Assignment 2

Data Structure and Algorithms
COMP 352
Section AA

By
Chloe Hei Yu Law
ID: 40173275
Concordia University
June 2021

QUESTION 1

1. for i = 0 to n = 1 do
2. Var[i] = 0 ⟶ n
3. end for
4.. for i = 0 to n – 2
5. for j = i + 1 to n – 1 do
6. if A[i] ≤ A[j] then ⟶ (n – 1) + (n – 2) + … + 1
7. Var[j] = Var[j] + 1
8. else ⟶ (n – 1) + (n – 2) + … + 1
9. Var[i] = Var[i] + 1
10. end if
11. end for
12. end for
13. for i = 0 to n – 1 do
14. S[Var[i]] = A[i] ⟶ n
15. end for
16. Return S

a) Total number of steps: $n + 2[(n -1) + (n – 2) + \dots + 1] + n = n^2 + n$
$O(n^2)$ and $\Omega(n^2)$

b) A = {60, 35, 81, 98, 14, 47}
Let Z = {0, 0, 0, 0, 0, 0}
Show the values in array Z after each iteration by for loop in line 4
(i = 0) Z = {3, 0, 1, 1, 0, 0}
(i = 1) Z = {3, 1, 2, 2, 0, 1}
(i = 2) Z = {3, 1, 4, 3, 0, 1}
(i = 3) Z = {3, 1, 4, 5, 0, 1}
(i = 4) Z = {3, 1, 4, 5, 0, 2}
Show the values in array Z after the for loop in line 13
(i = 0 to i =5) Z = {3, 1, 4, 5, 0, 2}
Show the values in array S
S = {14, 35, 47, 60, 81, 98}

c) This method sorts array A in ascending order where all the values in the array in array Z are indexes of the corresponding values of the array A in the array S. If the array A has n integers, the array S will also contain n integers where they are sorted in ascending order.

QUESTION 1

d) A heap sort algorithm can improve the runtime and have a time complexity of O(Log n)

```
Algorithm sort(A)
Input: Array A of integers containing elements to be sorted
Output: Sorted array A
n ← A.length
for (i = n / 2 – 1 ; i >= 0; i--)
        heapify(A, n, i)
for (i = n – 1; i > 0; i--)
        swap (A[0], A[i])
        heapify(A, i, 0)
```

```
Algorithm heapify(A, n, i)
Input: Array A of integers, inter n of the size of the heap, node i of the index of the array
Output: The heapify sub-tree
largest ← I
left ← 2 * i + 1
right ← 2 * i + 2
if (left < n AND A[left] > A[largest]) then
        largest ← left
if (right < n AND A[right] > A[largest]) then
        largest ← right
if (largest != i) then
        swap (A[i], A[largest])
        heapify(A, n, largest)
```

e) By using a heap sort algorithm, the space complexity is O(1) since there is only need to array the data with the smallest element at the back and then to swap the last element in the array with the first element in the array to then shuffle the largest element down the heap until there is the smallest remaining element at the last position of the array

QUESTION 2

Category 1

- Implement List ADT can be used because it contains the methods set, add and remove

Category 2

- Implement Positional ADT can be used because there is not requirement to add items strictly at the beginning, end or middle

Category 3

- Implement sequence ADT can be used because they need to be added and removed in a sorted alphabetical order of their destination

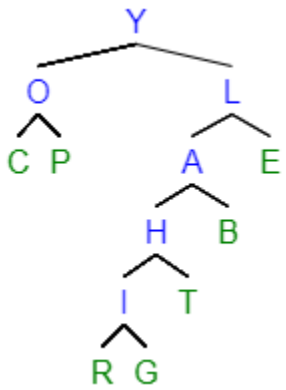
QUESTION 3

a)
Inorder: C O P Y R I G H T A B L E
Postorder: C P O R G I T H B A E L Y

Single binary tree where the root is the last element in post order so “Y” is the root of the tree
Search “Y” Inorder to find the subtrees

I.

II.



b) The contents of the array list are completely filled from the left side because it is a complete binary tree. The contents in the array list are: Y, O, L, C, P, A, E, H, B, I, T, R, G

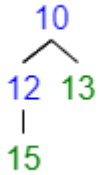
QUESTION 4

a) Draw the min-heap that results from the bottom-up construction algorithm on the following list of values: 10, 12, 13, 15, 4, 16, 8, 9, 2, 20, 7, 14, 6, 23,19
Starting from the bottom layer, use the values from left to right as specified above. Show immediate steps and the final tree representing the min-heap.

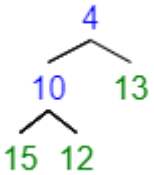
Step 1: Insert keys 10 and 12

Step 2: Insert key 13

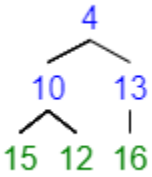
Step 3: Insert key 15



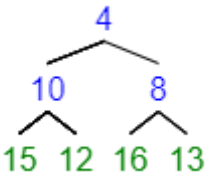
Step 4: Insert key 4



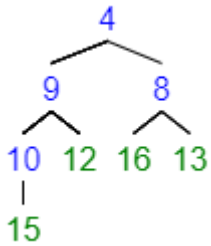
Step 5: Insert key 16



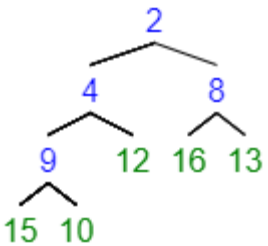
Step 6: Insert key 8



Step 7: Insert key 9

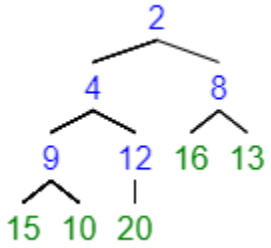


Step 8: Insert key 2

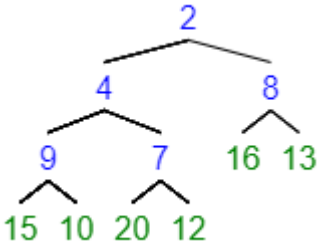


QUESTION 4

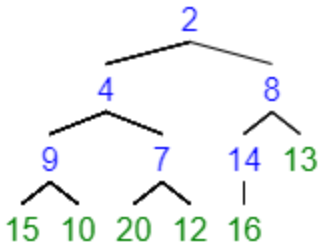
Step 9: Insert key 20



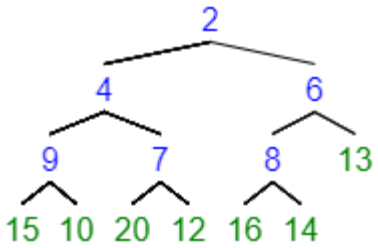
Step 10: Insert key 7



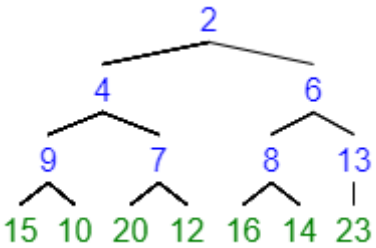
Step 11: Insert key 14



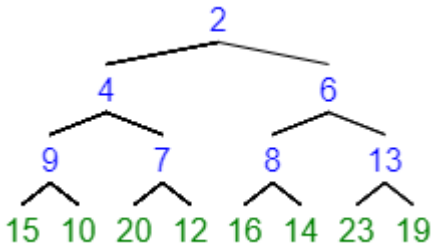
Step 12: Insert key 6



Step 13: Insert key 23

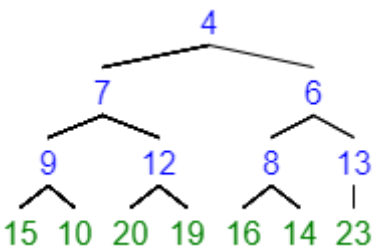


Step 14: Insert key 19

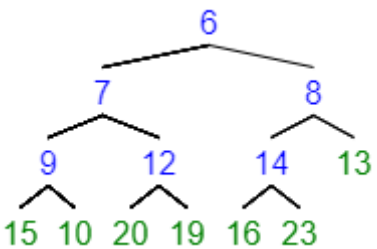


b) Perform the operation removeMin two times on the heap you created in part (a) and show the resulting min-heap after each step and the final tree representing the min-heap.

removeMin(2)



removeMin(4)



QUESTION 5

a) Given a tree T, where n is the number of nodes of T.
Give an algorithm for computing the depths of all the nodes of a tree T. What is the complexity of your algorithm in terms of Big-O?

maxDepth()
1. If tree is empty, then return 0
2. Else
    - Get the max depth of left subtree recursively
        - call maxDepth(tree → left-subtree)
    - Get the max depth of right subtree recursively
        - call maxDepth(tree → right-subtree)
    - Get the max of max depth of left and right subtrees and add 1 to it for the current node
        - max_depth = max(max depth of left subtree, max depth of right subtree) + 1
    - Return max_depth

O(n) is the time complexity

b) We say that a node in a binary search tree is full if it has both a left and a right child.
Write an algorithm called Count-Full-Nodes(t) that takes a binary search tree rooted at node t and returns the number of full nodes in the tree. What is the complexity of your solution?

Algorithm Count-Full-Nodes(t)
1. Create empty Queue Node and push the root node “t” to Queue
2. While nodeQueue is not empty
    - Pop an item from Queue and process it
        - If it is a full node, then increment count++
    - Push left child of popped item to Queue if available
    - Push right child of popped item to Queue if available

O(n) is the time complexity