Candidate Number: 2031B
Paper 1
Question 7

a.

i. Let k be the number of elements in the BST. If the BST is balanced, the cost of a single insertion is O(log k) because one comparison must be made for up to each level of the tree, and there are log k levels. In the worst case, the BST is totally unbalanced and there are k levels and so each insertion is O(k). Therefore the average complexity of phase 1 is O(log 1 + log 2 + log 3 + log 4 + … + log n) which is O(log (n!)) which is approximately O(n log n). In the worst case, phase 1 is O(1 + 2 + 3 + … + n) = $O(n^2)$

Extracting the minimum from the BST is similarly O(log k) in the average case (because the BST is balanced and there are log k layers) and O(k) in the worst case (because the BST is unbalanced and there are k layers), and so phase 2 is O(n log n) in the average case and $O(n^2)$ in the worst case.

ii. Have a pointer to the parent of the most recently removed node from the BST. Then, instead of searching the BST again for the next smallest node, simply check whether the node at this pointer has a child, and if so find the minimum node of the subtree whose root is that child (and update the pointer to that node's parent), and if it does not have a child, then the node at the pointer is the new min, and so the pointer to be updated to its parent.

This reduces the time complexity of phase 2 to O(n). The overall algorithm is still O(n log n) in the average case and $O(n^2)$ in the worst case.

iii.

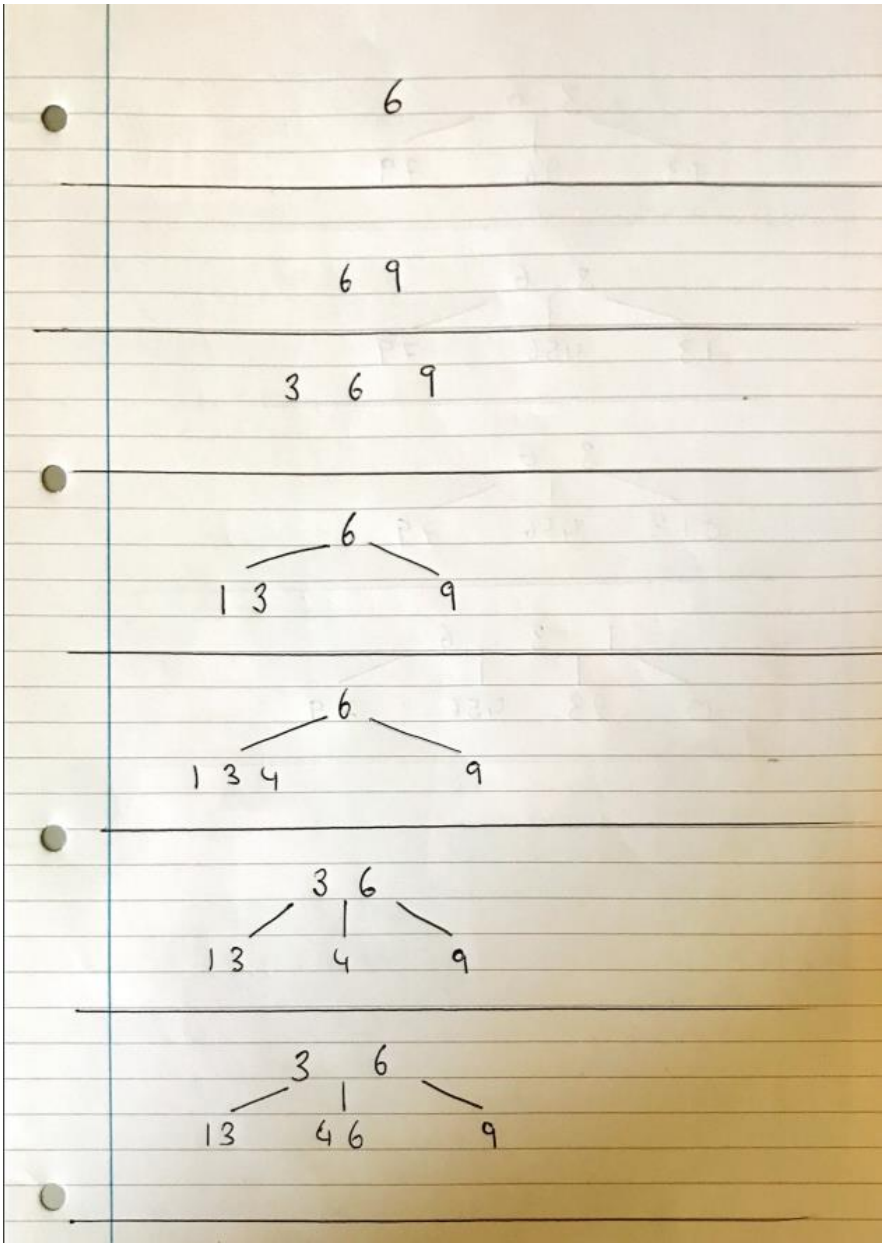
Algorithm	Worst Case Time Complexity	Worst Case Space Complexity
Enhanced BST-sort	$O(n^2)$	O(n)
Heapsort	O(n log n)	O(n)
Mergesort	O(n log n)	O(n)
Quicksort	$O(n^2)$	O(n)

Enhanced BST would never be preferable to any of the above. Even in the worst case of Quicksort, if the pivot choices for the quicksort implementation is likely to result in small partition, (for example, if the pivot is the first element, and the list is already close to sorted) these are the same conditions which would result in the BST being unbalanced, and so it too would perform at its worst.

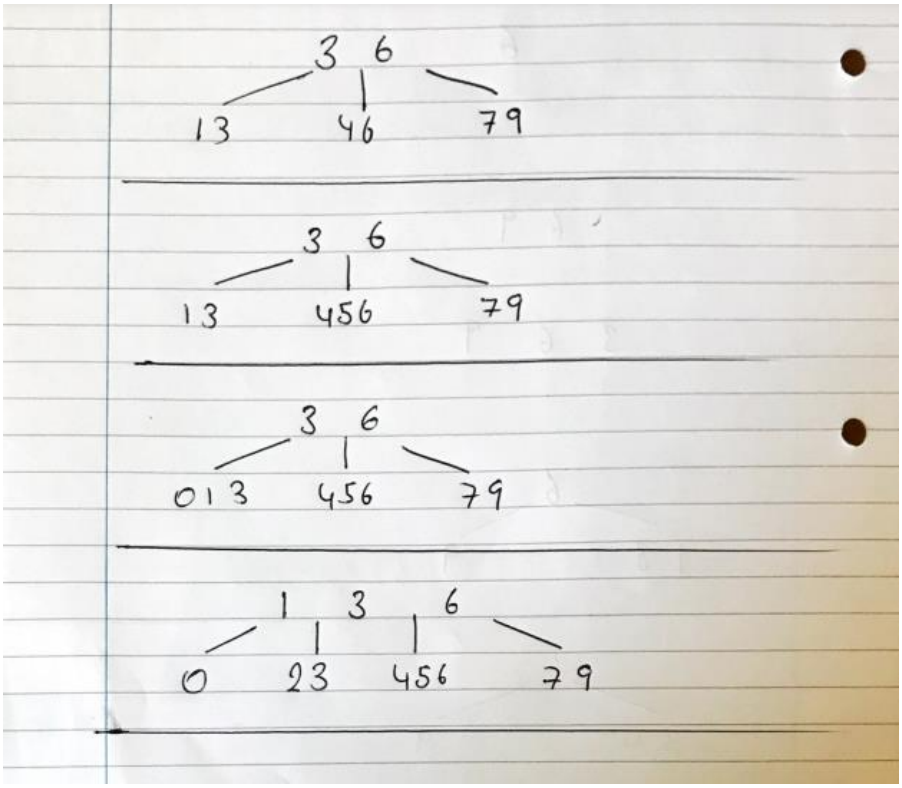
b.

i.

Candidate Number: 2031B
Paper 1
Question 7



Candidate Number: 2031B
Paper 1
Question 7



ii. The array will be nm bits long. The tree will also require nm bits of space to store the values. In the worst case, all of the nodes are 2 nodes and so there are 2(n-n/2)=n pointers, each of which must be log n bits long to uniquely identify a node. Therefore in the worst case the algorithm requires 2nm + n log n bits. In the best case, all of the nodes are 4 nodes and so there are 4(n/3-n/4)=4n/3 - n pointers, each of which must be log (n/3) bits long and so in the best case the algorithm requires 2nm + (4n/3-n)log(n/3) bits.

Upper bound: 2nm + n log n
Lower bound: 2nm + (4n/3-n)log(n/3)

iii. Each insert into the tree is O(log k) where k is the number of items already in the tree and so the total complexity of phase 1 is O(log 1 + log 2 + log 3 + … + log n) = O(log n!) = O(n log n) . Each popmin is O(1) and so the total time complexity of phase 2 is O(n)

iv.

Algorithm	Worst Case Time Complexity	Worst Case Space Complexity
Enhanced 2-3-4-sort	O(n log n)	O(n log n)
Heapsort	O(n log n)	O(n)
Mergesort	O(n log n)	O(n)
Quicksort	$O(n^2)$	O(n)

Candidate Number: 2031B
Paper 1
Question 7

It would be preferable to use Enhanced 2-3-4-sort over quicksort if there are many items to sort, and storage space is less of an issue than running time. It would never be beneficial to use Enhanced 2-3-4-sort over any of the other algorithms.