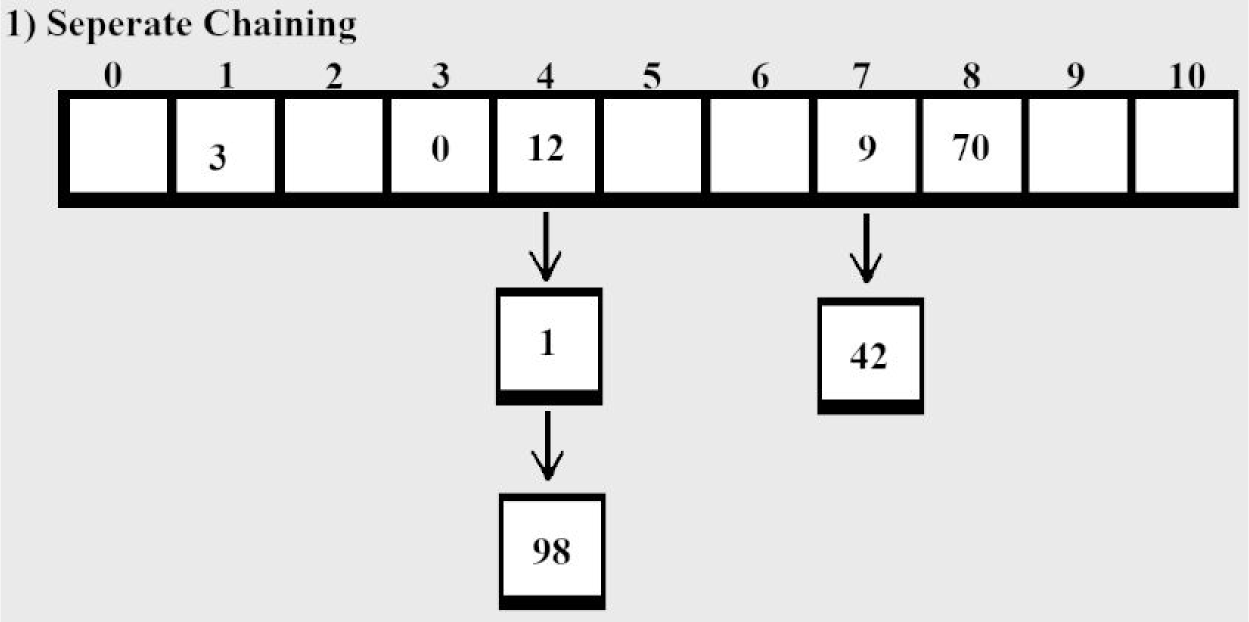
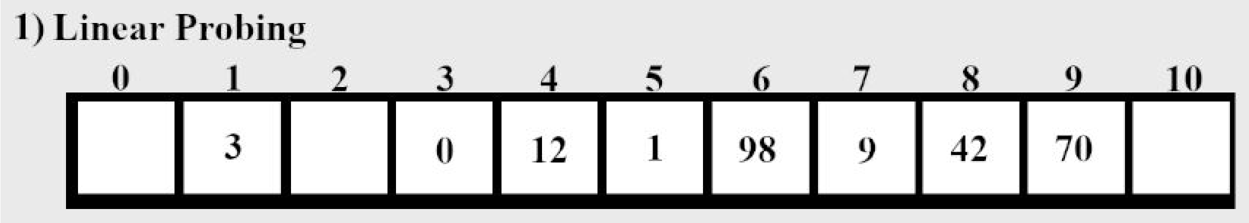
Gerald Hoff
Advanced Data Structures Java
Homework 3 - Heaps, Hashing, Sorting
11/19/2020

Git: https://github.com/geraldHoff/Java-HW3



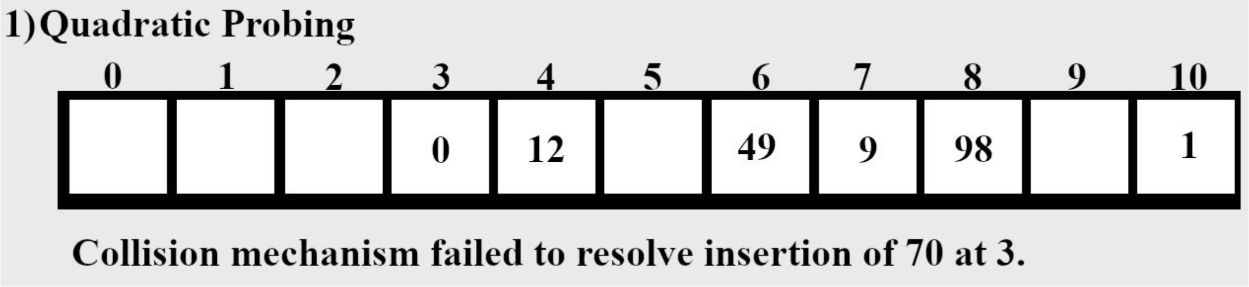
1) Seperate Chaining

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
|  | 3 |  | 0 | 12 |  |  | 9 | 70 |  |  |



1) Linear Probing

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
|  | 3 |  | 0 | 12 | 1 | 98 | 9 | 42 | 70 |  |



1)Quadratic Probing

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  | 0 | 12 |  | 49 | 9 | 98 |  | 1 |

Collision mechanism failed to resolve insertion of 70 at 3.

2.

15 would be the best initial table size. 15 is not so large that it takes up too much memory, and it is not so small that it will need to be immediately rehashed.

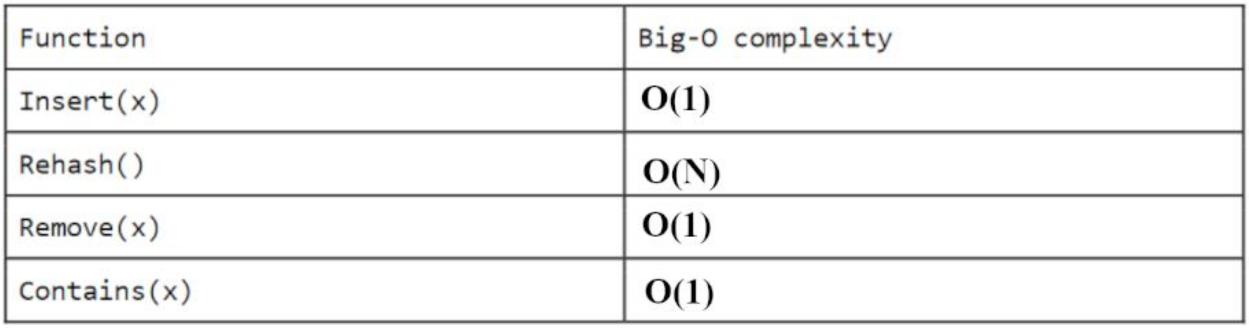
3.

a) Load factor = 53,491 entries / 106,963 buckets = 0.5

b) There is no need to rehash, as the load factor is less than 0.75.

c) There is still no need to rehash, the load factor is less than 0.75.

4.

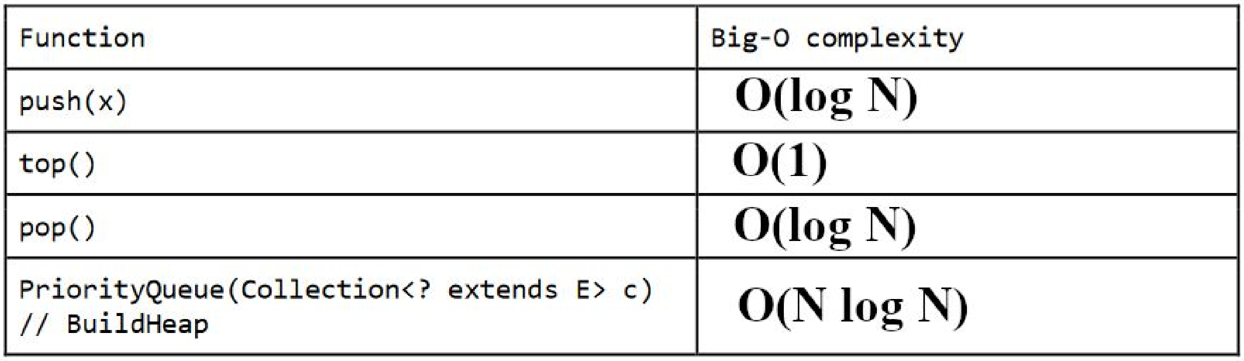


| Function | Big-O complexity |
|---|---|
| Insert(x) | O(1) |
| Rehash() | O(N) |
| Remove(x) | O(1) |
| Contains(x) | O(1) |

7.

No clue yet : https://www.geeksforgeeks.org/load-factor-and-rehashing/

8.



| Function | Big-O complexity |
|---|---|
| push(x) | O(log N) |
| top() | O(1) |
| pop() | O(log N) |
| PriorityQueue(Collection<? extends E> c)<br>// BuildHeap | O(N log N) |

9.

A good application for a priority queue would be anything that needs to store a bunch of values, and easily find the highest or lowest of those values. A hypothetical situation could be where you have a chain. Each chain link has a certain amount of force it can withstand before breaking. A min heap could be used to store these values. The weakest link in the chain would be the first to break, so having the smallest value be readily accessible would be very handy.

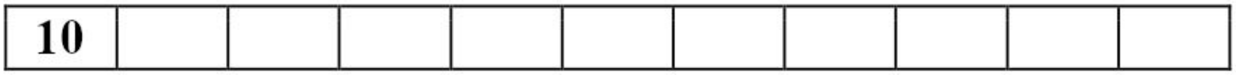
10.

For node at position i

Parent position: (i - 1) / 2
Child 1 position: 2i
Child 2 position: 2i + 1

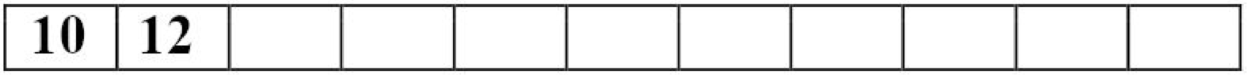
11.

After insert(10):

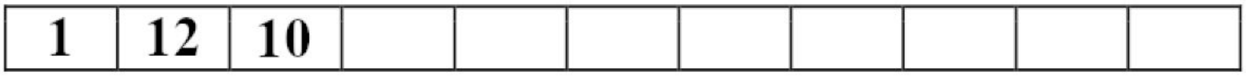


| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | | | | | | | | | | |

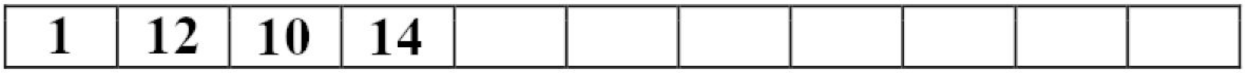
After insert (12):



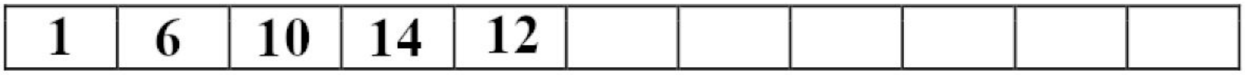
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 12 | | | | | | | | | |



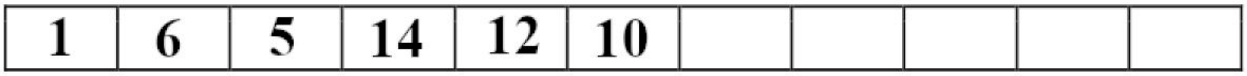
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 12 | 10 | | | | | | | | |



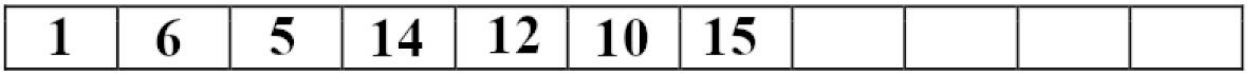
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 12 | 10 | 14 | | | | | | | |



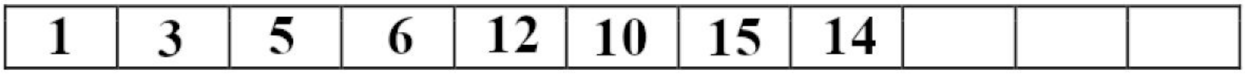
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 10 | 14 | 12 | | | | | | |



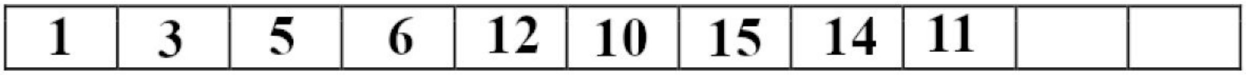
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 5 | 14 | 12 | 10 | | | | | |



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 5 | 14 | 12 | 10 | 15 | | | | |

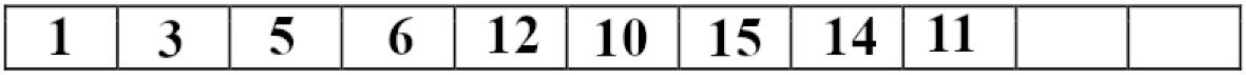


| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 6 | 12 | 10 | 15 | 14 | | | |



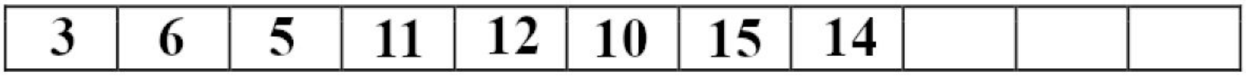
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 6 | 12 | 10 | 15 | 14 | 11 | | |

12.

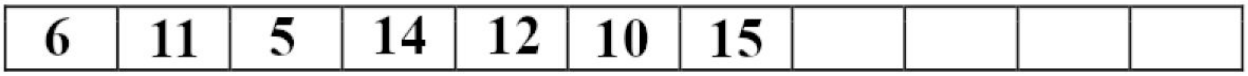


| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 6 | 12 | 10 | 15 | 14 | 11 | | |

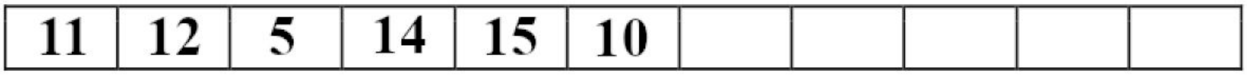
13.



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 6 | 5 | 11 | 12 | 10 | 15 | 14 | | | |

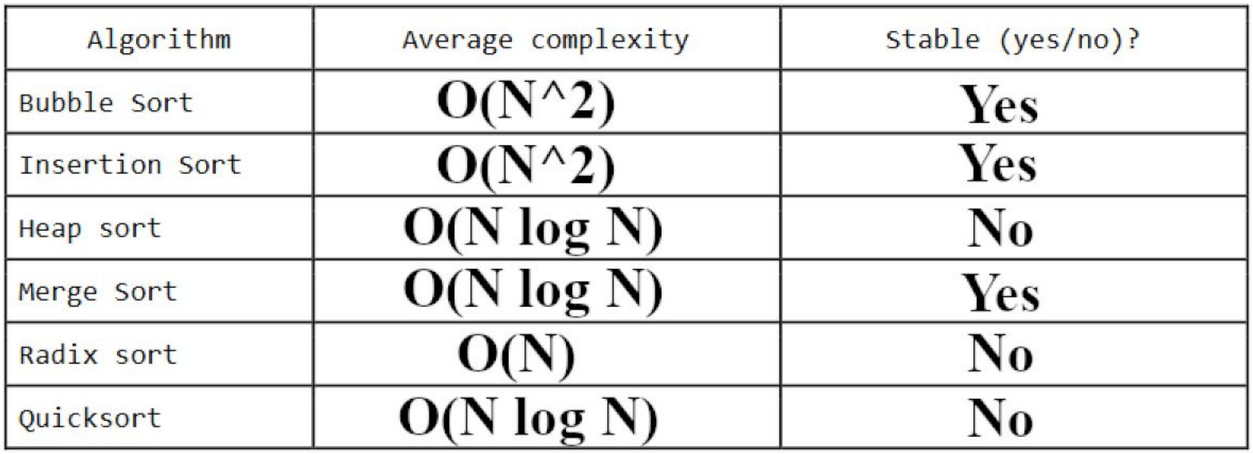


| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 11 | 5 | 14 | 12 | 10 | 15 | | | | |



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 12 | 5 | 14 | 15 | 10 | | | | | |

14.



| Algorithm | Average complexity | Stable (yes/no)? |
|---|---|---|
| Bubble Sort | O(N^2) | Yes |
| Insertion Sort | O(N^2) | Yes |
| Heap sort | O(N log N) | No |
| Merge Sort | O(N log N) | Yes |
| Radix sort | O(N) | No |
| Quicksort | O(N log N) | No |

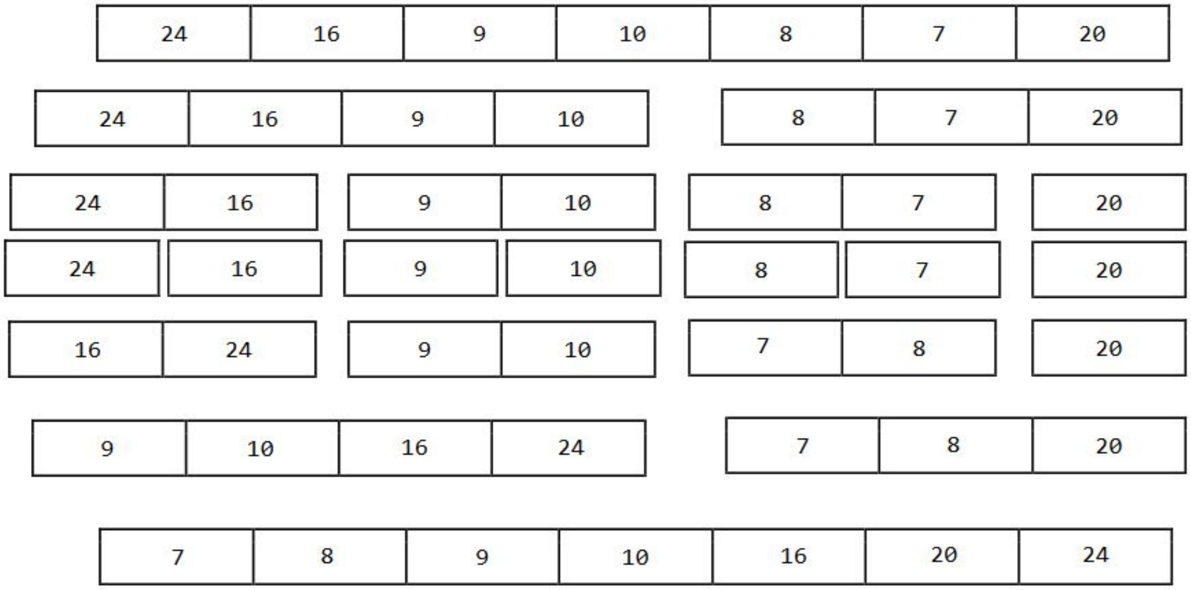
15.

Merge sort and quick sort have best case and average case time complexities of O(N log N). However, they have different worst case time complexities. Merge sort has worst case time complexity of O(N log N), while quick sort has a worst case time complexity of O(N^2).

Meaning, that if only time complexity is considered, merge sort is preferable.
However, merge sort requires an auxiliary array taking up more space.

So if space is limited, quicksort may be a better option, if worst case complexity is likely.

16.



| | | | | | | |
|---|---|---|---|---|---|---|
| 24 | 16 | 9 | 10 | 8 | 7 | 20 |

| | | | |
|---|---|---|---|
| 24 | 16 | 9 | 10 |

| | | |
|---|---|---|
| 8 | 7 | 20 |

| | |
|---|---|
| 24 | 16 |

| | |
|---|---|
| 9 | 10 |

| | |
|---|---|
| 8 | 7 |

| |
|---|
| 20 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 24 | 16 | 9 | 10 | 8 | 7 | 20 |

| | |
|---|---|
| 16 | 24 |

| | |
|---|---|
| 9 | 10 |

| | |
|---|---|
| 7 | 8 |

| |
|---|
| 20 |

| | | | |
|---|---|---|---|
| 9 | 10 | 16 | 24 |

| | | |
|---|---|---|
| 7 | 8 | 20 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 8 | 9 | 10 | 16 | 20 | 24 |

17.

| | | | | | | |
|---|---|---|---|---|---|---|
| 24 | 16 | 9 | 10 | 8 | 7 | 20 |

**move pivot to end:**

| | | | | | | |
|---|---|---|---|---|---|---|
| 24 | 16 | 9 | 8 | 7 | 20 | 10 |

**item from left: 24 | item from right: 7**

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 16 | 9 | 8 | 24 | 20 | 10 |

**item from left: 16 | item from right: 8**

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 8 | 9 | 16 | 24 | 20 | 10 |

**item from left and right cross, so swap left and pivot**

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 8 | 9 | 10 | 24 | 20 | 16 |

**item from left and right cross, so swap left and pivot**

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 8 | 9 | 10 | 16 | 20 | 24 |

**no item from left, so the list is sorted**