1. Choose the fastest runtime from the following recurrences:
   1. T(n) = 3T(n/2) + n = way worse than (b)
   2. T(n) = 2T(n/2) + n = O(n log(n))
   3. T(n) = 2T(n/2) + 1 = way better than (b)
   4. T(n) = T(n-1) + n = O(n^2)

(c) is fastest

1. ~~How many of the following functions are part of the defining methods of the~~ **~~Queue<E> ADT~~**~~?~~
   * 1. ~~add(E key)~~
     2. ~~add(E key, int pos)~~
     3. ~~add( )~~
     4. ~~remove(E key)~~
     5. ~~remove(E key, int pos)~~
     6. ~~remove( )~~
   1. 0 b) 1 c) 2 d) 3 e) 4
2. Which of the following is the fastest algorithm runtime for very large input size n?
   1. n2sqrt(n) = n^{2.5}
   2. 4n2(log(n))5  = n^2 times a log factor
   3. 2n3 + 3n2
   4. n4/10 + nlog(n)
   5. n3
3. If we have a set of unit tests for java class X.java which all pass and we see that the unit tests provide complete (full) code coverage for X.java, then we know that X.java is implemented correctly.

a) True b) False

1. Which one the following is a correct statement about ArrayLists and LinkedLists?
   1. In terms of big-O runtimes, it is faster to insert at position 0 into an ArrayList than a LinkedList
   2. In terms of big-O runtimes, it is faster to delete position 0 in an ArrayList than a LinkedList
   3. In terms of big-O runtimes, it is faster to get the value at position 0 of an ArrayList than a LinkedList
   4. In terms of big-O runtimes, it is faster to get the value of the middle of an ArrayList than a LinkedList
   5. In terms of big-O runtimes, it is faster to output the entire contents of an ArrayList than a LinkedList
2. **True or False:** In order to instantiate a java PriorityQueue of Strings, we must implement our own comparison function and pass it in as an argument to the Priority Queue constructor.

True b. False

1. Merge Sort is a O(*n* log(*n*))-time algorithm and Bucket Sort is an O(*n*) time algorithm. **True or False:** This means Bucket sort should always be used over Merge Sort if we already have a good implementation it.

True b. False

1. **True or False:** The largest key in a min heap can always be found at the last index (the rightmost leaf node).

True b. False

1. **True or False:** Mergesort is a stable sort that runs in O(n log(n)) time

True b. False

1. **True or False:** The big-O runtimes of each of the three functions add, remove, contains in a doubly-linked list are no different from the big-O runtimes of the same functions in a singly-linked list

True b. False

1. [2 points each] Consider the ADTs and data structures we have seen so far (for example:

Stack, Queue, SortedList, PriorityQueue, Heap, ArrayList, LinkedList)

Choose one from the above (or any other structure we’ve seen so far) that is appropriate for each of the following applications, and provide 1 or 2 lines of Java code to declare and instantiate it.

* + - * 1. A system needs to log a temperature reading continuously by adding a pair of values (currentTime, currentTemperature), and needs to add these to its collection as quickly as possible so that analysis can be done later. The system is only concerned with adding as fast as possible.
        2. A database for cars maintains a collection of Car objects and their manufactured year, and potentially thousands of customers will be requesting to see all available cars that were made between 2010 and 2015 (or any two years). This system needs to produce the cars within this range as quickly as possible.
        3. A Pokemon collector wants to maintain a collection of caught Pokemon names. She wants to be able to store a name as quickly as possible and also determine whether a name is in the collection already as quickly as possible (so she does not waste any pokeballs trying to catch one she already has). She will not be deleting any Pokemon from her collection.
        4. Same as in part (c) but now this Pokemon collector wants to catch three of every type.
        5. The President’s office has an email address, and the President is too busy to process emails so there is a team of employees who reads the emails and responds to them if necessary. The most **recent** item is the most important in this company, and so we need a system that gives an employee the newest unread email. Any number of employees will ask this system for the next email that needs processing, so this system needs to give the employee the email as quickly as possible.

1. [3 points] Modify the Node class below to use a generic type for its value instead of String. You may edit the code by crossing out text and adding text, as necessary.

**public** **class**  Node {

Node next;

String value;

**public** Node(String s){

next = **null**;

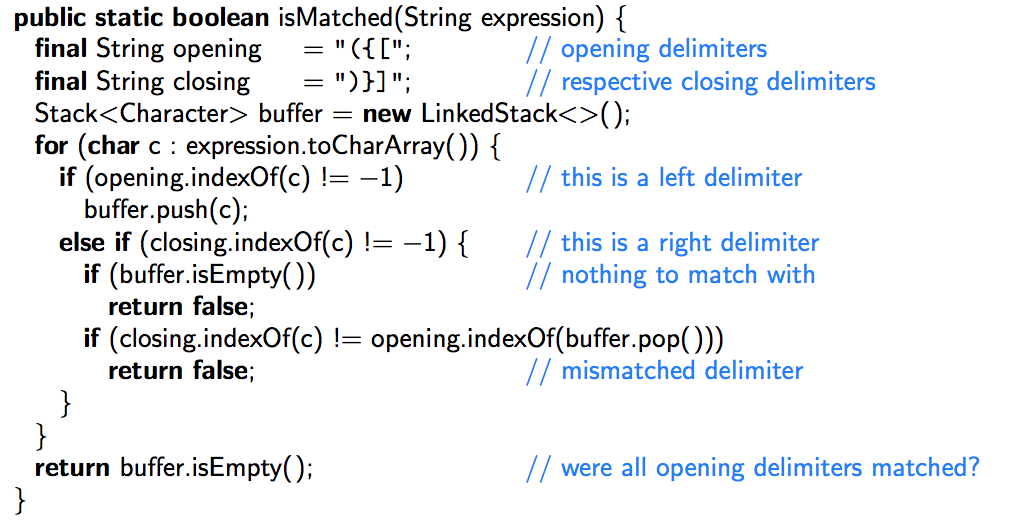
value = s;

}

}

1. [3 points] Fill in the following table with the big-O worst-case runtime (or amortized runtime, when applicable) for each of the operations given and each of the data structures given.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ADT** | **Data Structure** | **.add(n)** | **.remove(n)** | **.contains(n)** |
| SortedList | Array |  |  |  |
| SortedList | Linked List |  |  |  |
| Queue | LinkedList |  |  |  |

1. [2 marks] Consider the code on the right. With the expression

[5(x + y(2-z)])

as input, write the contents of the stack when c=z and also when the method terminates.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| When  c = z |  | At the end |

1. [2 marks] Rank the following runtimes with the slowest runtime first.

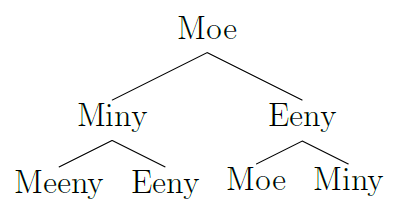
*n*2+2*n*, log(*n*), (1.5)*n*, *n* log(*n*), *n* sqrt(*n*), *n*, *n*3

1. [2 marks] In class, it was shown that a bottom-up heap construction was a faster way to create a heap. What method was it faster than? (That is, describe how to build a heap that is not the bottom-up heap construction). Use 3-4 sentences to answer this question.
2. [2 marks] Draw the 11-entry hash table that results from using the hash function

*h*(*k*) = (2*k*+6) mod 11

to hash the nine keys 12, 1, 13, 23, 25, 2, 3, 14, 24, assuming collisions are handled by linear probing

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  |  |  |  |  |  |  |

1.  [2 marks] Consider the tree to the right.
   1. Write the pre-order traversal of this tree
   2. Write the in-order traversal of this tree
2. [2 marks] Recall that an array-based implementation of a Heap represents the root at index 0 and for any node *i*, we can get its left child with 2*i*+1 and right child with 2*i*+2 and parent with (*i*-1)/2. Consider the following sequence of heap function calls:

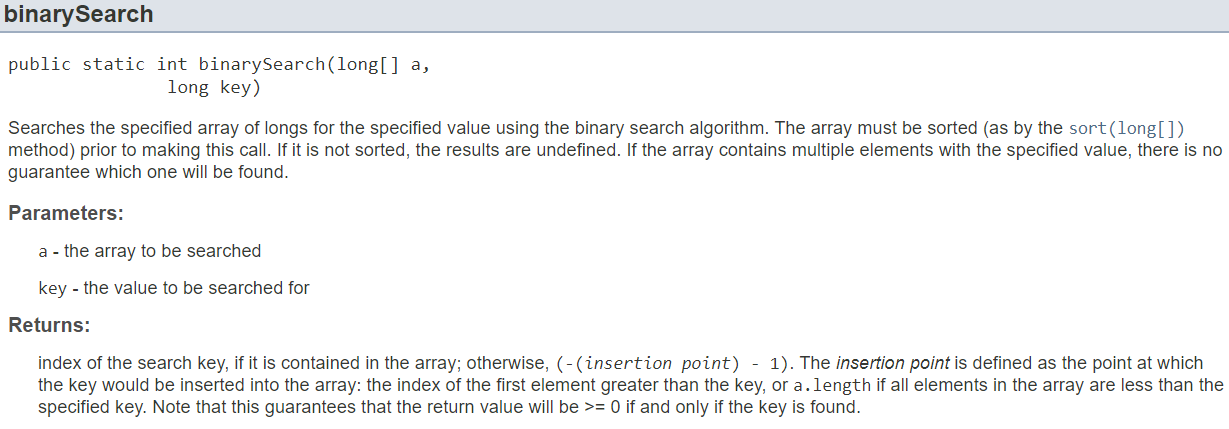
insert(6), insert(3), insert(8), removeMin(), insert(2), insert(7), removeMin(), removeMin()

**Show the array contents after each of the removeMin() calls and show the removed value in the rightmost column**. Reminder: after removing the minimum element, the heap must restructure itself to maintain the heap property

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | Removed Value |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. [2 marks] A linked list of nodes has a head, and every node points to either another node or points to null to mark the end of the linked list. But if the “last” node’s pointer does no point to null and instead points to one of the prior nodes, this linked list has no end and contains a cycle.

Given only the head of a linked list, and not knowing its size, **describe** (in words or in pseudo code) **how we can determine if this linked list contains a cycle or eventually terminates**, without using any more space than just a few pointers to the head of the list. (That is, you cannot store an extra value on each node to mark that you have visited it, you cannot store each node or a hash of each node to keep track of what you have seen, etc).

Here is the java documentation for Arrays.binarySearch( )  
  


1. The following java class is a SortedList class which keeps a sorted array of long ints in an array of size 1000. The creator of this java class knows that it will never be used to store more than 1000 items, so the instance array inside will never need to be re-sized.

Implement the following method that inserts a long integer into the array called values. Have the method return the index at which the number was inserted. You may use Arrays.BinarySearch to help you if you want, but you do not have to use it. Your code must run in O(n) time and should not allocate any additional arrays (but one or two additional variables is fine)

public class SortedList {

int[] values = new int[1000];

int size = 0;

public int add (long a) {

}

}