Conder Shou
cs3544
Data Structures in Java

Homework 5

7.11

Initial Heap Array:

879, 811, 572, 434, 453, 123, 543, 65, 111, 142, 242, 102

1. deleteMax

   811, 453, 572, 434, 242, 123, 543, 65, 111, 142, 102, 879

2. deleteMax

   572, 453, 543, 434, 242, 123, 102, 65, 111, 142, 811, 879

3. deleteMax

   543, 453, 142, 434, 242, 123, 102, 65, 111, 572, 811, 879

4. deleteMax

   453, 434, 142, 111, 242, 123, 102, 65, 543, 572, 811, 879

5. deleteMax

   434, 242, 142, 111, 65, 123, 102, 453, 543, 572, 811, 879

6. deleteMax

   242, 111, 142, 102, 65, 123, 434, 453, 543, 572, 811, 879

7. deleteMax

   142, 111, 123, 102, 65, 242, 434, 453, 543, 572, 811, 879

8. deleteMax

   123, 111, 65, 102, 142, 242, 434, 453, 543, 572, 811, 879

9. deleteMax

   111, 102, 65, 123, 142, 242, 434, 453, 543, 572, 811, 879

10. deleteMax

    102, 65, 111, 123, 142, 242, 434, 453, 543, 572, 811, 879

11. deleteMax

    65, 102, 111, 123, 142, 242, 434, 453, 543, 572, 811, 879

7.19

Initial List

3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5

Find Pivot:

(3, 9, 5) → 5

Swap Pivot with End of Array (Pivot remains in same place)

3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5

Counters **Stop**

3, 1, 4, 1, **5**, 9, 2, 6, 5, **3**, 5

Swap (5, 3)

3, 1, 4, 1, 3, 9, 2, 6, 5, 5, 5

Counters **Stop**

3, 1, 4, 1, 3, **9**, **2**, 6, 5, 5, 5

Swap (9, 2)

3, 1, 4, 1, 3, 2, 9, 6, 5, 5, 5

Counters Continue and Stop (No Swap because Counters are **Crossed**)

3, 1, 4, 1, 3, **2** (j), **9** (i), 6, 5, 5, 5

Swap Pivot with the position of counter "i" (9, 5)

3, 1, 4, 1, 3, 2, 5, 6, 5, 5, 9

First Set: (3, 1, 4, 1, 3, 2) *Pivot:*(5) *Second Set:* (6, 5, 5, 9)

Sorting first set of elements now:

3, 1, 4, 1, 3, 2

Find Pivot:

(3, 4, 2) → 3

Swap Pivot with End of Array (Swap 3, 2)

2, 1, 4, 1, 3, 3

Counters **Stop**

2, 1, **4**, 1, **3**, 3

Swap (4, 3)

2, 1, 3, 1, 4, 3

Sorting second set of elements:

6, 5, 5, 9

Find Pivot:

(6, 5, 9) → 6

Swap Pivot with End (6, 9)

9, 5, 5, 6

Counters **Stop**

9, 5, **5**, 6

Swap (9, 5)

5, 5, 9, 6

Counters Continue and Stop (Counters **Crossed)**

2, 1, 3, **1** (j), **4 (**i), 3

Swap Pivot with Position of Counter "i" (4, 3)

2, 1, 3, 1, 3, 4

First Set: (2, 1, 3, 1) *Pivot:* (3) *Second Set:* (4)

Find Pivot:

(2, 1, 3, 1) → 1

Swap Pivot with End of Array (Nothing Happens – Already there)

2, 1, 3, 1

Counters **Stop**

2, **1**, 3, 1

Swap (2, 1)

1, 2, 3, 1

Counters **Stop** (**Crossed**)

1, **2**, 3, 1

Swap Pivot with "i" (2, 1)

1, 1, 3, 2

First Set: (1) *Pivot:* (1) *Second Set:* (3, 2)

Counters **Crossed**

5, **5** (j), **9** (i), 6

Swap Pivot with "i" (9, 6)

5, 5, 6, 9

First Set: (5, 5) *Pivot:* (6) *Second Set:* (9)

Sorted.

Because the second subset is unsorted, and its size is below the cutoff, we will now use insertion sort

Sorted.

Second Set: (3, 2)

Swap (3, 2)

2, 3

Sorted.

Combining:

1, 1, 2, 3 3 (a pivot), 4, 5 (first pivot) 5, 5, 6, 9

Final Array:

1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9

17.24

[A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T]

We want pivot to be the largest possible (or as close to the end as possible).

We will use the optimized quicksort routine this time and sort the left, center, and right elements of the array

---

Choosing Pivot: 19

A = 20; J = 19; T=18

[20, B, C, D, E, F, G, H, I, 19, K, L, M, N, O, P, Q, R, S, 18]

Sort Pivot and Partition:

[18, B, C, D, E, F, G, H, I, S, K, L, M, N, O, P, Q, R, 19, 20]

Pivot is swapped with position "i" – if worst case, then "i" is at the end:

[18, B, C, D, E, F, G, H, I, S, K, L, M, N, O, P, Q, R, 19, 20]

First set would be everything besides the previous pivot (19) and the last element (20)

Choosing Pivot: 17

I = 17; R = 16

[18, B, C, D, E, F, G, H, 17, S, K, L, M, N, O, P, Q, 16]

Sort and Swap Pivot and Partition:

[16, B, C, D, E, F, G, H, Q, S, K, L, M, N, O, P, 17, 18]

Choosing Pivot: 15

H = 15; *P = 14*

[16, B, C, D, E, F, G, 15, Q, S, K, L, M, N, O, 14]

Sort and Swap Pivot and Partition:

[14, B, C, D, E, F, G, O, Q, S, K, L, M, N, 15, 16]

Choosing Pivot: 13

N = 12; G = 13

[14, B, C, D, E, F, 13, O, Q, S, K, L, M, 12]

Sort and Swap Pivot and Partition:

[12, B, C, D, E, F, M, O, Q, S, K, L, 13, 14]

Choosing Pivot: 11

F = 11; L = 10

[12, B, C, D, E, 11, M, O, Q, S, K, 10]

Sort/Swap Pivot and Partition:

[10, B, C, D, E, K, M, O, Q, S, 11, 12]

Pivot: 9

E = 9; S = 8

[10, B, C, D, 9, K, M, O, Q, 8]

Partition:

[8, B, C, D, Q, K, M, O, 9, 10]

Pivot: 7

D = 7; O = 6

[8, B, C, 7, Q, K, M, 6]

Partition:

[6, B, C, M, Q, K, 7, 8]

Pivot: 5

C = 5; K = 4

[6, B, 4, M, Q, 3]

Partition:

[4, B, Q, M, 3, 6]

Pivot: 3

B = 3; M = 2

[4, 3, Q, 2]

Q must be 1.

Recalling all the numbers we assigned letters to, and substituting them in with the original, alphabetized array, we get the initial array of:

[20, 3, 5, 7, 9, 11, 13, 15, 17, 19, 4, 10, 2, 12, 6, 14, 1, 16, 8, 18]

9.1

s, G, D, H, A, B, E, I, F, C, t

9.7a

Visual demonstration attached to end of pdf.

9.38

a.

We figure out whether stick *a* is above, below, or unrelated to *b* by projecting the sticks on to the XY-plane, testing for an intersection, and then if applicable, testing the Z coordinates of that intersection. In other words:

1. Projection on to XY-Plane:

   If stick *a* is defined by the coordinates: (x1, y1, z1) and (x2, y2, z2), and stick *b* is defined by: (x3, y3, z3) and (x4, y4, z4). We ignore the third z-coordinates, and reduce the ordered triples to (x1, y1), (x2, y2), (x3, y3), and (x4, y4).

2. Testing for intersection:

   Look for a 2D intersection between the two lines constructed from their corresponding endpoints.

3. If no intersection in the XY-plane was found, then there is no relation between stick *a* and stick *b*.

4. If an intersection in the XY-plane **was** found, then we test the Z coordinates of that intersection:

   If the Z coordinate of stick *a* at that intersection point is greater than the Z coordinate of stick *b*, then stick *a* is <u>above</u> stick *b*.

   If the Z coordinate of stick *a* at that intersection point is less than the Z coordinate of stick *b*, then stick *a* is <u>below</u> stick *b*.

b.

The topological sort algorithm determines whether it is possible to pick up all the sticks, and if so, provides the sequence of stick pickups that accomplishes this.

This is because, since one stick cannot be picked up without first removing all the stick(s) above it, this is problem can actually be structured as a directed acyclic graph, as shown below:

stick *a* → stick *b* → stick *c* → stick *d* → …. → (stick at the bottom of the pile)

As a result, since we are looking for an ordering of which sticks to pick up in this DAG, we can therefore use a topological sort.

More specifically:

1. We can set the starting stick to indegree 0, and the rest of the sticks to the indegree 1.
2. Then, we can use a for loop initialized at 0 that runs as many as times as there are sticks, and removes whichever stick that has indegree 0.
3. Upon removal, it will set the order number of the removed stick to the current counter value, and decrement the indegree of the adjacent stick (the one below)

We could also use a Queue. However, since the graph is relatively simple, such a data structure might not be needed to implement this kind of sorting algorithm.