EECS2101 M Midterm 2025

Instructor: Jeff Edmonds

Fill out the bubble sheets at the end of the exam.
Even if you have to guess, answer EVERY question.
If you guess a True/False question, you will get half right.
As such, 50% may or may not be a passing grade.
It will depend on how other people do.
There are $80/72 = 1$ minutes per question.
Be sure to hand in the question pages as well as the bouble sheets
(I will know and be unhappy if you don't)
Yes = a
No = b

Software Engineering

For the next questions consider the following options:
(a) Implementation
(b) Abstract Data Types (ADTs)
(c) Method
(d) Algorithmic Paradigm
(e) System Invariant

1. **b:** A Stack is an example of?
2. **a:** A Linked List is an example of?
3. **e:** "*top* points at the first node" is an example of?

For the next questions consider the following options:
(a) Encapsulation/Modularity
(b) Adaptable
(c) Efficient
(d) Robust
(e) Not a software Engineering Principle taught

4. **e:** It is important for the coders on the team to understand the details of each other's code.
5. **a:** Each object reveals only what other objects need to see.
6. **b:** Our algorithm sorts objects of type $\langle E \rangle$.
7. **c:** Merge sort is better than Insertion sort.
8. **a:** Mary wrote the stack routines and Ann used them when writing a compiler.

Positions and Pointer

Consider a linked list. Nodes have the fields *next* and *value*. For the next questions which code does the trick:
(a) $current.next = current$;
(b) $current = current.next$;
(c) $next.value = 5$;
(d) $current.next.value = 5$;
(e) None of these

9. **b:** *current* walks to the next node.
10. **a:** The later nodes are dropped.
11. **e:** The current node's value is changed.
12. **d:** The node after the current node's value is changed.
13. **a:** The current node points to itself.
14. **c:** Won't compile.

Contracts and Invariants

For the next questions, what does the logic expression prove?
(a) Maintaining the Loop Invariant
(b) Establishing the Loop Invariant
(c) Obtaining the postcondition
(d) Ensuring Termination
(e) None of the above

15. **e:** $\langle pre{-}cond \rangle$ $\&code \Rightarrow$ $\langle post{-}cond \rangle$

16. **a:** $\langle loop{-}invariant' \rangle$ & not $\langle exit{-}cond \rangle$ & $code_{loop} \Rightarrow \langle loop{-}invariant'' \rangle$

17. **e:** Be sure to establish the loop invariant each iteration.

18. **d:** Make progress each iteration.

For the next questions, suppose we are computing $S = 1^2 + 2^2 + 3^2 + ... + I^2$.

19. **N:** Part of the loop invariant is that the next action is to add the next term to the partial sum s.

    - Answer: This is an action not a picture

20. **N:** Part of the code within the loop is $s = s + i^2$; $i = i + 1$

    - Answer: The LI states that the next to add is $s = s + (i+1)^2$; $i = i + 1$

21. **Y:** If we are trying establish the loop invariant, $i = 0$ is Jeff's favorite number.

22. **Y:** The measure of progress is i. Hence, in order to officially make progress, all we have to do increment i.

23. **Y:** $i = i + 1$ is code and $i_{t+1} = i_t + 1$ is math encoded in the logic $code_{loop}$ mentioned above.

24. **Y:** If $i = i + 1$ pushes us off the loop invariant path, then all we have to do is the minimum code to get us back on.

---

25. **N:** The loop invariant for binary search is that we have a narrowed search space and the key we are looking at is definitely in this narrowed space.

    - Answer: No. If the key is not contained in the entire list, then it can't be in the narrowed search space.

26. **Y:** $\log_2 n$ is (within one) the number of bits to write down n and the number of time n can be divided by two.

27. **Y:** A singularly linked list for a queue requires a top and bottom pointer?

    - Answer: Yes. A queue is accessed from both ends.

28. **Y:** When implementing a circular array, we move to the next position with $top = top + 1 \ mod \ n$.

29. **N:** One implementation of a stack is a circular array.

    - Answer: No. This was for a queue. As items are inserted and removed they rotate around.

Running Time

For the next questions give the running time of the operation:
(a) $\mathcal{O}(1)$
(b) $\mathcal{O}(\log n)$
(c) $\mathcal{O}(n)$
(d) $\mathcal{O}(2^n)$
(e) Can't be done

30. **c:** Inserting a new item into a known location in array, eg $[1, 2, 3, 4, 5] \rightarrow [1, 2, 3, 12, 4, 5]$.

- Answer: The other values in the array need to be shifted

31. **a:** Inserting a new item into a known location in linked list, eg $[1, 2, 3, 4, 5] \rightarrow [1, 2, 3, 12.4, 5]$.

    - Answer: Once the location is known with pointers *current* and *next* just a few pointers need to be moved.

32. **c:** Remove the last node in a singularly linked list with a top and bottom pointer.

    - Answer: We can remove the last node quickly, but to maintain the system invariant, you need to get bottom pointer to the second last node.

33. **b:** Finding a given item in a balanced *binary search tree*.

34. **c:** Finding a given item in an unbalanced *binary search tree*.

    - Answer: One path down the tree is followed but it might have length $\mathcal{O}(m)$.

35. **d:** Find an assignment on which a given circuit evaluates to true.

    - Answer: There are 2^n assignments to check.

For the next questions give the running time of the operation:
(a) $\mathcal{O}(1)$
(b) $\mathcal{O}(n)$
(c) $\mathcal{O}(n^2)$
(d) $\mathcal{O}(2^n)$
(e) Can't be done

36. **d:** The kindergarten algorithm for multiplying $a \times b = a + a + a + \ldots$

    - Answer: Remember the size of the input is $n = \log a + \log b$ and the running time is linear in b but exponential in n. I can give you $b = 1000000000000$ fast but you don't want to count to it.

37. **c:** The high school algorithm for multiplying, i.e. multiply each of the n digits of a with each of the n digits of b.

---

38. **N:** Every day I inflate one balloon and throw way each old balloon that has lost all of its air (and has not been thrown away previously). It is possible that on the i^{th} day, my work includes throwing away up to i balloons. Summing over all n dates gives that the total time could be as much as $\sum_{i=0..n} i = \mathcal{O}(n^2)$,

    - Answer: No. A balloon can only be thrown away once. Hence, the total work is $\mathcal{O}(n)$.

39. **Y:** The amortized work done in one day of the balloon problem is $\mathcal{O}(1)$.

    - Answer: Yes. The total work $\mathcal{O}(n)$ divided by the number of days n is $\mathcal{O}(1)$.

For the next questions consider the following setting. I am growing my data and the array holding it gets full and hence I need to copy the data into a bigger array.

40. **N:** If the array had been of size n, I should grow it to size $n+c$ for some constant c, so that the cost of making the array bigger is not more than $\mathcal{O}(1)$.

- Answer: No. The cost of making the array bigger is $\mathcal{O}(n)$.

41. **N:** If copying an array of size i cost $\mathcal{O}(i)$ and we do this for $i = 1, 2, 4, 8, 16, \ldots, n$, then the total cost will be $\mathcal{O}(n \log n)$.

    - Answer: No. The sum of $i = 1, 2, 4, 8, 16, \ldots, n$ is geometric and is dominated by the biggest term, ie sums to $\mathcal{O}(n)$

For the next questions, classify the function into one of the following complexity classes:
(a) $\Theta(1)$
(b) $\Theta(n^2)$
(c) $n^{\Theta(1)}$
(d) $2^{\Theta(n)}$
(e) none of these

42. **a:** $5 + sin(n^2)$

43. **d:** $8^n = (2^3)^n = 2^{3n}$.

44. **c:** Poly

---

45. **Y:** Does time complexity include $n^3 \log^{100}(n)$ in the class of polynomials?

46. **Y:** Is $n^3 \log^{100}(n)$ bounded between n^3 and n^4?

For the next questions, determine the sum:
(a) $\Theta(1)$
(b) $\Theta(n)$
(c) $\Theta(n^2)$
(d) $\Theta(n^3)$
(e) $\Theta(3^n)$

47. **d:** $\sum_{i=1..n} i^2$
    - Answer: Arithmetic = bounded by number of terms times the last term.
48. **a:** $\sum_{i=1..n} 3$.
    - Answer: Arithmetic = bounded by number of terms times the last term.

Consider the following recurrence relations
(a) $T(n) = 2T(n/8) + n^5$
(b) $T(n) = 2T(n/2) + n$
(c) $T(n) = 1T(n/2) + 1$
(d) $T(n) = 8T(n/2) + n^5$
(e) $T(n) = 2T(n-1) + 1$

49. **c:** Which is the recurrence relation for binary search with $T(n) = \Theta(\log n)$?
50. **b:** Which is the recurrence relation for merge sort with $T(n) = \Theta(n \log n)$?
51. **e:** Which is the recurrence relation for Towers of Hanoi with $T(n) = 2^{\Theta(n)}$?
52. **d:** Which is the recurrence relation for the code

algorithm $OddRecursion(x_1, x_2, \ldots, x_n)$
⟨*pre−cond*⟩: n integers.
⟨*post−cond*⟩: $T(n)$ "Hi"s are printed

```
begin
    if( n ≤ 1 )
        Print("Hi")
    else
        loop i = 1...n^5
            Print("Hi")
        end loop
        loop i = 1...8
            OddRecursion(x_1, x_2, ..., x_{n/2})
        end loop
    end if
end algorithm
```

53. **c:** In the previous code, the number of base cases is
    (a) $\mathcal{O}(n^8)$
    (b) $\mathcal{O}(n^{3/2})$
    (c) $\mathcal{O}(n^3)$
    (d) $\mathcal{O}(n^5)$
    (e) none of these
    - Answer: The number of base cases is $\mathcal{O}(n^{\log a/\log b}) = \mathcal{O}(n^{\log_2 8/\log_2 2}) = \mathcal{O}(n^{3/1})$
54. **N:** The solution to a recurrence relation has an extra $\log n$ factor, eg $T(n) = \Theta(n \log n)$, when the

value n keeps getting divided by 2.

- Answer: No. The extra $\log n$ is added when $\log a / \log b = c$ because then all $\mathcal{O}(\log n)$ layers require the same amount of work.

---

55. **c:** Which statement below says "Problem P is computable".
(a) $\forall$ inputs I, $\exists$ algorithm A, $A(I) = P(I)$.
(b) $\exists$ inputs I, $\forall$ algorithm A, $A(I) = P(I)$.
(c) $\exists$ algorithm A, $\forall$ inputs I, $A(I) = P(I)$.
(d) $\forall$ algorithm A, $\exists$ inputs I, $A(I) = P(I)$.
(e) none of these

- Answer: You need one algorithm that works for all inputs

Recursion

56. **e:** Which of the following statements is FALSE about a recursive algorithm for filling a knapsack? Let $Knapsack(n, v)$ denote the instance when we are trying to put the first n objects into a knapsack of volume v.
(a) When my bird says to put the n^{th} object into the knapsack, I ask my friend $Knapsack(n-1, v-v_n)$ and add the price p_n of this object to my friend's solution.
(b) When my bird says to not put the n^{th} object into the knapsack, I ask my friend $Knapsack(n-1, v)$ and return my friend's solution unmodified.
(c) One of these birds is right, but I don't know which.
(d) So I just try both friend's answers and take the best of the resulting answers.
(e) All of these lines are correct.

57. **d:** Which of the following statements is CORRECT about a recursive algorithm for filling a knapsack?
(a) If you want to know complete set of possible solution, return union of the results from the two different bird answers yes/no.
(b) If you want to know the number of possible solutions, return sum of these two.
(c) If you want to know boolean whether there is a solution, return OR of these two.
(d) Lines a-c are correct.
(e) Lines a-c are false.

58. **c:** The following are statements about what you needed to change in the knapsack problem code to solve the problems in your your first assignment. Which of them is FALSE?
(a) To solve subset sum (i.e. find a subset of the values that sums to the target), drop the prices p_i of the knapsack objects.
(b) To solve subset sum, the base case in which there is volume left but no objects left should return that this was not a proper solution.
(c) All that stuff about the bird does not apply apply to subset sum.
(d) For the stair climbing problem, the bird tells you how many stairs to take in one step. All such bird answers are tried. For each, a friend is asked to help.
(e) All of these statements are true.

- Answer: Jeff likes the bird. Her answers represent the set of things that a stack frame tries.

59. **b:** Which of the following lines is NOT needed (according to Jeff) when describing merge sort?
(a) If the given list of numbers contains zero or one number then consider it sorted.
Otherwise, give one friend the first half of the list and another friend the second half.
(b) Each of these friends splits their list and sends each half to a friend.
(c) My friends give me their lists sorted.
(d) I merge their sorted lists together to get my sorted list.
(e) All of these lines are needed.

- Answer: (b) is micro managing your friend, which Jeff says gets too confusing.

60. **e:** Which of the following statements is FALSE about the running time of merge sort?
(a) If you follow the value 7 in the input $[3, 2, 6, 7, 1, 5, 4, 8]$, it gets looked at when the input narrows to $[3, 2, 6, 7]$, again when the input narrows to $[6, 7]$, and again when the input narrows to $[7]$.
(b) As such the value 7 gets looked at $\log n$ times.
(c) This is true for every number.
(d) This is why the running time is $\Theta(n \log n)$.
(e) All of these lines are correct.

61. **Y:** If the input to the recursive algorithm is $\langle x, y \rangle = \langle 3, 10 \rangle$, then it is ok to recurse on the instance $\langle 10000, 9 \rangle$ as long as the base case is "if($y \leq 0$) return(0)".

    - Answer: The instance passed to the friend needs to be smaller, but we can define "size" to be the value y. Hence, it does not matter that x has gotten bigger.

62. **N:** If the postcondition of a $\mathcal{O}(n)$ time recursive algorithm is that it returns yes/no whether or not the given tree is balanced, then it is ok to ask a friend the depth of the left subtree.

    - Answer: No. The subinstance passed to your recursive friend needs to meet the same pre/post condition.

63. **N:** If the postcondition of a $\mathcal{O}(n)$ time recursive algorithm is that it returns yes/no whether or not the given tree is balanced, then it is ok to call a subroutine that returns the depth of the left subtree.

    - Answer: No. Each stack frame of a $\mathcal{O}(n)$ time recursive algorithm can only spent $\mathcal{O}(1)$ time. Such a subroutine call would take $\mathcal{O}(n)$ time.

64. **c:** Which of the following lines is not needed (according to Jeff) in a recursive algorithm for counting the total number of nodes in a binary tree.
(a) If the given tree is the empty tree then the answer is zero.
(b) Give the left child to one friend and the right child to another friend.
Each friend returns the number of nodes in their subtree.
(c) Similarly, handle the cases when the root node does not have a left or right child or no children at all.
(d) I return the sum of their numbers plus one.
(e) All of these lines are needed.

    - Answer: Jeff does not like code that has all the cases in (c), because they are all handled by the same code if the base case is the empty tree.

65. **a:** Which of the following lines is not needed (according to Jeff) for a recursive algorithm for counting the total number of nodes in a nonempty multi-child tree.
(a) If the given tree is the empty tree then the answer is zero.
(b) Give each child subtree to a separate friend.
(c) Each friend returns the number of nodes in their subtree.
(d) I return the sum of their numbers plus one.
(e) All of these lines are needed.

    - Answer: Sorry, this is a hard one. The base case (a) is not needed because when a node has zero children, it will recurse zero times, and effectively becomes a base case. It only recurses on actual children, not on the empty tree.

66. **c:** The height of a binary tree is given by:
(a) return(Height(tree.left) + Height(tree.right) + 1)
(b) return(max(Height(tree.left), Height(tree.right)))

(c) return(max(Height(tree.left), Height(tree.right)) + 1)
(d) return(Height(tree.left) OR Height(tree.right))
(e) None of these lines work.

67. **e:** Which of the following lines is NOT needed (according to Jeff) when computing the number of leaves in a binary tree?
(a) If the given tree is empty return 0.
(b) If the given tree is consists of just one node return 1.
(c) Otherwise, give one friend the left subtree and the other the right and return the sum of their answers.
(d) This code does not work.
(e) All of these lines are needed.

- Answer: You need the extra single node base case so that your count increases from 0.

Paradigms

68. **N:** An *iterative* algorithm follows a steady well predicted path from the pre-condition to the post-condition.

69. **Y:** A *recursive* algorithm asks his friends any instance that is smaller and meets the precondition. It is best not to micro managing these friends.

70. **Y:** A *recursive backtracking* algorithm tries various things. For each thing tried it recurses. Then it backtracks and tries something different.

Binary Search Trees

71. **Y:** The system invariant of binary search trees is that for every node, all the values in the node's left subtree is less or equal to the value at the node and every value in it's right subtree is bigger or equal.

72. **Y:** AVL trees are "mostly" balanced. The invariant is that for each node the heights of their siblings differ by at most 1. This guarantees that the height is $\mathcal{O}(\log n)$