



Review Test Submission: Exam 1 (Winter 2014)

User	Eric S Rouse
Course	ANALYSIS OF ALGORITHMS (CS_325_400_W2014)
Test	Exam 1 (Winter 2014)
Started	2/9/14 9:56 AM
Submitted	2/9/14 11:09 AM
Due Date	2/9/14 11:59 PM
Status	Completed
Attempt Score	37.9 out of 39 points
Time Elapsed	1 hour, 12 minutes out of 1 hour and 50 minutes.
Instructions	Be sure you carefully think about the true-false questions and any multiple choice. They are not trick question, they are designed to test whether you can read a statement for what it is really asking and whether you can correctly answer in a concise manner. If you would like, then you can include an explanation to any and all of these in the free-form explanation question (I do not promise you will get more points for an incorrect answer, but I will go through your explanations to see if you understand the topics presented and give points accordingly).

Question 1

2 out of 2 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

n^2 is $O(n^3)$

Selected Answer: ☒ True

Answers: ☒ True
☐ False

Question 2

2 out of 2 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

$2^{(\log_2(n))}$ is big-theta(n).

Selected Answer: ☒ True

Answers: ☒ True
☐ False

Question 3

2 out of 2 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

if $f(n)$ is big-theta($g(n)$), then $g(n)$ is big-theta($f(n)$).

Selected Answer: ☒ True

Answers: ☒ True

☐ False

Question 4

2 out of 2 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

Algorithm A, with a running time of big-omega(n^2) has more run-time complexity (more expected theoretical running time) than algorithm B with a running time of big-theta(n^2).

Selected Answer: ☒ True

Answers: ☒ True

☐ False

Question 5

2 out of 2 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

If $f(n) / g(n)$ approaches 0 as n approaches infinity, then $f(n)$ is big-omega($g(n)$).

Selected Answer: ☒ False

Answers: ☐ True

☒ False

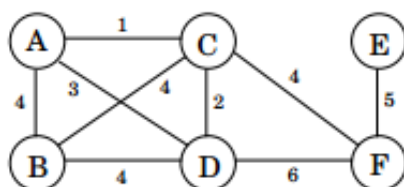
Question 6

3 out of 3 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

The minimum spanning tree of the graph below has weight 19.



Selected Answer: ☒ False

Answers: ☐ True

☒ False

Question 7

3 out of 3 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

The heaviest edge in a graph could be an element of its minimum spanning tree depending on the graph in question.

Selected Answer: ☒ True

Answers: ☒ True

☐ False

Question 8

3 out of 3 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

There are minimum weight spanning tree algorithms that work correctly even with negative weights.

Selected Answer: ☒ True

Answers: ☒ True

☐ False

Question 9

5 out of 5 points



Indicate whether the statement is true or false. You may explain your answer in the free-form explanation question.

If a recursive algorithm has two recursive calls on sub-problems, then the running-time of the algorithm is $O(n^2)$; where n is the size of the input.

Selected Answer: ☒ False

Answers: ☐ True

☒ False

Question 10

5 out of 5 points



For the following algorithm:

- Write a recurrence relation that describes the run-time in terms of n .
- Solve the recurrence relation to give an asymptotic bound on the run-time. Your bound should be as good as possible.

For your answer, you must respond in the following format:


- Show the recurrence in mathematical form on the first line of your response(such as: $T(n) = 27 * T(n^2) + c$),
- Show the final run-time bound on the second line of your response in Big-greek-letter notation (such as: $O(n^2)$ or Big-Theta($n * \log n$))
- Put a line of hyphens on a new line (to separate the concise answer from the explanation),
- Show your work on any lines after the separator so that partial credit **might** be awarded.

funCtion(n) //where n is a positive integer
if n > 1

```
    print("still going")
    funCtion(n / 3)
    funCtion(n / 3)
    funCtion(n / 3)
```

Selected Answer: $T(n) = 3 * T(n/3) + c$;
Big-Theta(n);

there is a print function that occurs everytime for some cost of time, c;
there are 3 functions executed recursively every time;
each function reduces the input by a third as new input for the recursive call;
hence, 3 funCtions of n/3 reduction and c time for printing, or $T(n) = 3 * T(n/3) + c$;
After k recurrnsions we are left with $3^k * T(n/3^k) + c * k$;
The depth, after k recursions, is $T(n/3^k) = T(1) \Rightarrow n/3^k = 1 \Rightarrow 3^k = n \Rightarrow \log_3(n) = k$;
substituting: $3^{\log_3(n)} * T(n/3^{\log_3(n)}) + c * \log_3(n)$;
reduction: $n * T(n/n) + c * \log_3(n) \Rightarrow n + c * \log_3(n)$;
since the n term dominates the logarithmic term, the bound is most closely n.
Since there is no "early opt-out" in the function, it will execute all those functions every time, making it a good Big-Theta bound rather than Big-Omega.
It could be thought of as Big-O, but Big-Theta implies more accuracy in the bound;

Correct 

Answer: $T(n) = 3 * T(n / 3) + c$ which is $\Theta(n)$; we may be using integer division, in which case some cases will execute in the same time as other (larger) values, but these are never off by more than a total amount equal to n, so a multiplicative constant of 2 looks like it would be enough to bound this to the values of this function.

Response Great!
Feedback:

Question 11

4.9 out of 5 points



For the following algorithm:

- Write a recurrence relation that describes the run-time in terms of n .
- Solve the recurrence relation to give an asymptotic bound on the run-time. Your bound should be as good as possible.

For your answer, you must respond in the following format:

- Show the recurrence in mathematical form on the first line of your response (such as: $T(n) = 27 * T(n^2) + c$),
- Show the final run-time bound on the second line of your response in Big-greek-letter notation (such as: $O(n^2)$ or Big-Theta($n * \log n$))
- Put a line of hyphens on a new line (to separate the concise answer from the explanation),
- Show your work on any lines after the separator so that partial credit **might** be awarded.


```
STOOGESORT(A[0...n-1])           % input is one array of length n
  if n = 2 and A[0] > A[1]
    swap A[0] and A[1]
  else if n > 2
    k = [2n/3]
    STOOGESORT(A[0...k-1])
    STOOGESORT(A[n-k...n-1])
    STOOGESORT(A[0...k-1])
```

Selected $T(n) = 3 * T((2/3) * n) + c$;

Answer: Big-Theta ($3^{\log_{2/3}(n)}$)

-----;

there is an if/else operation that occurs everytime for some cost of time, c ;
 there are 3 functions executed recursively every time;
 each function reduces the input by $2/3$ as new input for the recursive call;
 hence, 3 functions of $2n/3$ reduction and c time for printing, or $T(n) = 3 * T((2/3) * n) + c$;
 After k recursions we are left with $3^k * T(n * (2/3)^k) + c * k$;
 The depth, after k recursions, is $T(n * (2/3)^k) = T(1) \Rightarrow n * (2/3)^k = 1 \Rightarrow (2/3)^k = 1/n \Rightarrow \log_{2/3}(n) = k$;
 substituting: $3^{\log_{2/3}(n)} * T(n * (2/3)^{\log_{2/3}(n)}) + c * \log_{2/3}(n)$;
 reduction: $3^{\log_{2/3}(n)} + c * \log_{2/3}(n)$;
 The power term will dominate. All operations will occur, so it is big-theta not big-omega and it isn't necessarily an upper bound, thanks to the second term, hence not Big-O;

Correct Answer:  $T(n) = 3T(2n/3) + O(1)$ which is $\Theta(n^{\log_{3/2} 3})$ by the recursion tree method

Response Feedback: Good, though you could also use log-exponent laws to swap the 3 and the n :).

Question 12

3.5 out of 5 points



For the following algorithm:

- Write a recurrence relation that describes the run-time in terms of n .
- Solve the recurrence relation to give an asymptotic bound on the run-time. Your bound should be as good as possible.

For your answer, you must respond in the following format:

- Show the recurrence in mathematical form on the first line of your response (such as: $T(n) = 27 * T(n^2) + c$),
- Show the final run-time bound on the second line of your response in Big-greek-letter notation (such as: $O(n^2)$ or Big-Theta($n * \log n$))
- Put a line of hyphens on a new line (to separate the concise answer from the explanation),
- Show your work on any lines after the separator so that partial credit **might** be awarded.

This algorithm does not return anything but updates two arrays called *left* and *right* which you may assume are stored globally.


```

TREEIFY(pre[1, ..., n], post[1, ..., n])           % input is two arrays of length n
  if pre and post are empty, do nothing
  otherwise
    find i such that post[i] = pre[2] (by linear search)
    left[pre[1]] = TREEIFY(pre[1, ..., i], post[1, ..., i])
    right[pre[1]] = TREEIFY(pre[i + 1, ..., n - 1], post[i + 1, ..., n - 1])

```

Selected Answer: $T(n) = 2T(r*n) + n$, where r is some random ratio determined by linear search;
Big-Omega($n * \log r(n)$), where r (the base of the log) is some random ratio determined by linear search;

There is a linear search for every operation, costing n time;
Then, we recursively call two functions and then operate on a random reduction of n on those two things;
hence, 3 operations of $r*n$ reduction and n time for printing, or $T(n) = 2*T((r)*n) + n$;
After k recursions we are left with $2^k * T(n*(r)^k) + n*k$;
Solve depth for k , $k = \log r(n)$;
 $2^{\log r(n)} * T(n/r^{\log r(n)}) + n * \log r(n)$;
 $2^{\log r(n)} + n * \log r(n)$;
I think the $n * \log r(n)$ will dominate here, and since this problem is somewhat random it could possible end early. So, big-Omega.

Correct Answer:  $T(n) = T(a) + T(b) + cn$ where $a + b = n - 1$. In the worst case, the depth of the recursion tree is $O(n)$ the total time is proportional to: $c * \sum_{i=1}^n (i)$, which is $O(n^2)$. Note that the depth of the recursion tree could be as small as $\log n$ if $a \approx b$.

Response Feedback: Are you sure the calls are always the same size? What happens if the element we are looking for is at the end of the list each time? How does this affect each call time, depth of recursion, and then total run-time?

Question 13



0.5 out of 1 points (Extra Credit)
 This is your explanation question. You will use this space to explain your answers from above. You may provide an explanation to any or all of the above questions that did not allow typed responses (all true-false for this version of the exam I believe).

This is your chance to get partial credit for your answers above by providing an explanation that shows that you have some understanding of the relevant questions.

Please be sure to make it obvious which question your explanation is related to, and **separate your explanations by a newline with a single space on it** (in case blackboard removes your blank lines like it does mine...). Improper spacing after this warning will be met with potentially grumpy grading (appearance of capability is part of being capable of doing the work, even if just in simple formatting issues).

Selected Answer: Q1 - Since big O is an upper bound, anything above $O(n^2)$ is fair;
 Q2 - Since $2^{\log_2(n)}$ reduces to n ;
 Q3 - By definition of big theta;
 Q4 - Because big omega is a lower bound, or best case, the actual complexity could be much higher. Big theta is more of an "actual" bound so we expect the algorithm to perform very closely to n^2 ;
 Q5 - All that $f(n)/g(n)$ going to 0 as n goes to infinity tells us is that $f(n)$ is smaller than $g(n)$. It is possible that $f(n)$ is the big omega of $g(n)$, but not necessary;
 Q6 - The MST is A-C(1), C-D(2), D-B(4), C-F(4), F-E(5) or $1+2+4+4+5 = 16$;
 Q7 - If there is a vertex connected by a single edge and that edge is the highest weight.
 Q8 - I don't remember which ones exactly, but I think Dijkstra and Prim both do.
 Q9 - Need more information to be absolutely sure, but since the question says sub-problems, we must infer that the size operated on is reduced every time, implying a more logarithmic complexity rather than exponential.

Correct Answer: [None]

Response Feedback: Q9. sub-problems does not necessarily mean the work is not something silly like $n - 1$, and this does not even touch on the non-recursive part of the problem.

Monday, March 17, 2014 9:41:30 AM PDT

← OK