School of Computer Science

First Year Undergraduate

06-30175

30175 LC Data Structure & Algorithms

Resit Examinations 2022

[Answer all questions]

Question 1

(a) Write the missing pseudocode in the code below of an efficient recursive method to print, in order, all values in a Binary Search Tree of integers that fall within the inclusive range [min, max]

Assume the method `print(int val)` will print a value and that if `t` is a `BSTNode`, then `t.left`, `t.val` and `t.right` allow access to the left, val and right fields of the node.

```
void printRange(BSTNode t, int min, int max)
{
  // WRITE THE CODE THAT SHOULD BE HERE
  return
}
```

[10 marks]

Ollie

```
void printRange(BSTNode t, int min, int max){
if (t==null) return;
if (t.val >= min)
    printRange(t.left, min, max);
if (min <= t.val <= max)
    print(t.val);
if (t.val <= max)
    printRange(t.right, min, max);
}
```

Shay

```
if t.left != NONE {
    printRange(t.left, min, max)
}

if t.val >= min && t.val <= max {
    print(t.val)
}

if t.right != NONE {
    printRange(t.right, min, max)
}
```

Phoebe

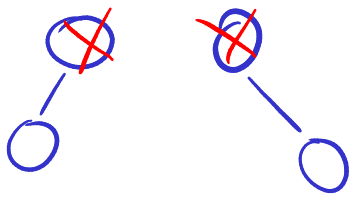
```
a) if (not isEmpty(t)):
    if (not isEmpty(left(t)):
        printRange(left(t))

    if (min <= t.val <= max):
        print(t.val)

    else if (not isEmpty(right(t)):
        printRange(t.right)
```

(b) The standard algorithm for deleting a node from a binary search tree can be described as follows:

```
1:  find the node containing the value to be deleted
2:  if it is a leaf {
3:      remove it
4:  }
5:  else if only one of its children is not empty {
6:      replace the node with the root of the non-empty subtree
7:  }
8:  else {
9:      find the left-most node in the right sub-tree
10:     replace the value to be deleted with that of the left-most node
11:     replace the left-most node with its right child
12: }
```



Explain, in detail, why the complexity of this operation is $O(\log n)$ if the tree is balanced. **[10 marks]**

$O(\log n) + O(1) + O(1) + O(\log n) = O(\log n)$

Shay

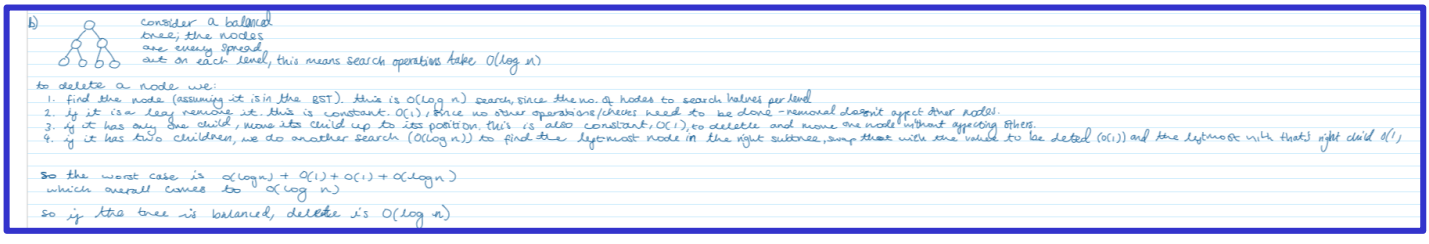
1) b) The complexity of this deletion algorithm would be O(log n) if the tree is balanced, since we have to visit at most h nodes, where h is the height of the tree.

This is because we visit the root node, and then until we find our target node, we keep visiting a node at one level down. Once we find our target node it is either:

- a leaf node (so we've visited h nodes in total)
- it's got one child subtree (so we visit that subtree down to a leaf node, which is h nodes in total)
- or it has two child subtrees, in which case we only explore one of them (the right subtree), visiting h nodes in total.

The height h of a binary search tree is logarithmic to the number of nodes n if the tree is balanced (log n), since every time we move one level down the tree, we're halving our search space.

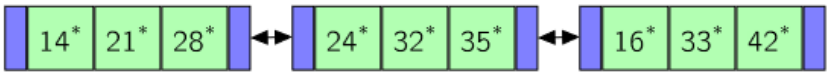
Phoebe



Question 2

(a) Assume an external record file, where each record can contain up to 3 records, is stored on secondary storage as follows (with record block pointers as indicated and records shown as n^* for a record with key value n):

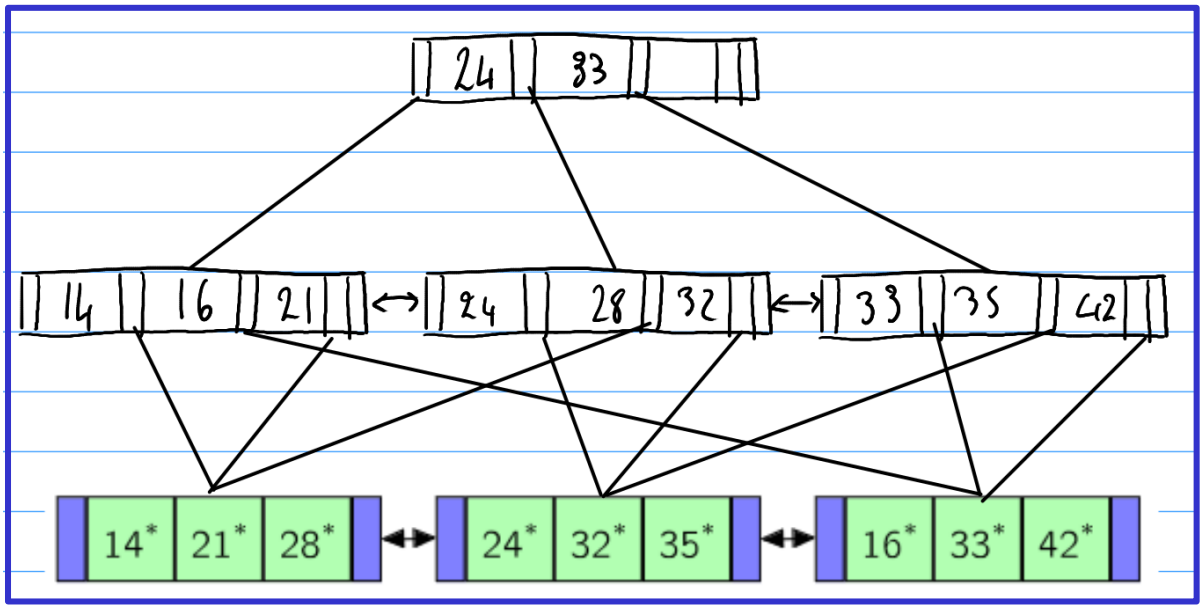
Record file:



Construct a Secondary Index B+Tree on this record file and present your results as a **single** diagram, showing all secondary storage pointers, blocks and block contents for all of the B+Tree and record file blocks. Assume that the B+Tree blocks can hold a maximum of 3 key values.

[10 marks]

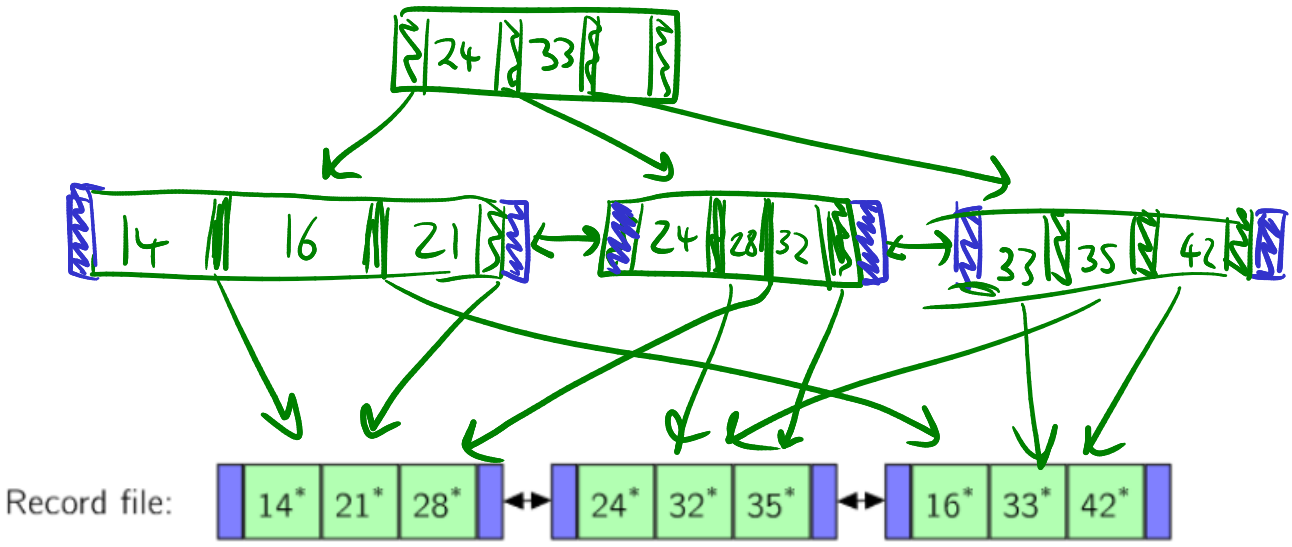
Ollie



Order keys:

14, 16, 21, 24, 28, 32, 33, 35, 42

bulk load:



(b) Assume you have a list of records, each with two fields (X, Y), where X is an integer field and Y is a character field. Explain, showing intermediate steps, how Binsort sorts this list so that it results in all the records sorted by the X field first and then by the Y field for groups of records with the same X value. Use the following list of records in the given order for your demonstration (draw contents of Queues as lists with the head of the list to the left)

$$[(2, c), (1, b), (3, a), (3, c), (1, a), (2, a), (2, b), (3, b), (1, c)]$$

[10 marks]

[(1,a), (1,b), (1,c), (2,a) ... (3,c)]

Sid

Binsort on Y/char field first into 3 bins a,b,c assuming $Y \in \Sigma : \Sigma = \{a,b,c\}$

	Bin a	Bin b	Bin c
Left	(3,a),	(1,b),	(2,c),
	(1,a),	(2,b),	(3,c),
Right	(2,a)	(3,b)	(1,c)

Concatenate all bins in order a,b,c :

Left [(3,a),
(1,a),
(2,a),
(1,b),
(2,b),
(3,b),
(2,c),
Right (3,c),
(1,c)]

Binsort on X (1/2/3)

	Bin 1	Bin 2	Bin 3
Left	(1,a)	(2,a)	(3,a)
	(1,b)	(2,b)	(3,b)
Right	(1,c)	(2,c)	(3,c)

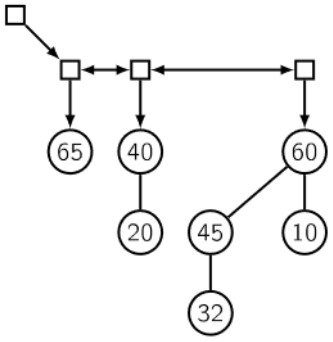
concatenate bins in order 1,2,3 to obtain sorted list:

[(1,a), (1,b), (1,c), (2,a), (2,b), (2,c), (3,a), (3,b), (3,c)]

Question 3

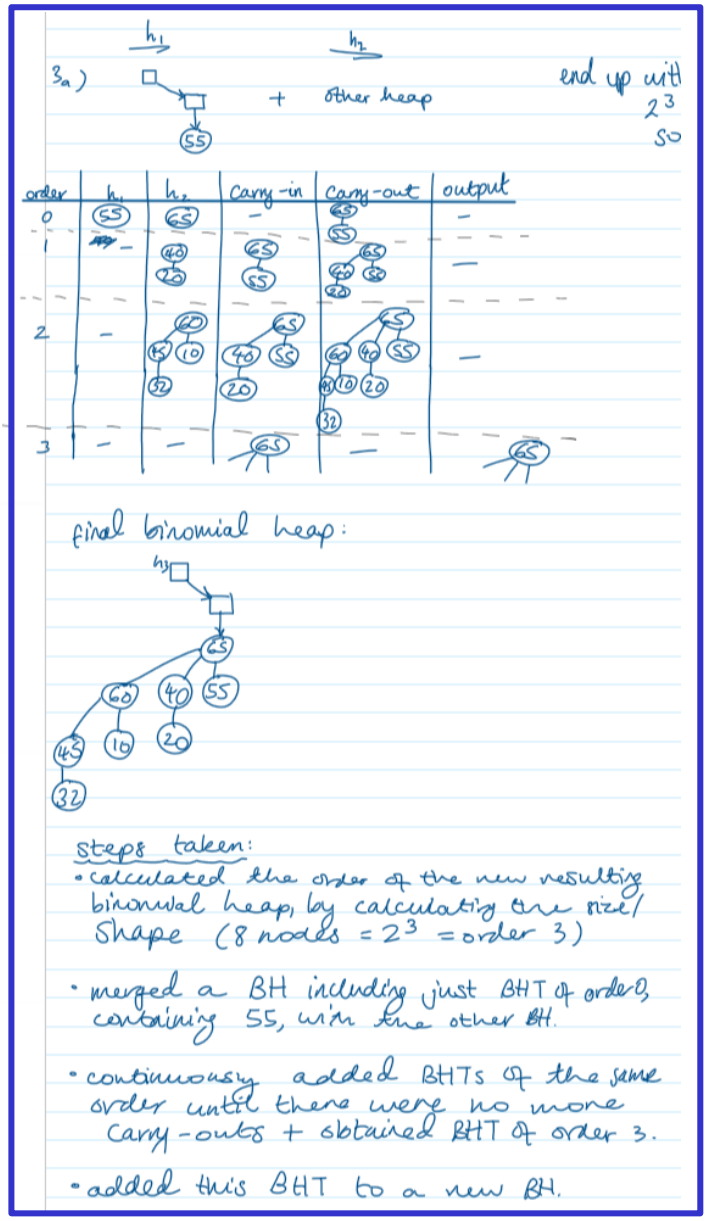
(a) Draw a diagram of the result of inserting the value **55** into the following Binomial Heap, where the highest priority is the highest integer value, and explain the steps by which you came to your result.

max heap

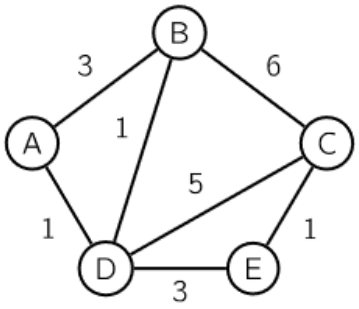


[10 marks]

Phoebe



(b) Consider the following graph:



Demonstrate the execution of the Dijkstra algorithm for finding the minimal path on this graph from node A to node C, by writing out a table of the execution steps in the following format where the first row, following initialisation, is already provided:

A	B	C	D	E	Finished
0,A	∞, B	∞, C	∞, D	∞, E	
⋮	⋮	⋮	⋮	⋮	⋮

Each row should show the results of one iteration of the algorithm, where the *Finished* column identifies the node that is finished in that iteration, and the remaining columns show the current shortest distance of the node of that column from node A, the previous node to the node of this column on the current shortest path from A and a tick mark if the node of the column is finished. Finally, explain how the shortest path, and its length, can be read from the final line of the table on completion of the algorithm. **[10 marks]**

Shay

A	B	C	D	E	Finished
0,A	∞,B	∞,C	∞,D	∞,E	
0,A,✓	3,A	∞,C	1,A	∞,E	A
:	2,D	6,D	1,A,✓	4,D	D
:	2,D,✓	:	:	:	B
:	:	5,E	:	4,D,✓	E
0,A,✓	2,D,✓	5,E,✓	1,A,✓	4,D,✓	C

Our algorithm is complete when we have visited all the nodes (denoted by a tick being in each column in the last row of the table above).

We can obtain the shortest path by looking at the cell in the bottom row corresponding to the column of the destination node, so if we want to get from A to C, we look at the 5,E,tick. This tells us that the shortest path from our initial node A to the node C has a total path cost of 5, and the node we get to C from is E.

We can then look at E's last row, 4,D,tick, which tells us that we get to E from D.

Looking at D, 1,A,tick, we get to D from A. A is our initial node, so our path to get from A to C is from A->D->E->C.

Ollie

A	B	C	D	E	Finished
0,A✓	∞,B	∞,C	∞,D	∞,E	A
0,A	3,B	∞,C	1,A✓	∞,E	D
0,A	2,D✓	6,C	1,A	4,E	B
0,A	2,D	6,C	1,A	4,D✓	E
0,A	2,D	5,E✓	1,A	4,D	C