# CS 4442b Assignment 1

Danilo Vladicic - Dvladici@uwo.ca - 250861339

February 27, 2020

# Question 1

Let $A[1, ..., n]$ be an array storing n integers within the range [0, k]

```
Input: Array A, k is the max element
Output: Array B[1, ..., k] storing the count of
        elements less than or equal to i
preprocess(A, k):
    Let B[1, ..., k] be an array of 0's
    for i = 1 to A.length:
        B[A[i]] = B[A[i]] + 1
    for j = 2 to k:
        B[j] = B[j] + B[j - 1]
    return B

Input: a, b represent the start and end indices respectively.
       B is the preprocessed array created from a call to
       preprocess(A)
Output: The number of integers in A between the range [a, b]
QUERY(B, a, b):
    if(b > k) b = k;
    if(a == 0) return B[b]
    return B[b] - B[a - 1]
```

Preprocess runs through all the elements n one time, and then it runs another k-1 times.Thus the time complexity is $T(n) = n + k - 1$. Since the loops execute only a constant number of instructions each, it is easy to see that the time complexity is both is lower and upper bound.

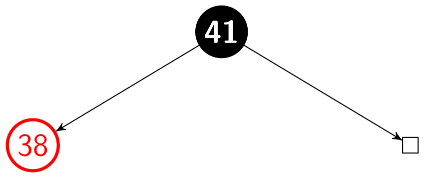
Query only executes a constant number of instructions regardless of a or b and thus it is O(1).

# Question 2

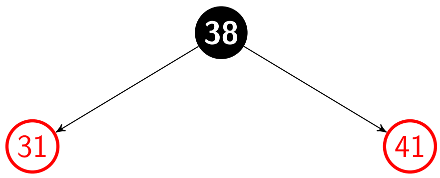
Note: The NIL pointers are removed The RB tree generated by inserting: 41,38,31,12,19,18

Insert 41

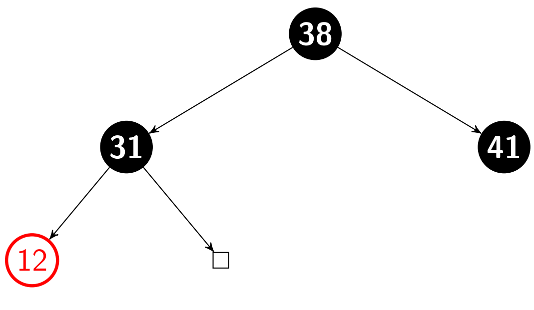

Insert 38



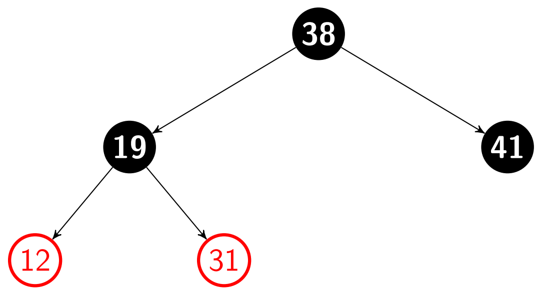

Insert 31



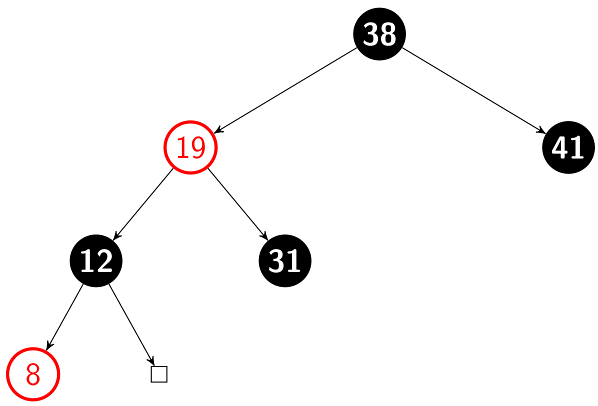

Insert 12




Insert 19

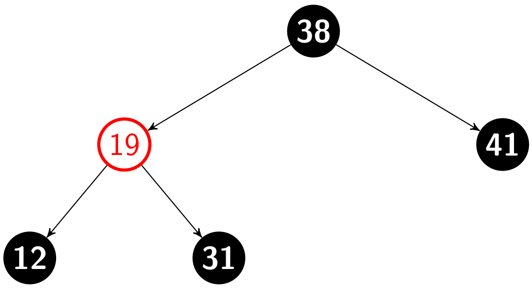



Insert 8

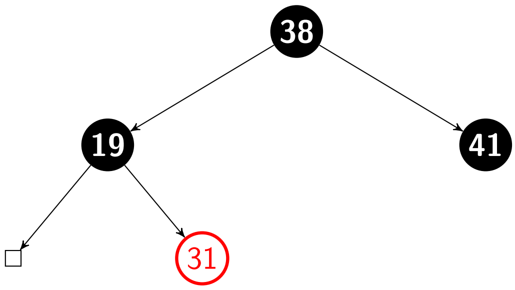


## Question 2

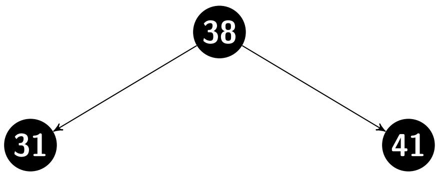
Remove elements in order: 8, 12, 19, 31, 38, 41
Remove 8



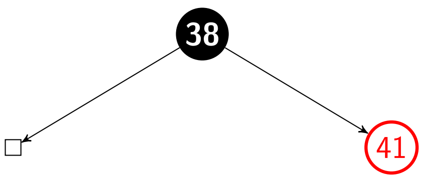

Remove 12




Remove 19




Remove 31



Remove 38


Finally removing 41 leaves an empty tree.

# Question 4

Suppose nodes with keys 4, 3, 2 are inserted into an RB tree, in that order. Before inserting the node with key 2, the node with key 4 will be the root of the tree. After inserting the node with key 2, the RB tree will violate the property that every red node's children must be black, since 2 is inserted as red, and its parent 3 is also red. This results in the tree rotating, and the new root of the tree becomes 3. If 2 is removed, the resulting tree will still have key 3 as its root node. Therefore, the resulting tree is not the same as the initial tree.

# Question 5

The height of a Tree is defined as the largest number of edges from the root node to a leaf, where the height of a tree with only on node is 0. For avl tree's, there is also the condition that

$$|height(x.Left) - height(x.Right)| \leq 1$$

for every node x. From this property we can observe some cases. A tree with height 0 has at most 1 node A tree with height 1 has at least 2 nodes. A tree with height 2 has at least 4 nodes. A tree with height 3 has at least 7 nodes. From this we observe that the minimum number of nodes in an AVL tree is

$$N(h) = 1 + N(h-1) + N(h-2)$$

Now consider the Fib sequence $F_x = 0, 1, 1, 2, 3, 5, 8, ....$ We can see that

$$N(h) = F_{h+3} - 1$$

and we know

$$F_k = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$$

Thus,

$$N(h) \geq \frac{(\frac{1+\sqrt{5}}{2})^{h+3} - (\frac{1-\sqrt{5}}{2})^{h+3}}{\sqrt{5}} - 1$$

We can eliminate the negative root by always subtracting 1. So,

$$N(h) \geq \frac{(\frac{1+\sqrt{5}}{2})^{h+3} - 1}{\sqrt{5}} - 1$$

$$\sqrt{5}(N(h) + 1.45) \geq (\frac{1+\sqrt{5}}{2})^{h+3}$$

Let $\phi = \frac{1+\sqrt{5}}{2}$ then taking $\log_\phi$ of both sides results in

$$\log_\phi(\sqrt{5}(N(h) + 1.45)) \geq h + 3$$

$$h \leq \log_\phi(\sqrt{5}(N(h) + 1.45)) - 3$$

And since $\phi < 2$ it can be seen the height of a tree is always

$$h \leq log_2(n)$$

# Question 6

Goal: Given n elements and an integer k, write an algorithm the outputs a sorted sequence of the smalles k elements with time complexity O(n) when $k \log n \leq n$

```
Input: Array A of Size size, and start index i
Output: The minimum heap of A for index i
Heapify(A, size, i):
    while(i ≤ ⌊size/2⌋):
        if(2i + 1 < size and A[2i + 1] < A[2i])
            smallest = 2i + 1
        else
            smallest = 2i
        if(A[i] < A[smallest]) break
        else
            tmp = A[smallest]
            A[smallest] = A[i]
            A[i] = tmp
            i = smallest
```

```
Input: Array A
Output: The minimum heap of A for all indices
BUILD-MINIMUM-HEAP(A):
    size = A.length
    for i = ⌊size/2⌋ to 1
        Heapify(A, size, i)
```

This builds a heap in O(n) time. Finally

```
Input: Array A, and the integer k
Output: The minimum k values from A
SORT-SMALLEST(A, k):
    BUILD-MINIMUM-HEAP(A)
    Array T[1,...,k]
    for i = 1 to k
        T[i] = A[1]
        A[1] = A[A.length - i]
        Heapify(A, size - i, 1)
```

Since the BUILD-MINIMUM-HEAP runs in O(n) the loop runs in k times calling Heapify which runs O($\log n$) the total time complexity is O($n+k\log n$) which runs in O(n) time when $k \log n \leq n$.

## Part B

Generalize the algorith to run in O($k \log k + n$) time. To do this we observe that after building the MIN heap, the children of a node, will always be less than the value at that node. So once the initial heap is built, every sub tree is also a heap, meaning the root node of both the subtree will be smaller than all of their children. From this we can see that the smallest k elements will be within nodes B[1, ..., K] where B = BUILD-MINIMUM-HEAP(A).

```
Input: Array A, and the integer k
Output: The minimum k values from A
SORT-SMALLEST(A, k):
    BUILD-MINIMUM-HEAP(A)
    Array B[1,...,k] = [A[1], ... A[k]]
    Array T[1, ..., k] = [0, ..., 0]
    for i = 1 to k
        T[i] = B[1]
        B[1] = B[B.length - i]
        Heapify(B, B.lenght - i, 1)
```

The BUILD-MINIMUM-HEAP will run in O(n) time as before. The loop runs k times, but now Heapify() runs on n = k elements so the overall complexity is $O(n + k \log k)$

# Question 7

Prove every node has rank at most $\lfloor \log n \rfloor$ Thus $rank(x) \leq \lfloor \log n \rfloor$ Let $y = \lfloor \log n \rfloor$ then we know

$$\log n - 1 < y \leq \log n$$

$$\log(n) < \lfloor \log n \rfloor + 1 \leq 1 + \log n$$

and we have

$$rank(Union(x, y)) = Max rank(x), rank(y)$$

when two tree of different ranks are unioned or

$$rank(Union(x, y)) = Max rank(x), rank(y) + 1$$

when the ranks are the same.

Base: A single Node x, has rank 0 when created,

$$rank(x) = \lfloor \log n \rfloor$$

$$rank(x) \leq log 1 = 0$$

Therefore this holds true in the base case. Induction: The only time rank changes is when 2 nodes of the same rank are unioned. For two nodes to have the same rank and highest rank each would undergo an i amount of unions. First you would union all nodes with rank 0, then union all nodes with rank 1, then all nodes with rank 2 and so on. So we assume $rank(n) = \log n$ for sets A and B both with n elements and $rank(A) = rank(B)$ and C is their union then $rank(C) = rank(A) + 1 = rank(B) + 1$

$$rank(A) \leq \log n + 1$$

$$rank(C) \leq rank(A) + 1$$

Sub-ing in the max possible value for rank(A) $rank(A) = 1 + \log n$

$$rank(C) \leq \log n + 2$$

$$rank(C) \leq \log n + \log(2) + \log(2)$$

$$rank(C) \leq \log(n) + \log(2) + 1$$
$$rank(C) \leq \log(2n) + 1$$
$$rank(C) \leq \lfloor \log(2n) \rfloor \leq \log(2n) + 1$$

Therefore since for C to have rank(A) + 1 it must store 2n elements. It can be seen that the max possible rank of any node is $\lfloor \log n \rfloor$

## Part B

For question 10 we are dealing with a 71 by 71 image, which results in 5041 nodes. The worst case rank is rank(Max) = $\lfloor \log(5041) \rfloor$. 1 byte is 8-bits, which can represent numbers in the range $0 \leq x \leq 2^b - 1$ where b = number of bits Then this is asking is $2^b - 1 larger than \lfloor \log(5041) \rfloor$

$$2^b - 1 \geq log(n) + 1$$
$$2^b - 2 \geq log(n)$$
$$2^{2^b - 2} \geq n$$
$$2^{126} \geq n$$

Which is true since $2^{126}$ is significantly larger than 5041.