
```

b) **[10 Points]** Write a recursive algorithm in pseudocode that tests whether a string is a palindrome.

```
Algorithm palindromeRecursive (A)
Input : A is a string to be checked
output: True if A is a palindrome.

if A.length is 1      // base case.
  return A
return palindromeRecursive (A.substring(1) + A.charAt(0))
```

c) **[8 Points]** Describe how you could use a *Stack* or *Queue* to check whether a string is a palindrome, and write the pseudo code for that.

```
Algorithm palindromeStack (A)
Input : A is a string to be checked
output: True if A is a palindrome.
 mid ← A.length/2
  loop till mid:
      push chars in stack.
  loop till length:
    pop(stuff).equals(current)
if one doesn't match return false
```

For each of the **4** questions in this part, mark **T** if the given statement is **ALWAYS** true. Otherwise mark **F** and **justify** your answer. If you do not justify the FALSE case you will lose $^4/_6$ of the mark. There is **no penalty** for selecting a wrong answer. **Hint**: a correct counter example and/or correct specification will give you better marks. A correct answer will get you **6** points.

1. If $f(n) = 5n^2$ then $f(n) \in \Omega(2^n)$ □T ☒F

$f(n) \geq c \cdot g(n)$

$5n^2 \geq c \cdot 2^n$ since we can't find a c to make that true, it's false.

2. The worst-case asymptotic running time for the best algorithm for finding something in a sorted array? is $O(n)$ □T ☒F

Best search is binary search which is $O(\log n)$ on a sorted array.

3. The worst-case asymptotic running time of finding and removing all values greater than 12 from a stack implemented with a linked-list (leaving the rest of the stack in its original order) is O(1) □T ☒F

$O(n)$ we need to run through the whole list to find stuff $>12$

4. Performing a remove operation at the tail in a list ADT implemented as a singly LinkedList that keeps track of the head, is very efficient: performed in a constant time O(1). □T ☒F

To remove last element, we move it to the second last element and the reference of the last element has to be set to null so $O(n)$