Question 1

Class 1: Positional ADT is the best data type to use in this case since it works with the position and gives the user the possibility to add and remove items.

Class 2: List ADT is what should be use in this case since it uses indices. It also has the functions required already implemented.

Class 3: Sequence ADT is best in this class as it is great in storing an ordered collection of elements. Packages can be added anytime and sorted.

Question 2

Input: Stack S and Queue Q, and an element E

temp

found <- false

FOR i <- 0 to N do

Q.enqueue() = S.pop()

FOR i <- 0 to N do

S.push() = Q.dequeue()

FOR i <- 0 to N do

temp = S.pop()

if temp is equal to E

found <- true

Q.enqueue() = temp

FOR i <- 0 to N do

S.push() = Q.dequeue()

return found

Question 3

In this case, we should implement a double ended queue as an array list, as it would give the permission to “queue” from the end and from the start and “dequeue” from both as well.

front <- -1

rear <- 0

-add(i,e):

IF i == 0 THEN

Question 4

a)

b)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | Value | 7 | T | 15 | - | 23 | - | 31 | - | 39 | - | 47 | - | 55 | - |
| 0 | F | 8 | P | 16 | - | 24 | - | 32 | - | 40 | - | 48 | - | 56 | - |
| 1 | C | 9 | R | 17 | - | 25 | - | 33 | - | 41 | - | 49 | - | 57 | - |
| 2 | B | 10 | A | 18 | - | 26 | - | 34 | - | 42 | - | 50 | - | 58 | - |
| 3 | K | 11 | - | 19 | - | 27 | - | 35 | - | 43 | V | 51 | - | 59 | - |
| 4 | J | 12 | - | 20 | - | 28 | - | 36 | - | 44 | Q | 52 | - | 60 | - |
| 5 | - | 13 | - | 21 | I | 29 | - | 37 | - | 45 | - | 53 | - | 61 | - |
| 6 | - | 14 | - | 22 | L | 30 | - | 38 | - | 46 | - | 54 | - | 62 | - |

Question 5

a)

log2(15+1) - 1 = 3 = height of the tree

15 17 18 20 9 21 13 14

7 22 12 19 replace

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

🡺15 17 22 20 12 21 19 14

7 18 9 13

4 23 replace

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

🡺15 17 22 20 23 21 19 14

7 18 12 13

4 9

6 replace

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

🡺15 17 22 20 23 21 19 14

7 18 12 13

6 9

4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

🡺 Min-Heap

4

6 9

7 18 12 13

15 17 22 20 23 21 19 14

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b)

removeMin #1

14

6 9

7 18 12 13

15 17 22 20 23 21 19 []

🡺 6

7 9

14 18 12 13

15 17 22 20 23 21 19 []

* 14 goes to the top and gets deleted
* 14 and 6 switch
* 14 and 7 switch

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

removeMin #2

19

7 9

14 18 12 13

15 17 22 20 23 21 [] []

🡺 7

14 9

15 18 12 13

19 17 22 20 23 21 [] []

* 19 goes to the top and gets deleted at the end
* 19 and 7 switch
* 19 and 14 switch
* 19 and 15 switch