Using Graph Algorithms

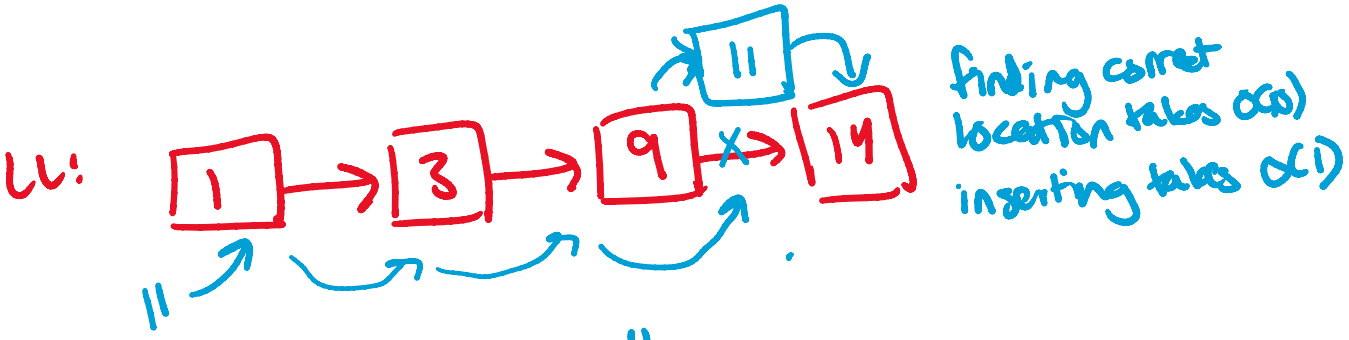
Thursday, December 15, 2022 11:49 AM

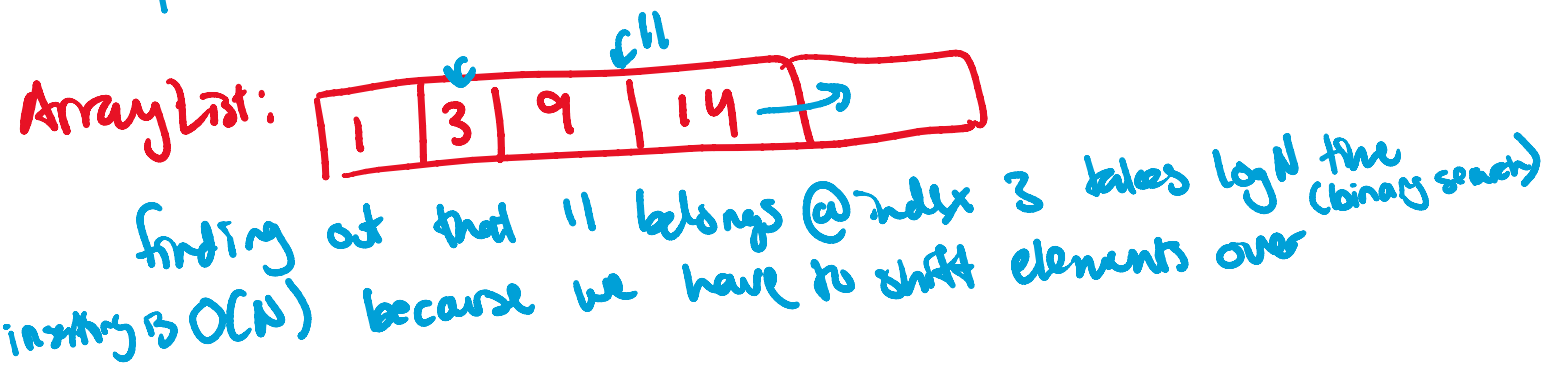
	BFS	DFS	Dijkstra	MST (Prim/Kruskal)
Short description	Traverse a graph to find a path between a start vertex and an end vertex (searches in "levels")	Traverse a graph to find a path between a start vertex and an end vertex (searches by going down a path)	Find the shortest cost path between a start vertex and all vertices in the graph	Given a graph, find a minimum spanning tree on it (a connected tree that has all of the vertices of the graph, with the lowest possible sum of edge weights)
Works on unweighted graph?	yes	yes	no	no
Works on weighted graph?	yes (but weight doesn't matter)	yes (but weight doesn't matter)	yes	yes
Works on directed graph?	yes	yes	yes	no (for the way we discussed in class)
Works on undirected graph?	yes	yes	yes	yes
Data structure(s) needed?	a set to keep track of visited vertices a queue (FIFO) to keep track of the next vertices to check backtrack: dict from a vertex to the vertex it came from	a set to keep track of visited vertices a stack (LIFO) to keep track of the next vertices to check backtrack: dict from a vertex to the vertex it came from	a priority queue that compares vertices by the current known lowest cost route (requires comparator that is able to use a mapping from vertices to route costs) backtrack: dict from a vertex to the vertex it came from	some way to store the resulting tree (pick a graph representation) some way of keeping track of the next edge we want to consider (e.g. sorted list of edges for Kruskal) disjoint sets (to help us quickly compute whether an edge will introduce a cycle or not)
Features/applications	find the shortest-length path between two vertices	will give you back a path between two vertices (not necessarily shortest) typically uses less memory than BFS finding cycles we also saw it used in garbage collection	finding shortest-cost paths	finding lowest-cost spanning trees (Spring 2022 exam question on difference between MST algos and shortest-cost path)

Thursday, December 15, 2022 2:22 PM

Difference between PQ and sorted list:
PQ has 3 main operations: getMin (O(1) time), removeMin (logN), and insert (logN)
To translate to a sorted list, N * logN (remove each elt)
Need (N-1) removes to get N-th smallest elt

Sorted list, insertion takes O(N) (in ArrayList or LinkedList, but for different reasons):
get(i) will get N-th smallest element (O(1) for ArrayList, O(N) for LinkedList)





Heaps -- 2019 Q1

Thursday, December 15, 2022 12:34 PM

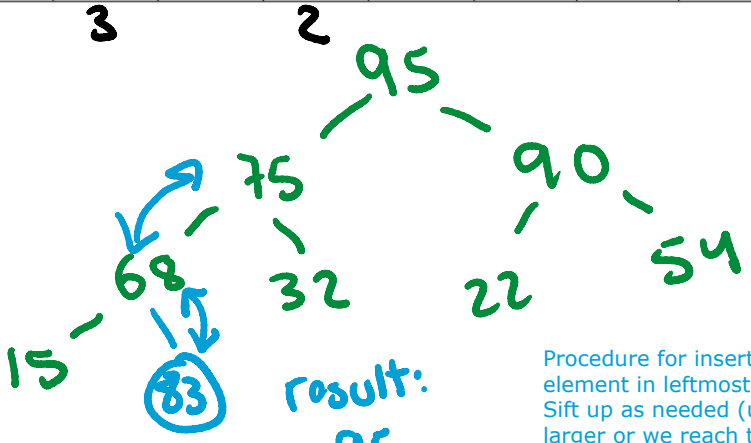
2. (8 points) We want to insert object (Q, 83) into the specific heap shown above. On the copy of the array below, number the array positions that this new object occupies in order during insertion according to the sift-up operation covered in class and homework. Put your numberings under the array. As an example, if an element was in index 2 then 3 then 0 (in an array of letters), your answer should look like:

a	b	c	d	e
3		1	2	

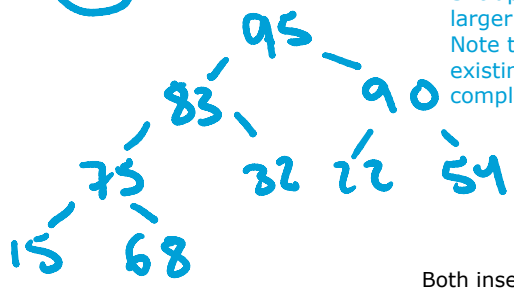
For array-based implementation of heaps, we want to make sure they are as complete as possible (all levels completely full except bottom level, which is filled out left-to-right with no holes)
- Can easily find next open spot to insert in (leftmost leaf) because it will be the first available spot in the array
- Guaranteed to have a balanced heap, so sift up/down is logN

Your answer:

(E, 95)	(B, 75)	(R, 90)	(C, 68)	(A, 32)	(L, 22)	(P, 54)	(S, 15)		
	3		2						1



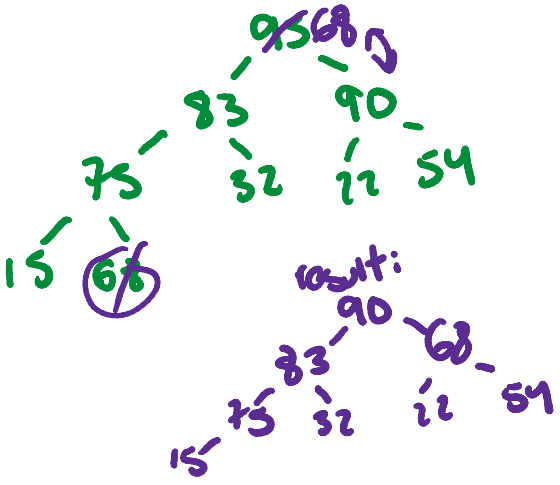
result:

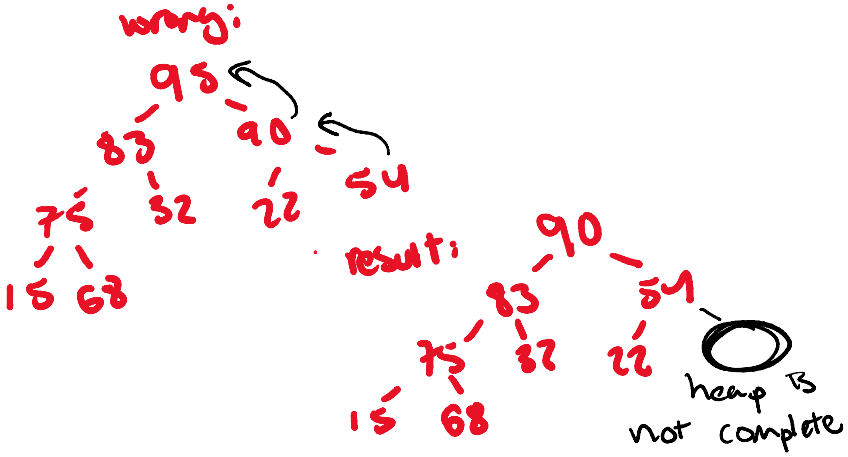


Procedure for insertion: put new element in leftmost leaf
Sift up as needed (until parent is larger or we reach the top)
Note that because sift up swaps two existing elements, this maintains the completeness of the heap

Both insert and removeMax are logN for a balanced/complete heap because the number of swaps in sift up/down is at most the height of the heap

Bonus: procedure for removeMax:
temporarily move the leftmost leaf into root (because we will create a hole that might violate completeness -- see red example
Sift down as needed (until both children are smaller)





(6 points) The game developer decides to only store the top 20 scores in the heap. Once the heap has 20 elements, each new score is compared to the current lowest score in the heap. If the new score is higher than the lowest score in the heap, the lowest score is removed and the new score is added. Otherwise the heap is unchanged.

The developer suggests doing this by comparing the new score to the one in the last-occupied array cell, putting the new score in the last cell to kick out the previous low score if needed, then sifting the new score into place.

Does the proposed approach maintain the top 20 scores for the game as a valid max heap? Explain why or why not.

This question is checking if you can recognize that the last element in the array may not necessarily be the smallest element in the whole heap (look at the example above after we insert 83: 68 becomes the last element in the array, even though 15 is the smallest element). This is because the heap has a partial ordering that helps us optimize insert and removeMax, but is not fully sorted.

DP -- 2019 Q3

Thursday, December 15, 2022 12:29 PM

You want to group equally-sized items into boxes of different sizes while using as few boxes as possible. Imagine that you have 10 items and a supply of boxes, each of which can hold 1, 5, or 7 items. You could put the 10 items in ten boxes of size 1, two boxes of size 5, or one box of size 7 and three boxes of size 1. The two-box solution uses the fewest boxes.

Here's a recursive program that computes the answer, assuming an unlimited supply of boxes of each size and inclusion of boxes of size 1:

```
/*
  * Determines the minimum number of boxes needed to hold a number
  * of items with no wasted space in the boxes. This version assumes
  * that there is always a box of size 1 (which simplifies the code).
  * This also assumes an unlimited supply of boxes of each size.
  *
  * @param numItems: how many items to pack into boxes
  * @param boxSizes: the sizes of boxes available
  *
  * @return: the minimum number of boxes required
  */
def boxCount(numItems: Int, boxSizes: List[Int]): Int = {
   if (numItems <= 0)
      0
   else
      1 + (boxSizes.map(bSize => boxCount(numItems - bSize, boxSizes))).min
}
```

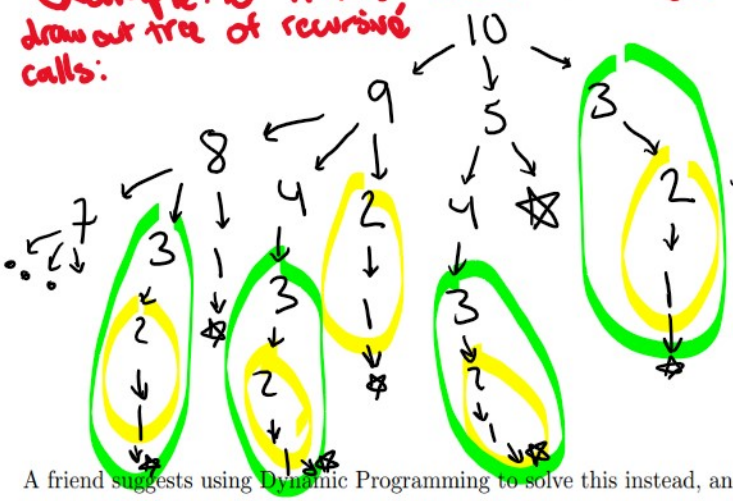
To understand the recursive solution: say you have 10 items. You have to make a choice for the first box you will use.

- If you use a box of size 1, you have put away 1 item, so now you need to solve the problem for 9 items.
- If you use a box of size 5, you have put away 5 items, so now you need to solve the problem for 5 items
- If you use a box of size 7, you have put away 7 items, so now you need to solve the problem for 3 items

So the minimum # of boxes you need is 1 (for the box you just used) + the minimum answer for the problem of 9, 5, or 3 items

1. (7 points) Assuming there is always a box size of 1, what's the worst-case number of calls that will be made to `boxCount` in terms of the number n of items that need to be packed and the number s of different box sizes?

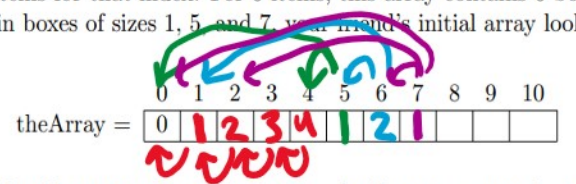
example: 10 items, 3 box sizes (1, 5, 7)
draw out tree of recursive calls:



10 levels (# of items)
for each item (# of boxes)
up to 3 branches
$\sim 3^{10}$ computations
$O(s^n)$

Note the repeated computations in the expansion of the recursive calls: this is a good example of when we might want to use dynamic programming (to store the results of repeated computations that are being used for larger computations!)

A friend suggests using Dynamic Programming to solve this instead, and sets up an array with one cell for each value up to the number of items. Each cell contains the number of boxes needed to pack the number of items for that index. For 0 items, this array contains 0 boxes. For our earlier example of 10 items in boxes of sizes 1, 5, and 7, your friend's initial array looks like:



2. (5 points) How should you compute the contents of cell `theArray[8]` based on the contents of other (specific) cells in `theArray`, given boxes of sizes 1, 5, and 7? Write the formula/expression you would use for index 8 in particular. Use code-like notation, but syntax doesn't matter if your meaning is clear.

1 + min(theArray[7], theArray[3], theArray[1])

8-1 8-5 8-7