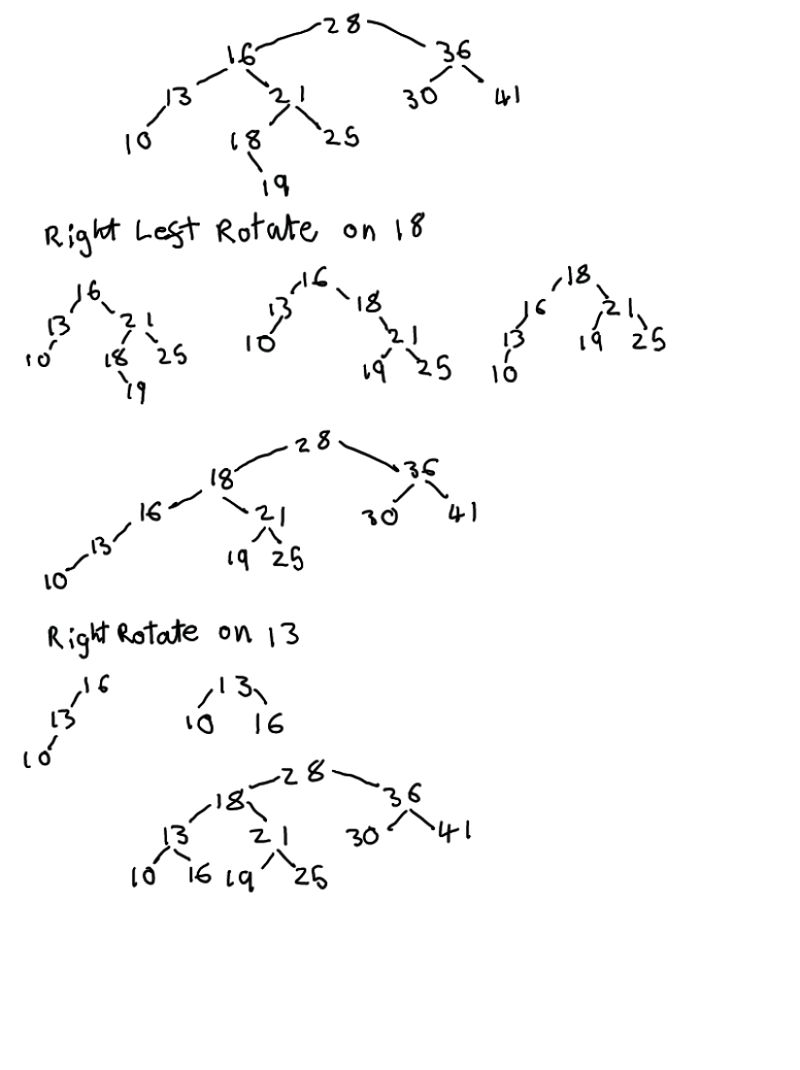
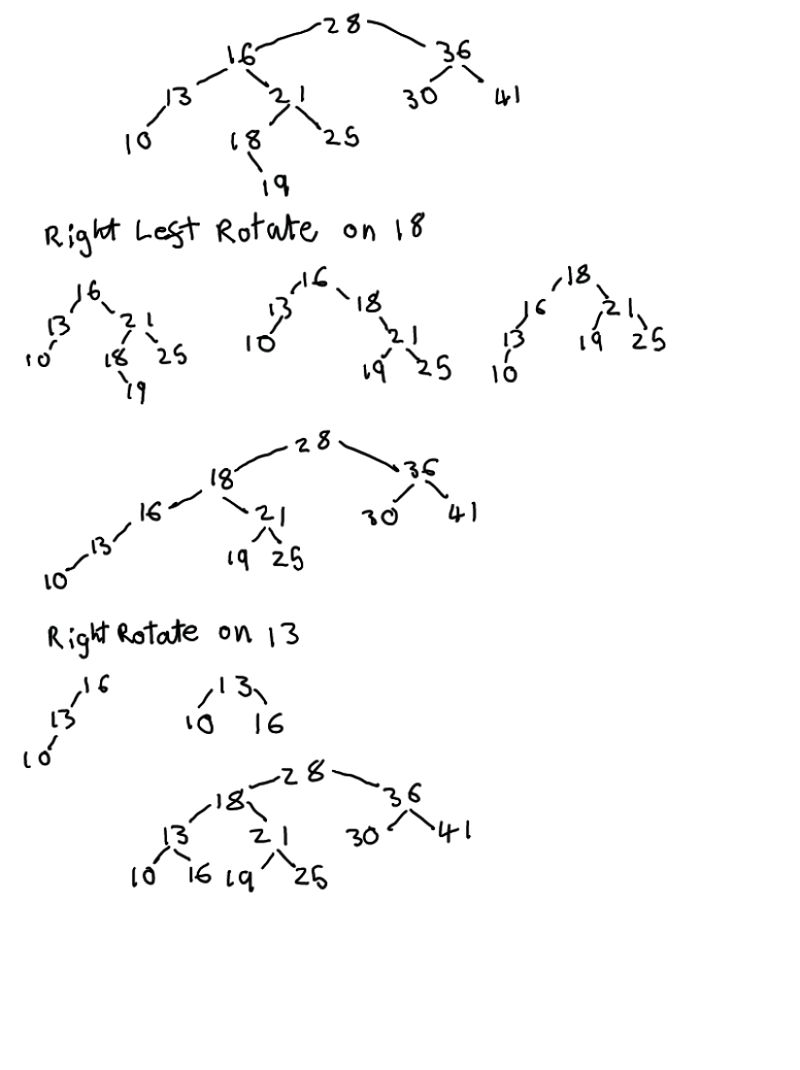
# Data Structures And Algorithms - Assignment 2

## Part 1

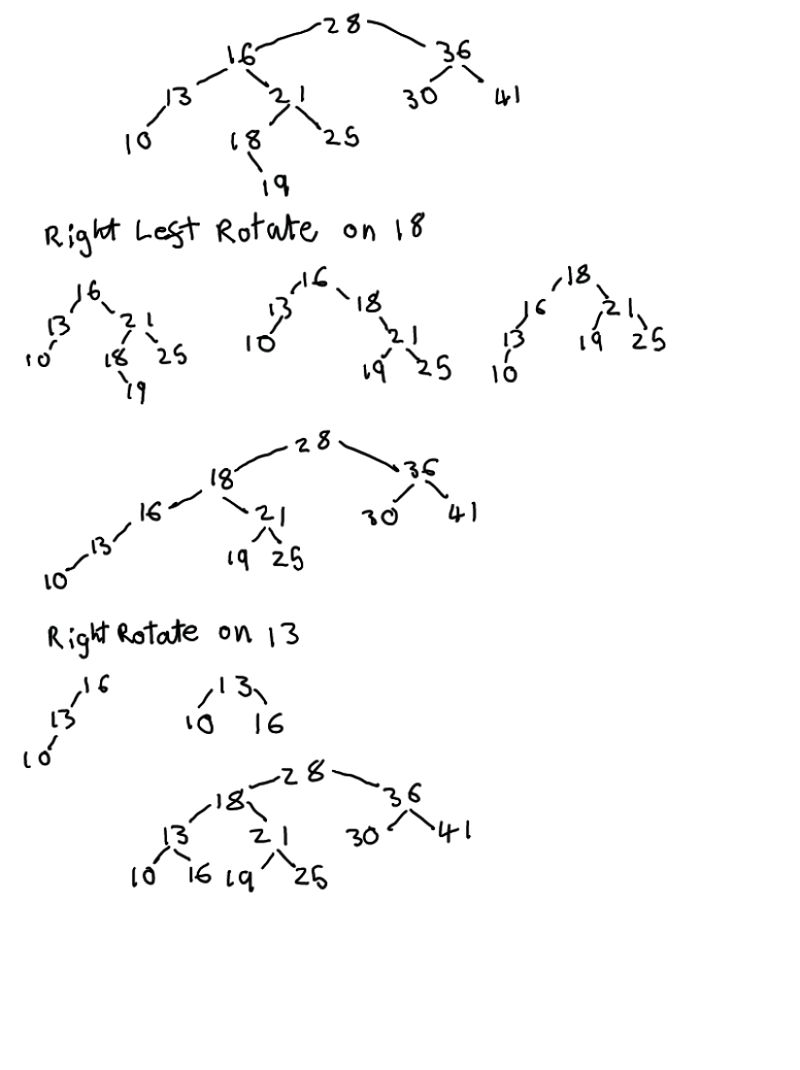
1)

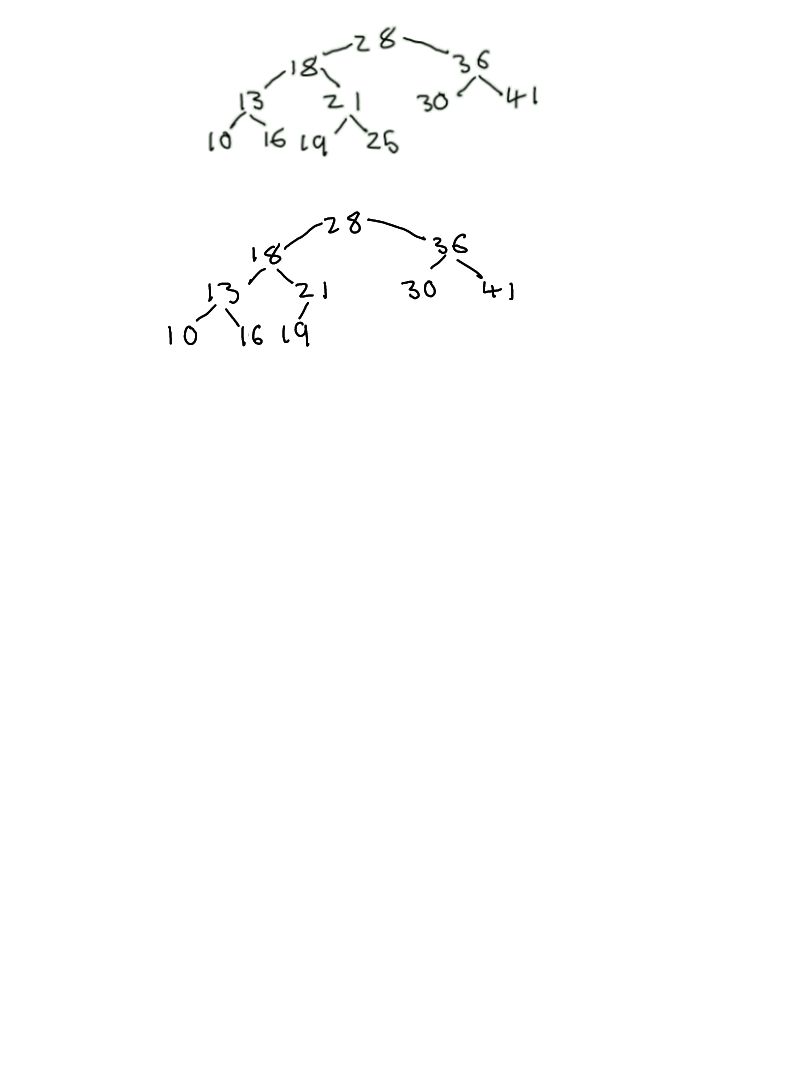
BST for 28,16,36,13,30,10,21,41,18,25

Insert 19



To rebalance this tree first Right Left rotate on 18, then right rotate on 13. As shown below:

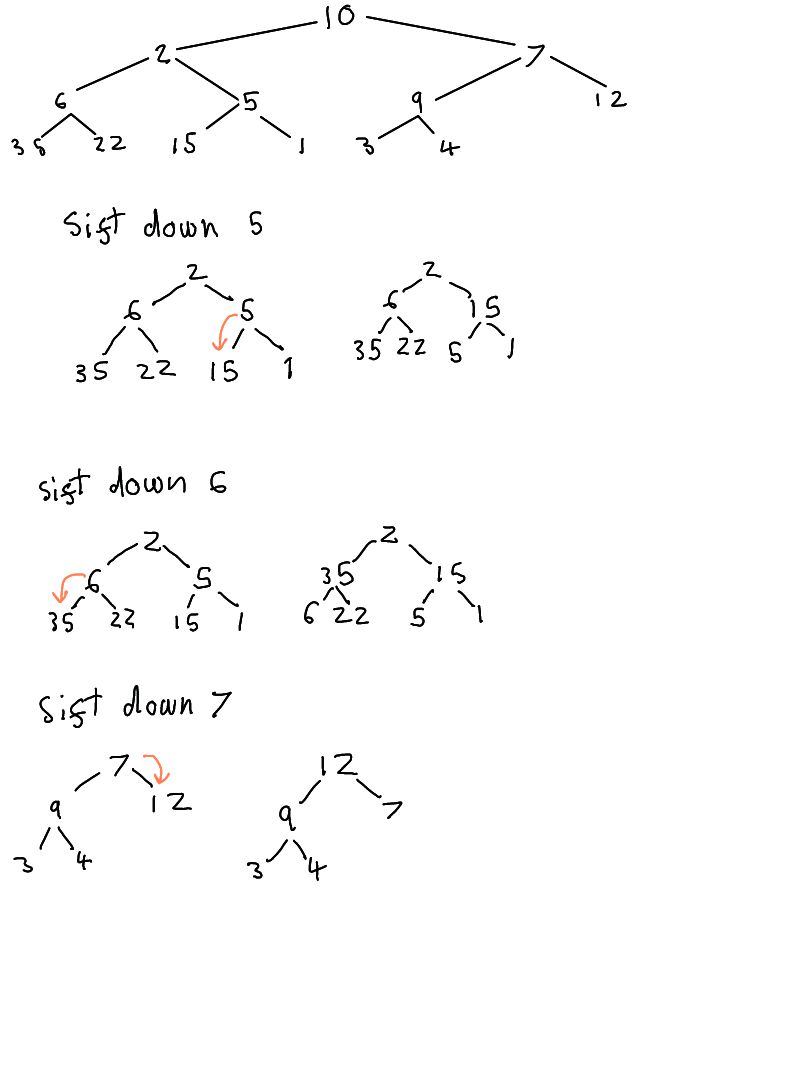


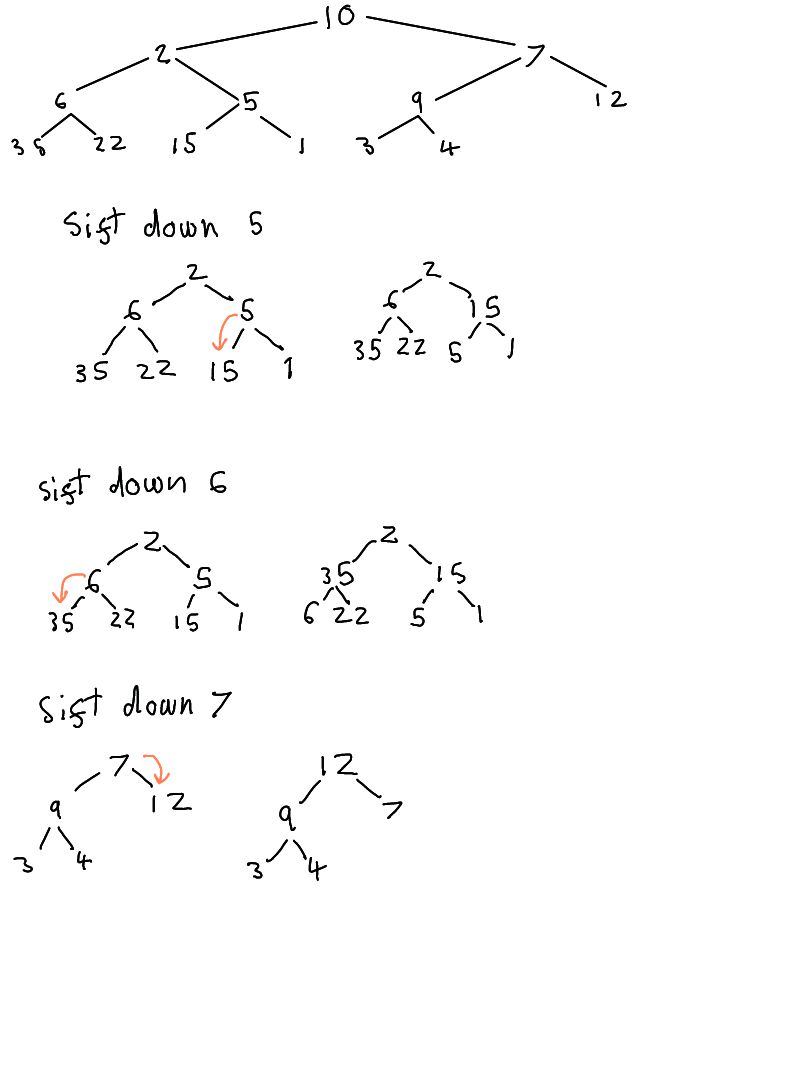
Delete 25 – After the delete the tree is still balanced so no re-balancing operations are required.

## 2)

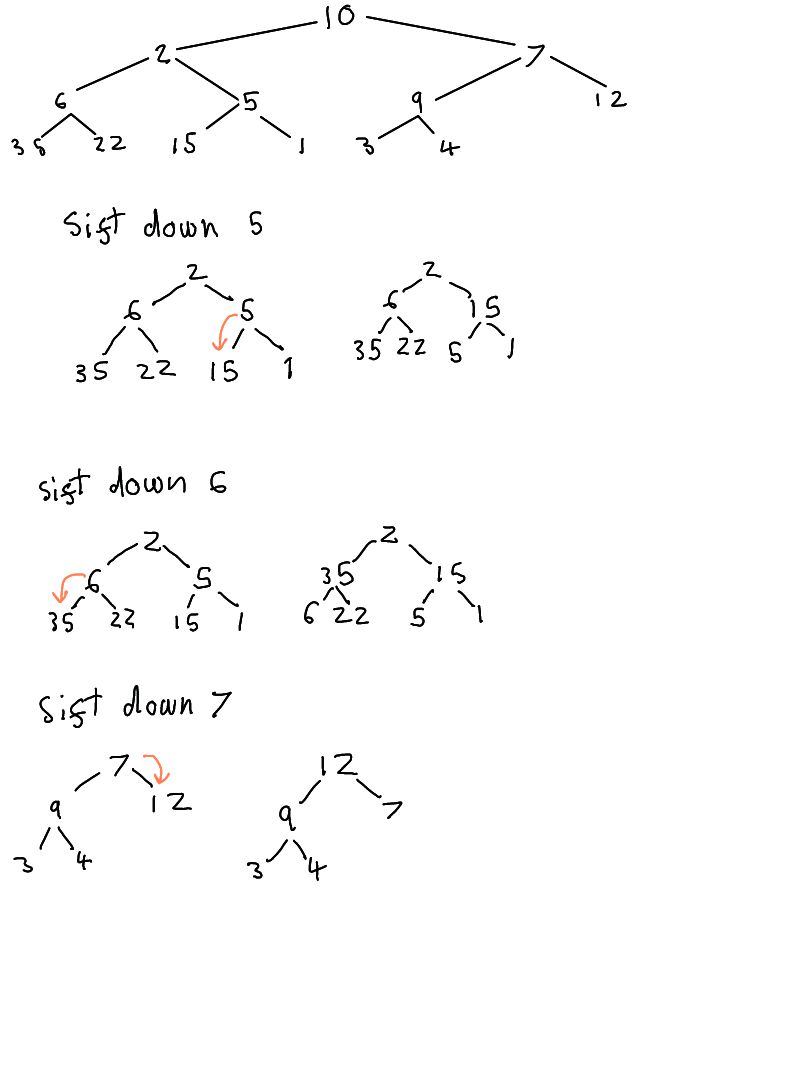
a) Complete Binary tree

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 10 | 2 | 7 | 6 | 5 | 9 | 12 | 35 | 22 | 15 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |



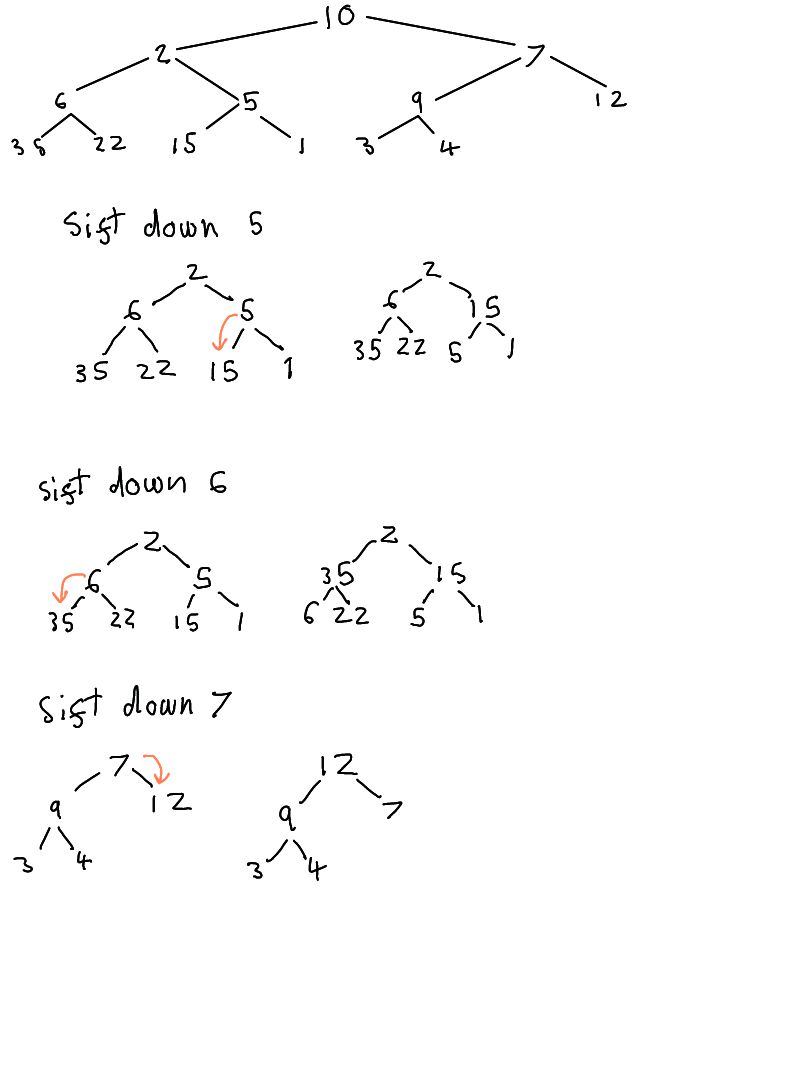
b) Heapify using sift down

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 10 | 2 | 7 | 6 | 15 | 9 | 12 | 35 | 22 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

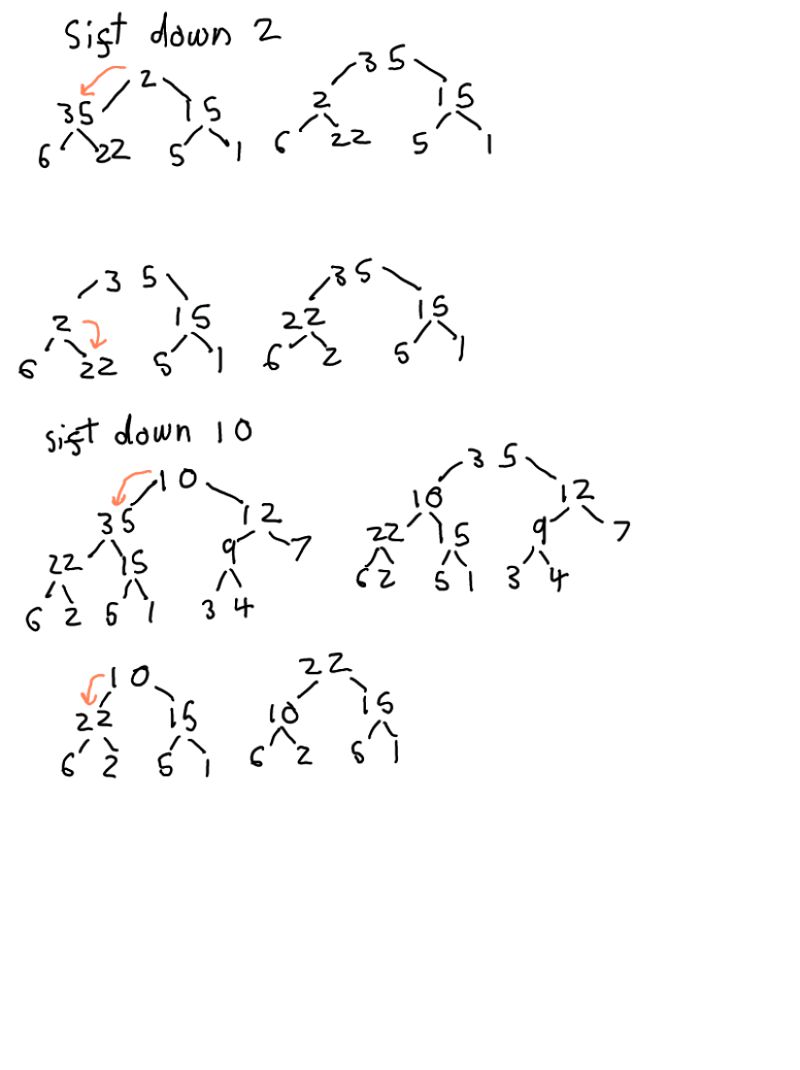


|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 10 | 2 | 7 | 35 | 15 | 9 | 12 | 6 | 22 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

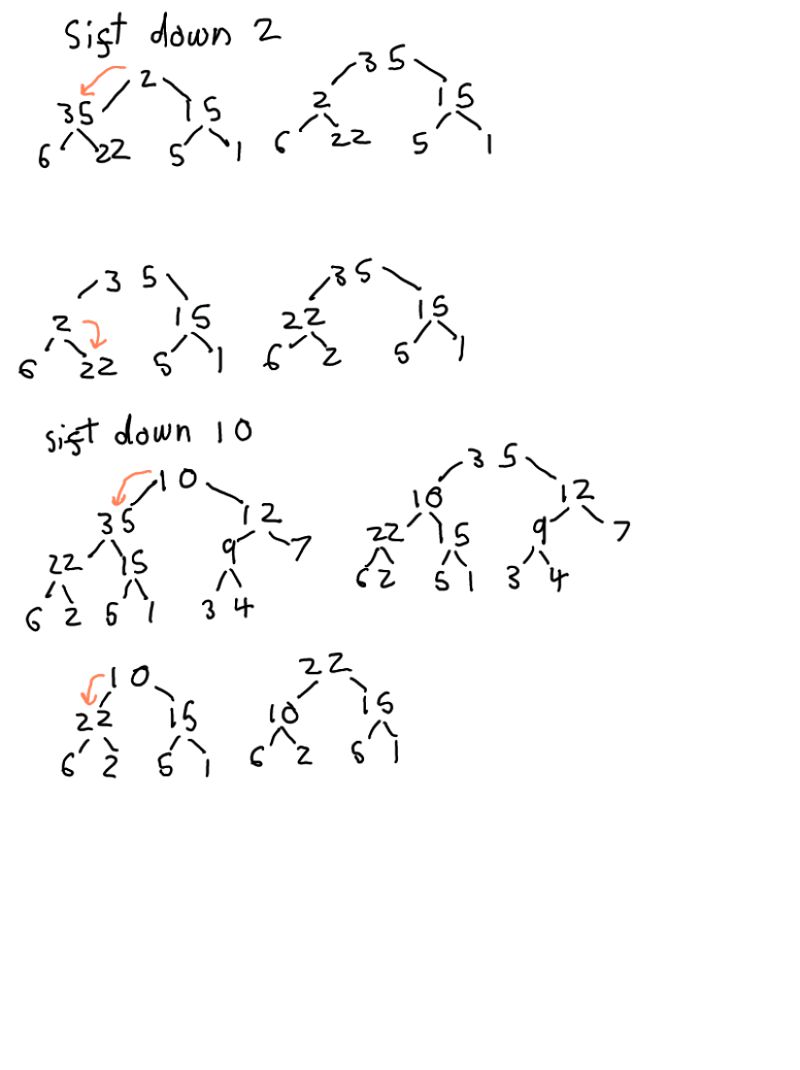
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 10 | 2 | 12 | 35 | 15 | 9 | 7 | 6 | 22 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

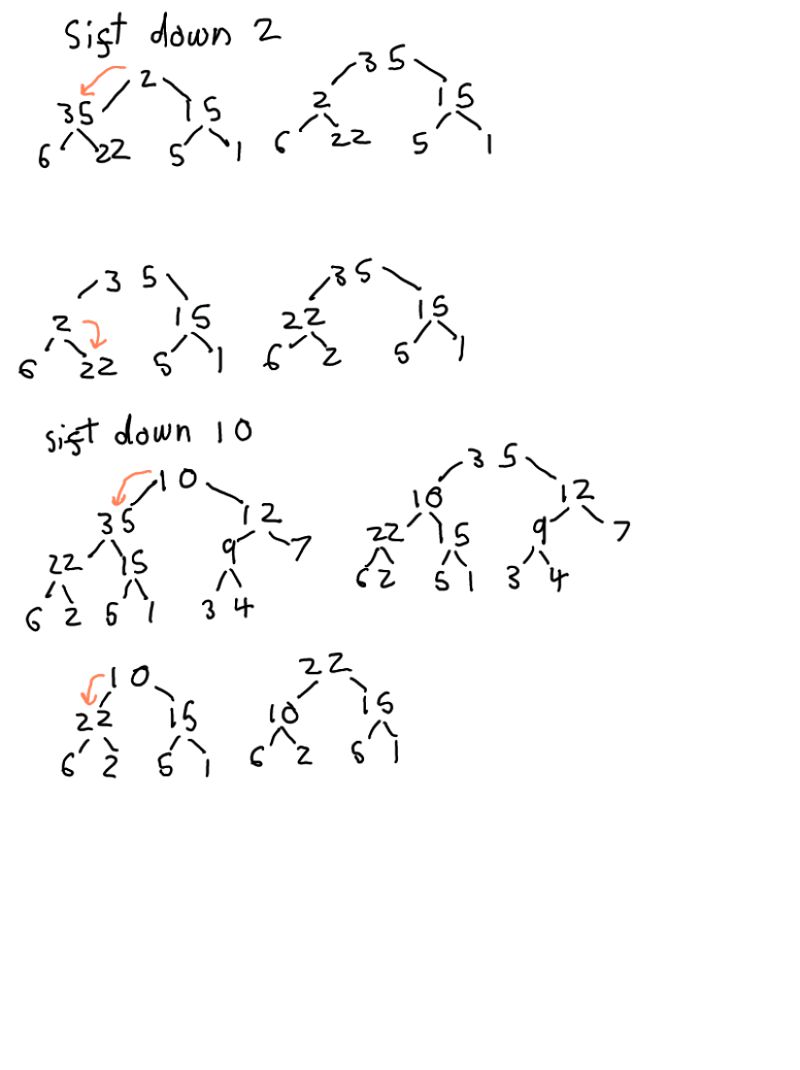


|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 10 | 35 | 12 | 2 | 15 | 9 | 7 | 6 | 22 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

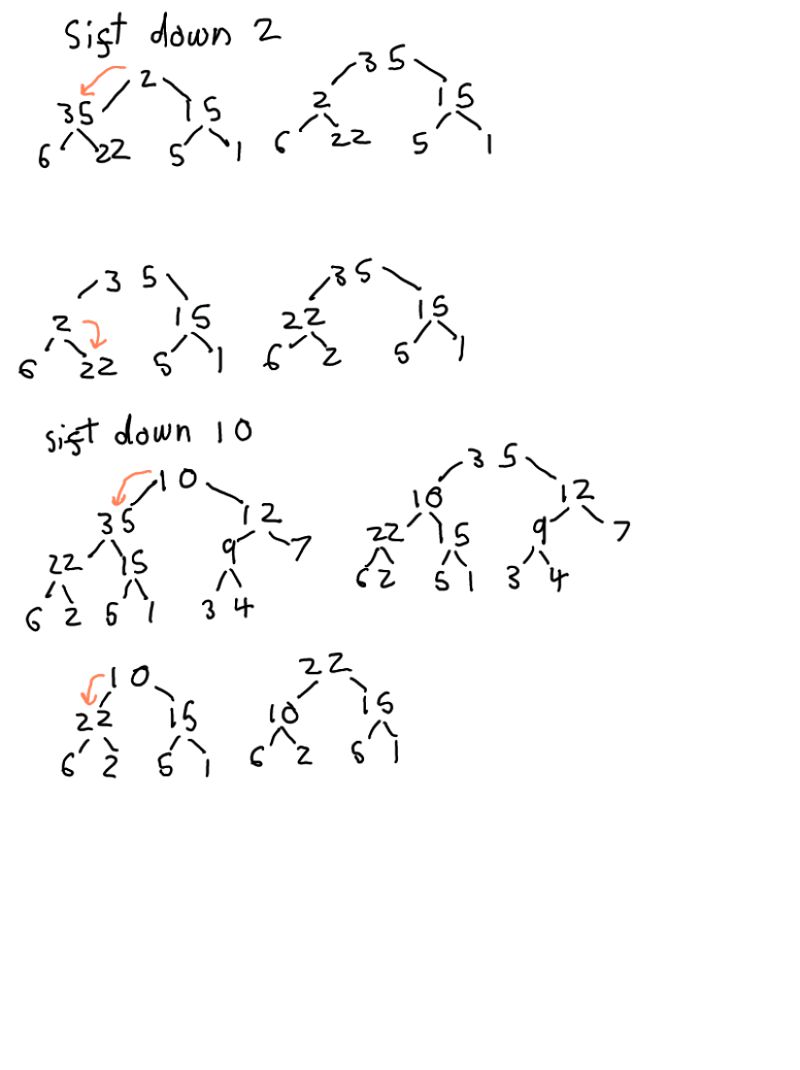


|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 10 | 35 | 12 | 22 | 15 | 9 | 7 | 6 | 2 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

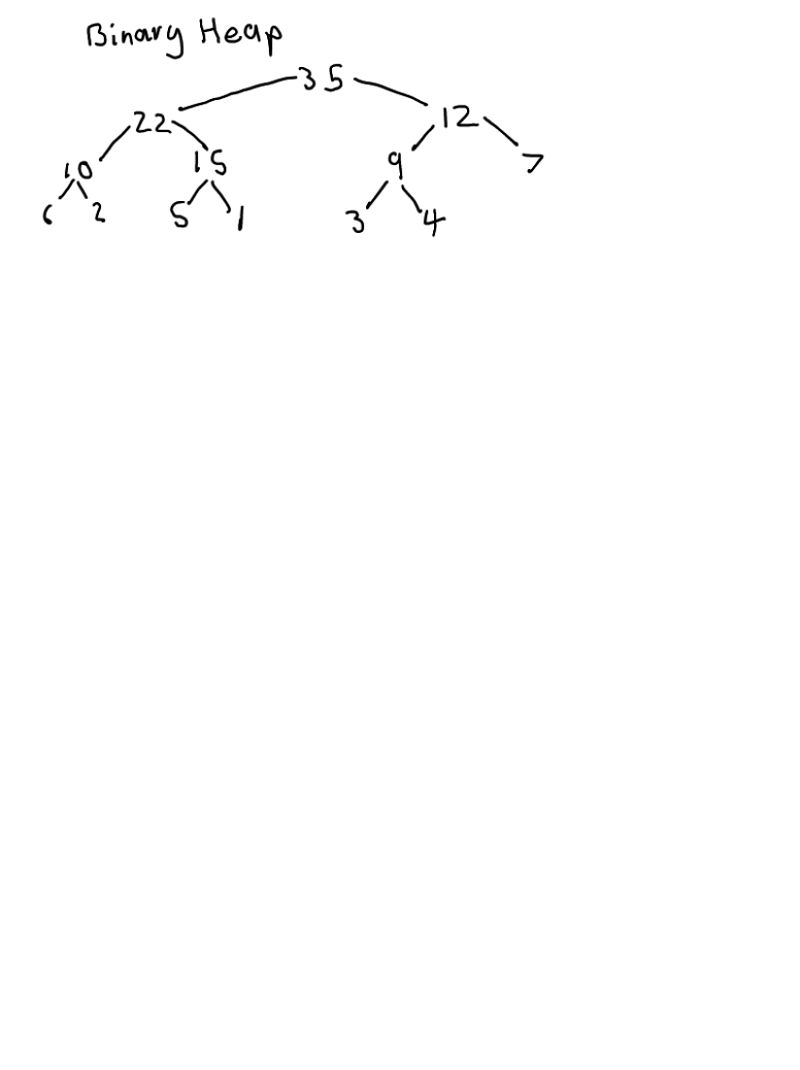




|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 35 | 10 | 12 | 22 | 15 | 9 | 7 | 6 | 2 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

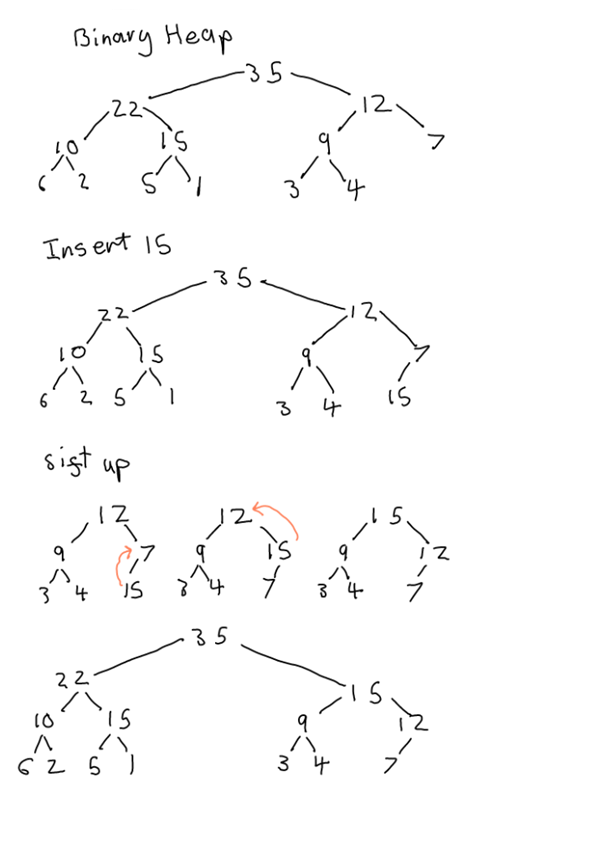


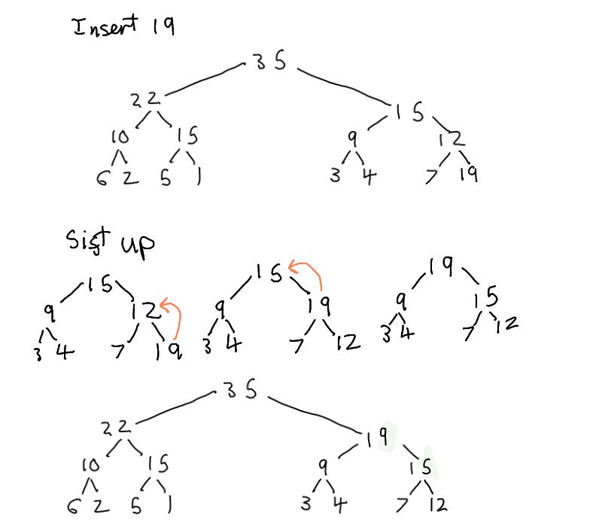
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 35 | 22 | 12 | 10 | 15 | 9 | 7 | 6 | 2 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

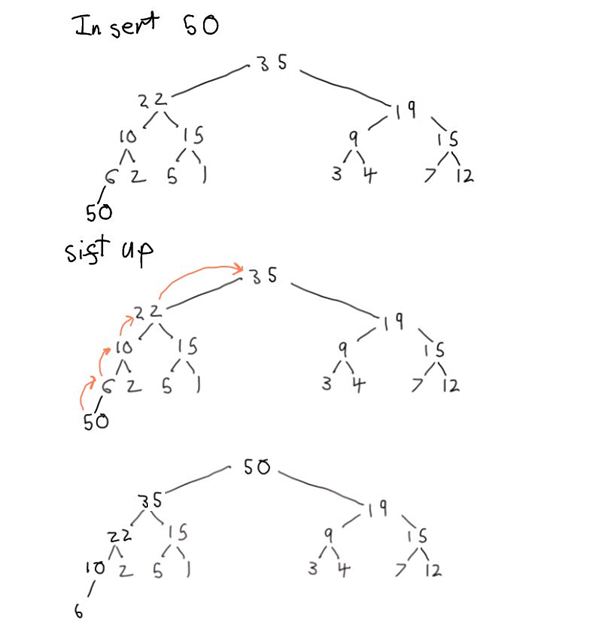


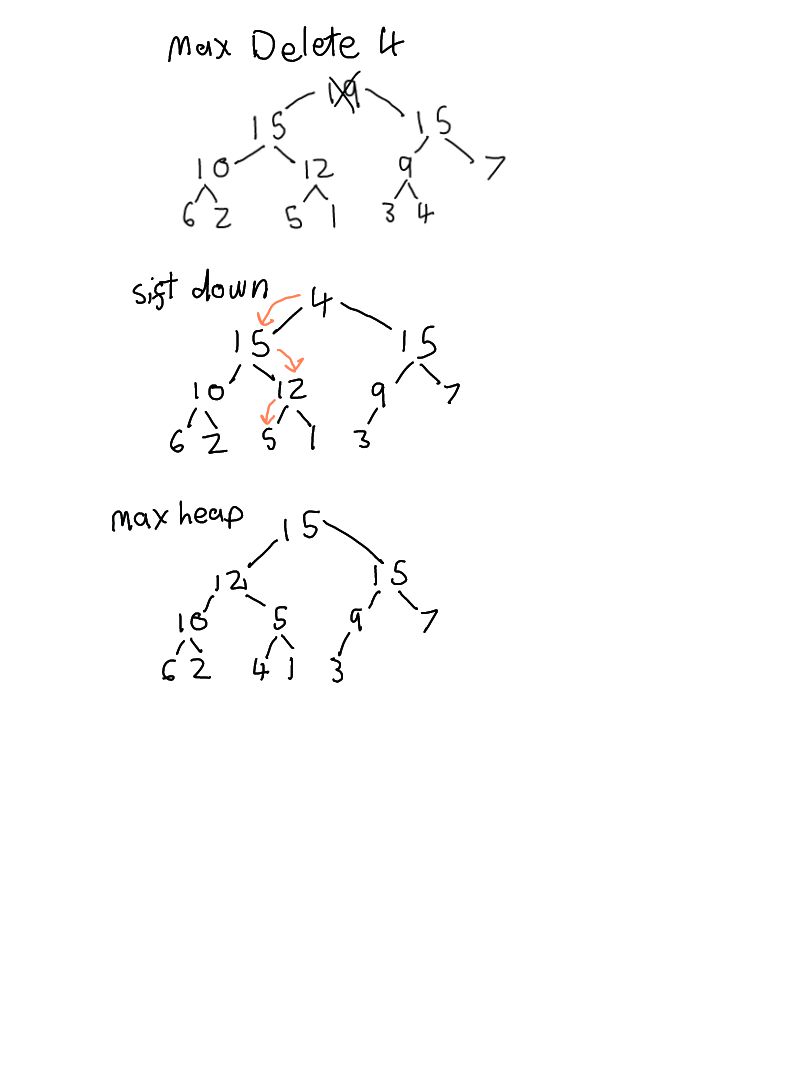
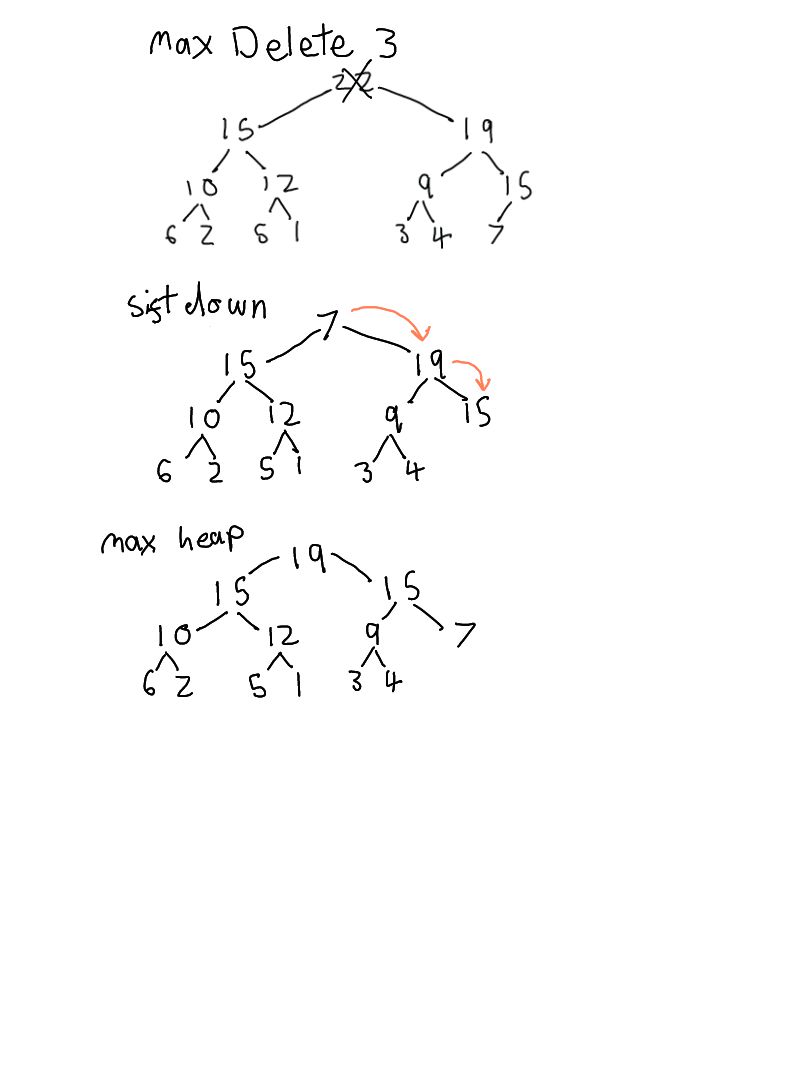
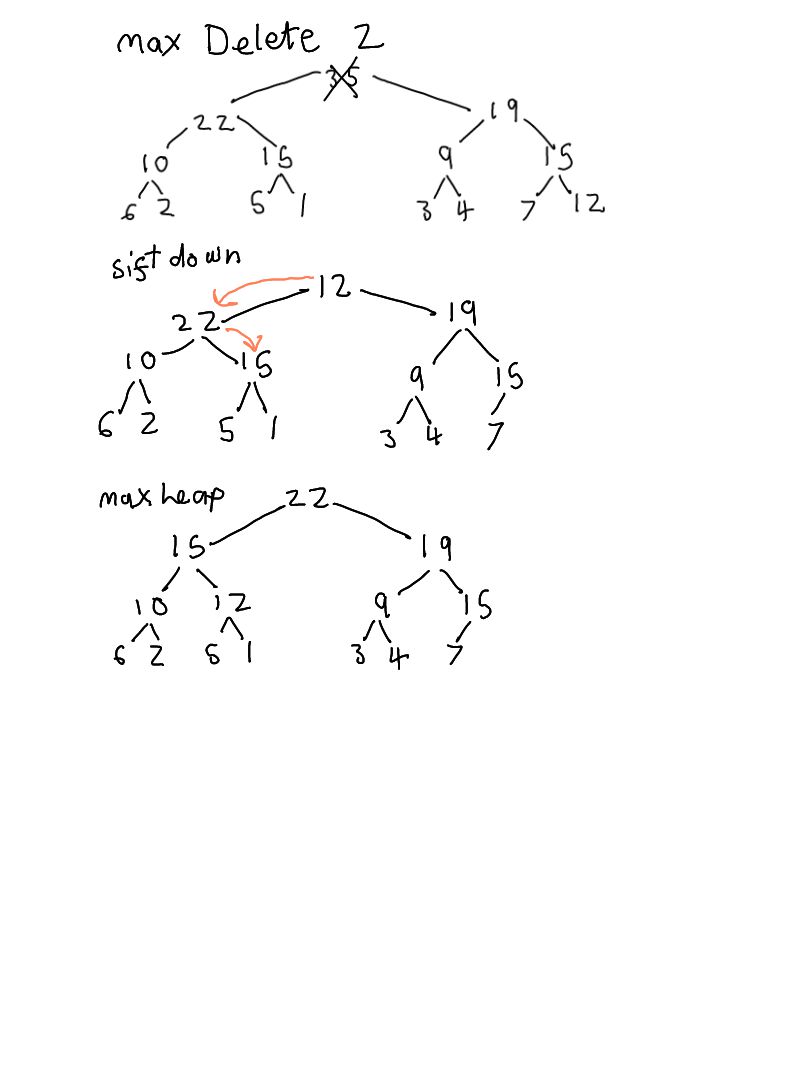
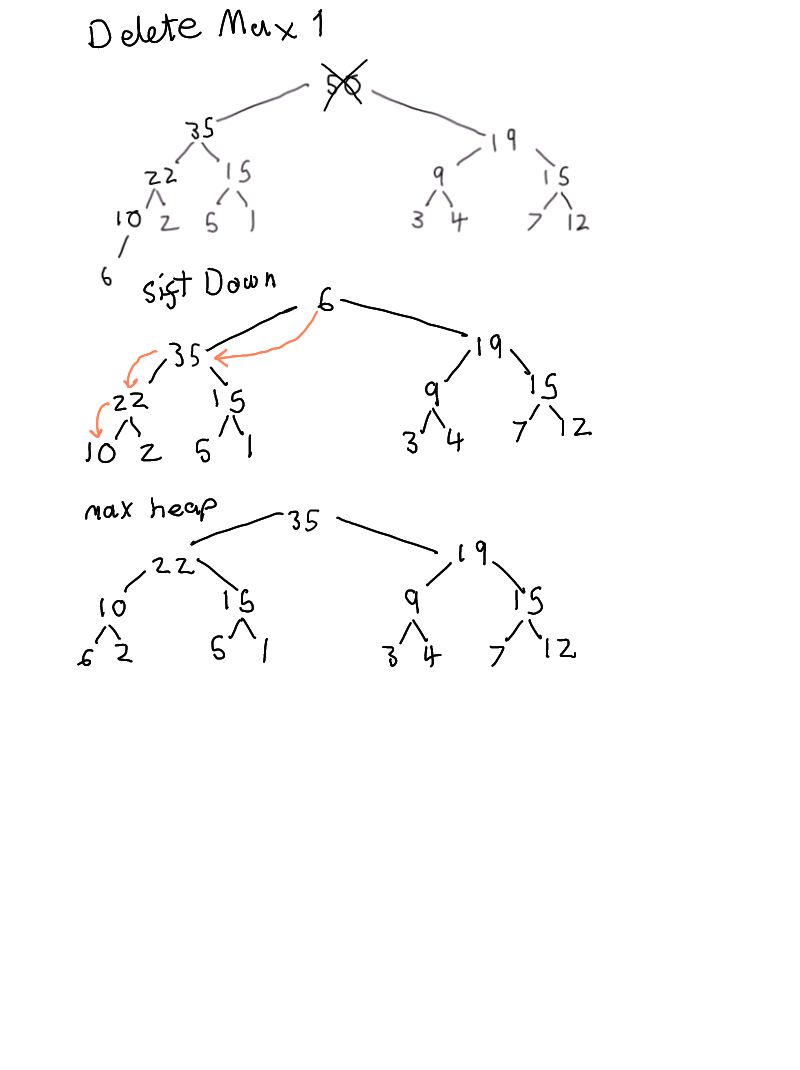
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 35 | 22 | 12 | 10 | 15 | 9 | 7 | 6 | 2 | 5 | 1 | 3 | 4 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

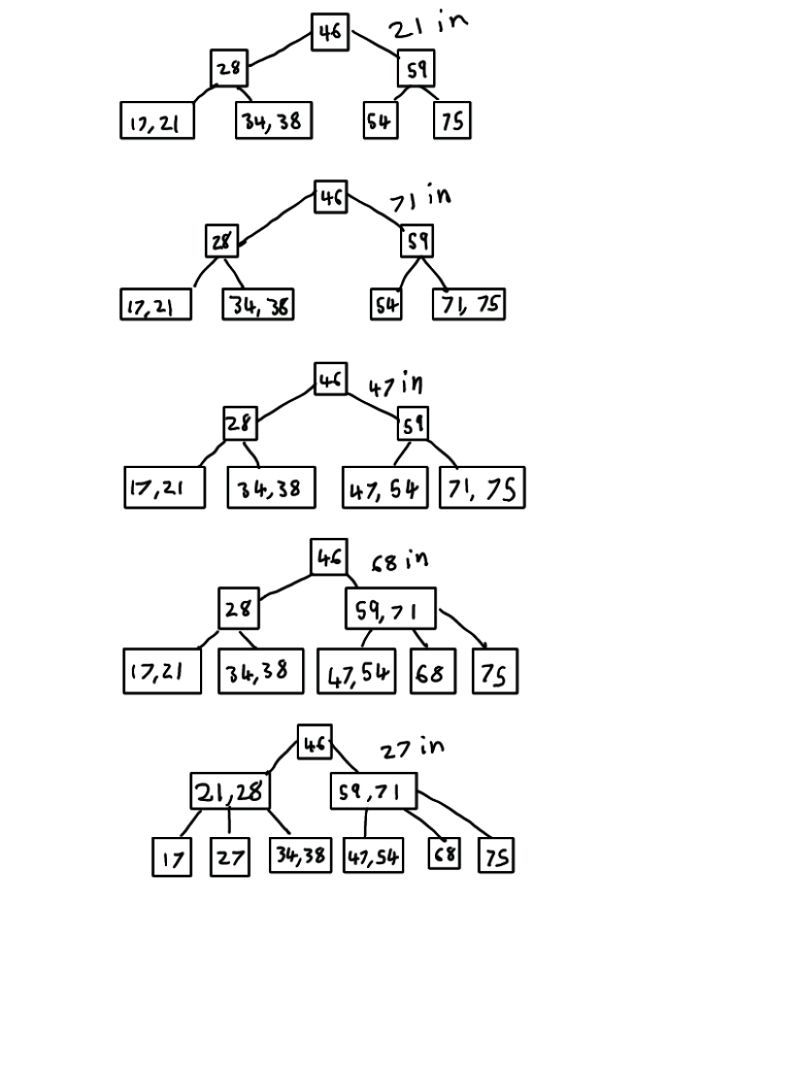
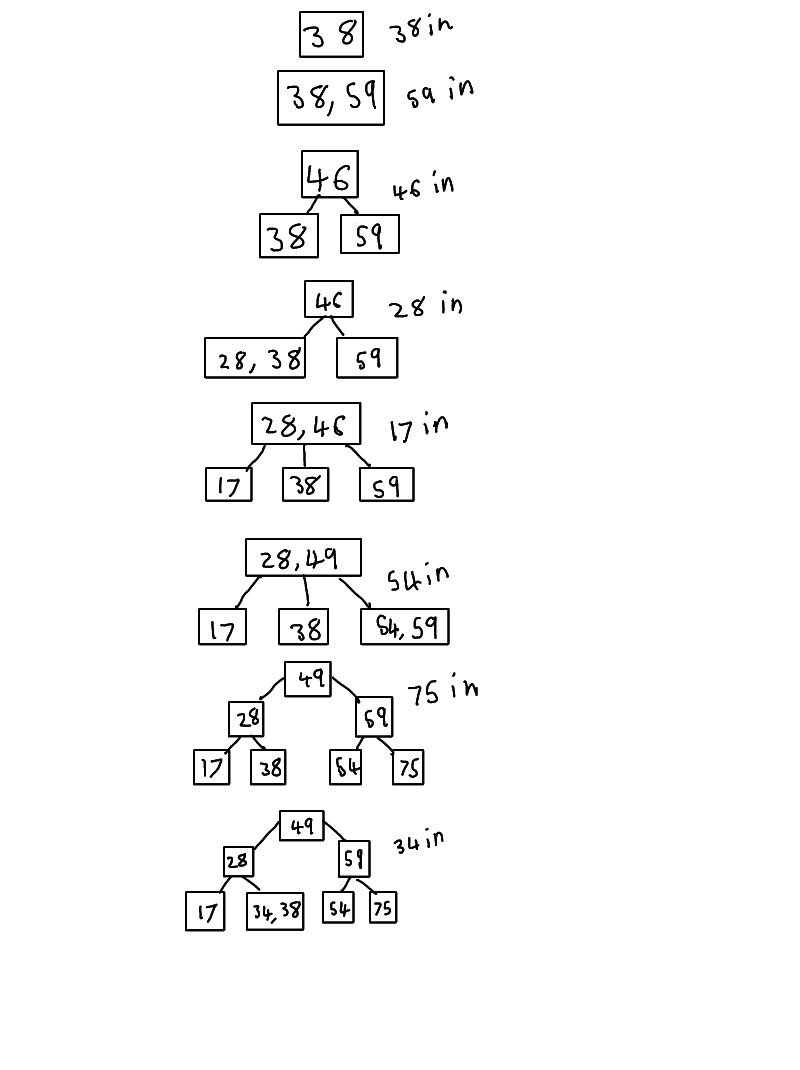
c)







d) Delete Max 4 times

3) 2-3 Tree for 38,59,46,28,17,54,75,34,21,71,47,68,27

## 4)

a) Hash Table, quadratic probing to resolve collisions.

Keys: 8, 42, 29, 51, 47, 13, 34

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Value (k)** | **8** | **42** | **29** | **51** | **47** | **13** | **34** |
| k%13 | 8 | 3 | 3 | 12 | 8 | 0 | 8 |
| (( k%13)+1\*1)%13 |  |  | 4 |  | 9 |  | 9 |
| (( k%13)+2\*2)%13 |  |  |  |  |  |  | 12 |
| (( k%13)+3\*3)%13 |  |  |  |  |  |  | 4 |
| (( k%13)+4\*4)%13 |  |  |  |  |  |  | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Value** | **13** |  |  | **42** | **29** |  |  |  | **8** | **47** |  | **34** | **51** |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

b) Hash Table, double hashing to resolve collisions with probe increment of p(k) = 5−(k%5).

Keys: 8, 42, 29, 51, 47, 13, 34

h(k) = k%13

p(k) = 5 − (k%5)

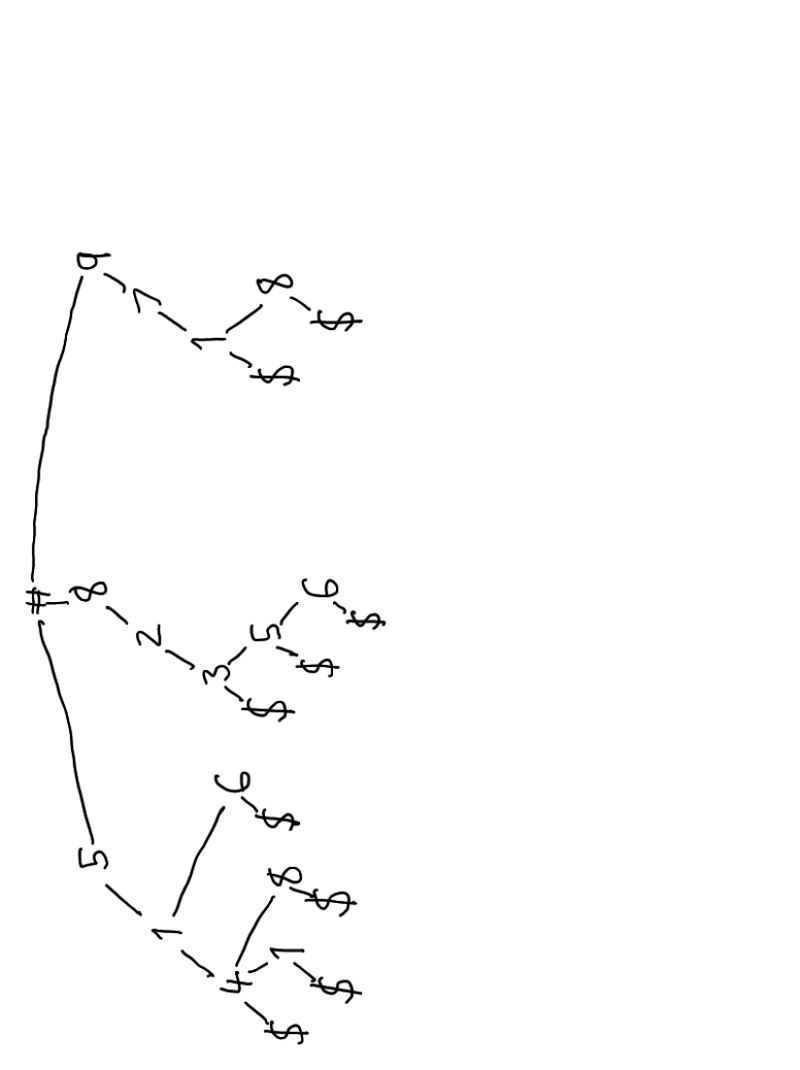
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Value (k)** | **8** | **42** | **29** | **51** | **47** | **13** | **34** |
| h(k) | 8 | 3 | 3 | 12 | 8 | 0 | 8 |
| p(k) |  |  | 2 |  | 2 |  | 2 |
| (h(k)+1\*p(k))%13 |  |  | 5 |  | 10 |  | 10 |
| (h(k)+2\*p(k))%13 |  |  |  |  |  |  | 12 |
| (h(k)+3\*p(k))%13 |  |  |  |  |  |  | 1 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Value** | **13** | **34** |  | **42** |  | **29** |  |  | **8** |  | **47** |  | **51** |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

## 5)

a) Draw the trie corresponding to S.

S = {514, 5141, 5148, 516, 823, 8235, 82356, 971, 9718}



b) Represent the trie from (a) using a two-dimensional array.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  | (3) |  | 5141 |  |  |  |  |  | (11) |  |
| 2 |  |  |  |  | (6) |  |  |  |  |  |  |
| 3 |  |  |  |  |  | (7) |  |  |  |  |  |
| 4 |  |  | (4) |  |  |  |  |  |  |  |  |
| 5 | (2) |  |  |  |  |  | (8) |  |  |  |  |
| 6 |  |  | 516 |  |  |  |  | 82356 |  |  |  |
| 7 |  |  |  |  |  |  |  |  | (10) |  |  |
| 8 | (5) |  |  | 5148 |  |  |  |  |  |  | 9718 |
| 9 | (9) |  |  |  |  |  |  |  |  |  |  |
| $ |  |  |  | 514 |  |  | 823 | 8235 |  |  | 971 |

## Part 2

1 a)

### Informal Description

Pass the count method an array called **SearchArray**. Have an integer that keeps track of the count(set to 0), and another that keeps track of the dominant element(set to the first element in searchArray).

First, loop through the searchArray and start with the first element as the dominant element with its count set to 1, then if the next element is the same add 1 to the counter and if it is different subtract 1 from the counter. If the count hits 0 then the dominant element is set to the next element and its count is set to 1. If a value is dominant in the array then it will be set as the dominant element by the end of the loop.

Then Loop through searchArray again to find the count of the dominant element found in the first loop by setting the count to 0 and adding 1 to it each time the dominant element is found in the array. Finally, check if the count is greater than searchArrays length / 2, if it is then return the dominant element, otherwise return a negative number to indicate a failure.

### Formal Description - Pseudo Code

**SearchArray** is passed to the method as a parameter

Initialize **count** to zero

Initialize **dominantElement** to first element in **SearchArray**

FOR each int in **SearchArray**

IF **count** equals zero THEN

Set **dominantElement** to the current int of **SearchArray**

Set **count** equal to 1

ELSE IF **dominantElement** equals current int of **SearchArray** THEN

Add one to **count**

ELSE

Subtract one form **count**

ENDIF

ENDFOR

Set **count** to zero

FOR each int in **SearchArray**

IF **dominantElement** equalscurrent int of **SearchArray** THEN

Add one to **count**

ENDIF

ENDFOR

IF **count** is greater than the length of **SearchArray** THEN

Return **dominantElement**

ELSE

Return -1

ENDIF

### Space and Run-Time Complexity

#### Space Complexity

This algorithm only relies on one array of size n, so the space complexity of this algorithm is constant and is O(n)

#### Worst case Run-Time Complexity

The worst case Run-Time complexity of this algorithm is O(n)

**Fundamental Operation**

searchArray[i] == dominantElement

**Run-time complexity function**

The algorithm has 2 for loops that are based on the length of the searchArray that are not nested and all of the comparisons done are constant time, so run time complexity will be:

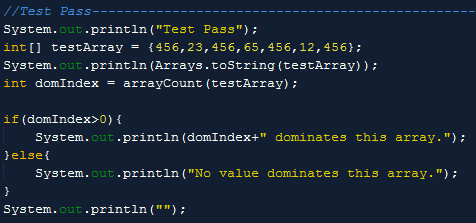
t(n) = n+n+c

= 2n+c

=O(n)

## Test

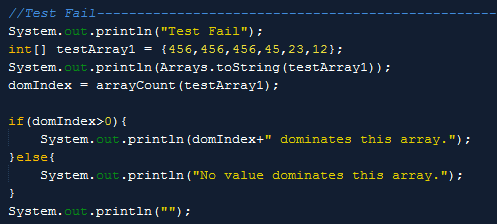
### Pass



### Output



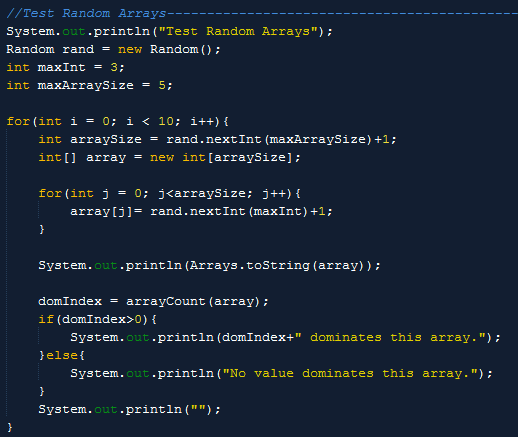
### Fail



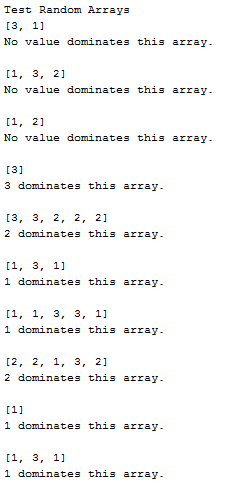
### Output



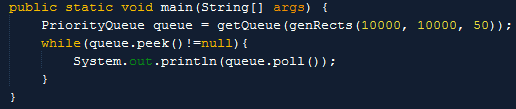
### Test Random



### Output



## Rectangle Cut Test



### Output

