Quizlet

Week 7

Study online at quizlet.com/_8rj1g6

1. **If you're lookin at a SLL or DLL with the head & tail equaling the same node, what does that tell you?**

   That the length of the list 1.

2. **Of the two swapping sorts, which is the most efficient when write speeds are limited?`**

   Selection sort, because Bubble will swap every value in each comparison, where Selection will only sort after each inner loop is completed.

   Both are O(n^2) time complexity & O(1) space complexity.

3. **Rank the following from least to most complex:**
   O(n)
   O(n!)
   O(1)
   O(n log n)
   O(c^n)
   O(log n)
   O(n^2)

   1. O(1) - constant
   2. O(log n) - logarithmic
   3. O(n) - linear
   4. O(n log n) - loglinear, linearithmic, quasilinear
   5. O(n^2) - polynomial
   6. O(c^n) - exponential
   7. O(n!) - factorial

4. **What are the costs of Merge Sort? Benefits?**

   The major cost is space - it has O(n) space complexity.

   However, it has O(n log n) time complexity, so it's much faster than bubble, selection, or insertion sort

5. **What are the methods of a Queue?**
   What do they do?

   Enqueue(Insertion): Adds a Node to the front of the Queue. Returns an Integer - New size of Queue
   Dequeue(Deletion): Removes a Node from the front of the Queue. Returns the Node removed from front of Queue.
   Size: Returns the current size of the Queue. Returns an Integer.

6. **What are the methods of a Stack?**
   What do they do?

   Push(Insertion): Adds a Node to the top of the Stack. Returns an Integer - New size of stack
   Pop(Deletion): Removes a Node from the top of the Stack. Returns the Node removed from top of Stack
   Size: Returns the current size of the Stack. Returns an Integer.

7. **What are the names of the complexity classes?**

   Constant, O(1)
   Logarithmic O(log n)
   Linear O(n)
   Loglinear O(n log n)
   Polynomial O(n^2)
   Exponential O(c^n)
   Factorial O(!n)

8. **What are the two most easily confused complexity types?**

   Which is worse?

   Polynomial, O(n^2), and exponential O(2^n)

   Exponential is worse.

9. **What complexity does O(3^n) represent?**

   Exponential

10. **What defines a doubly-linked list?**

    Nodes have two pointers connecting them bi-directionally (`.previous` and `.next`).

11. **What defines a singly-linked list?**

    Nodes have a single pointer connecting them in a single direction (`.next`)

12. **What is a Linked List?**

    A Linked List data structure represents a linear sequence of "vertices" (or "nodes").

13. **What is the Big-O of Binary Search?**

    Time Complexity: O(log(n))
    The number of recursive calls is the number of times we must halve the array until its length becomes 0.
    Space Complexity: O(n)
    Our implementation uses n space due to half arrays we create using slice.

14. **What is the Big-O of Bubble Sort?**

    Time Complexity: O(n^2)
    The inner for loop contributes O(n) in isolation. In the worst case scenario, the while loop will need to run n times to bring all n elements into their final resting positions.
    Space Complexity: O(1)
    Bubble sort uses the same amount of memory and create the same amount of variables regardless of the size of the input

15. **What is the Big-O of Insertion Sort?**

Time Complexity: O(n^2)
The outer loop i contributes O(n) in isolation. The inner while loop will contribute roughly O(n / 2) on average. The two loops are nested so our total time complexity is O(n * n / 2) = O(n^2).
Space Complexity: O(1)
We use the same amount of memory and create the same amount of variables regardless of the size of our input.

16. **What is the Big-O of Merge Sort?**

Time Complexity: O(n log(n))
Since we split the array in half each time, the number of recursive calls is O(log(n)). The while loop within the merge function contributes O(n) in isolation and we call that for every recursive mergeSort call.
Space Complexity: O(n)
We will create a new subarray for each element in the original input.

17. **What is the Big-O of Quick Sort?**

Avg Case: O(n log(n))
The partition step alone is O(n). We are lucky and always choose the median as the pivot. This will halve the array length at every step of the recursion O(log(n)).
Worst Case: O(n2)
We are unlucky and always choose the min or max as the pivot. This means one partition will contain everything, and the other partition is empty O(n).
Space Complexity: O(n)
Our implementation of quickSort uses O(n) space because of the partition arrays we create.

18. **What is the Big-O of Selection Sort?**

Time Complexity: O(n^2)
The outer loop i contributes O(n) in isolation. The inner loop j will contribute roughly O(n / 2) on average. The two loops are nested so our total time complexity is O(n * n / 2) = O(n^2).
Space Complexity: O(1)
We use the same amount of memory and create the same amount of variables regardless of the size of our input.

19. **What is the Big-O Simplify Products rule?**

if the function is a product of many terms, we drop the terms that don't depend on the size of the input. O(2n) => O(n)

20. **What is the Big-O Simplify Sums rule?**

if the function is a sum of many terms, we keep the term with the largest growth rate and drop the other terms. O(n^2 + n) => O(n^2)

21. **What is the complexity of a function with a nested loop?**

Typically polynomial, O(n^2)?

22. **What is the least complex Big O?**

Constant, O(1), followed closely by logarithmic, O(log n)

23. **What sort type works through swapping, then ordering, the position of elements?**

Bubble sort

24. **What sort works by identifying the middles index, then splitting table, then repeating until there are arrays containg each value of the original array?**

Merge Sort

25. **What's the worst complexity? When would you use that?**

Factorial

Literally never! It's the worst, by far.

26. **What type of sort works through iterating through the unsorted region, finding the min, and swapping it with the first value?**

Selection sort

27. **When building a constructor for a doubly linked list, what are its' properties?**

DLL Properties:
this.head
this.tail
this.length

DLL Node Properties:
this.value
this.next
this.previous

28. **When building a constructor for a singly linked list, what are its' properties? What about the node constructor for a SLL**

SLL Properties:
this.head
this.tail
this.length

SLL Node Properties:
this.value
this.next

29. **When do we use Binary Search?**

The input data is sorted!

30. **When working with stacks & queue's, what is the Big-O space complexity of their insertion & deletion?
How do they achieve that?**

O(n)

In either a stack or a queue, there are no indices, so to find or access any value, you must traverse the entire stack/queue's nodes.
Queue's have some advantage when ordered, because you have the option to start at the front or back & traverse.

31. **When working with stacks & queue's, what is the Big-O time complexity of their insertion & deletion?
How do they achieve that?**

O(1) time complexity.

They both have a constant reference to the front or back, which allows them to make additions or deletions in a single action.

32. **When would we use Quick Sort?**

When we need an easy to write, relatively efficient, sort, and especially if we know our array is already sorted to some predictable degree.

In the worst case, where we grab a number that happens to be the min or max value of the table, the time complexity O(n^2), but best case is O(n * log(n))>
The worst case is exceedingly rare in actual practice.

33. **Which data structure allows deleting data elements from front and inserting at back? What 'out' structure does it have?**

Queue

FIFO

34. **Which data structure allows deleting & inserting elements from the front? What 'out' structure does it have?**

Stack

LIFO

35. **Which sort works by splitting the array around a pivot point & filtering the two remaining arrays?**

Quick Sort

36. **Which sort works by splitting the array around & mid point, comparing the value to the first value of the 'upper' array & 'lower' array, then repeating?**

Binary Search