CSCI203 – ALGORITHMS AND DATA STRUCTURES

Tutorial – Data Structure

Sionggo Japit
sjapit@uow.edu.au

18 July 2019

Introduction

What are the objectives of studying data structure?

- To identify and create useful mathematical entities and operations to determine what classes of problems can be solved using these entities and operations.
- To determine the representation of these abstract entities and to implement the abstract operations on these concrete representation.

Introduction

What data structure would you mostly likely see in a non-recursive implementation of a recursive algorithm?

Stack

In a linked list with n nodes, the time taken to insert an element after an element pointed by some pointer is … O(1).

Expression Tree and Stack

Postfix notation is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if “(exp1) op (exp2)” is a normal fully parenthesized expression whose operation is op, then the postfix version of this is “pexp1 pexp2 op”, where pexp1 is the postfix version of exp1 and pexp2 is the postfix version of exp2. The postfix version of a single number or variable is just that number or variable. So, for example, the postfix version of the fully parenthesized expression

(((5 + 2) * (8 - 3)) / 7)

is

52+83-*7/

Expression Tree and Stack

Describe an algorithm

(a) for converting a fully parenthesized expression into its corresponding postfix notation.

(b) evaluating an expression in postfix notation.

5 minutes to discuss.

Expression Tree and Stack

- In a fully parenthesized expression, to each operator op a pair of parenthesis is associated. For example, in expression, (((5 + 2) * (8 - 3)) / 7), the outermost pair of pretenses is associated to the division operator – divides the result of ((5 + 2) * (8 - 3)) by 7. In turns, in ((5 + 2) * (8 - 3)), the outermost parentheses is associated with the multiplication operator of two terms (5 + 2) and (8 - 3), and so forth.

Expression Tree and Stack

- The algorithm for converting a fully parenthesized expression into its corresponding postfix notation, simply works as follows: For each pair of parentheses, move the associated operator to the position of the right parenthesis, and remove that pair of parentheses.

Expression Tree and Stack

- The steps of converting (((5 + 2) * (8 - 3)) / 7) into its corresponding postfix notation is shown below:
- (((5 + 2) * (8 - 3)) / 7) ((5+2)*(8-3)) 7/ (5+2) (8-3)* 7/ 5 2+ 8 3-* 7/
- which gives 5 2 + 8 3 - * 7 / as the postfix notation.

Expression Tree and Stack

b) Evaluating a postfix notation uses a stack and works as follows;

Expression Tree and Stack

- Scan the expression from left to right.
- If the element is an operand, push it into the stack.
- If the element is an operator, o1, pop two operands, p1 and p2; compute c = p2 o1 p1 and push c into the stack.
- If you reach to the end of expression, pop the stack and get the result

Expression Tree and Stack

The following shows the steps for evaluating 5 2 + 8 3 - * 7 /.

```
processed element  Content of the Stack --------------------------
-           ----------------------------
                                               |  5  |
 5                                             |     |
                                               +-----+
```

Expression Tree and Stack

```
                              |  2   |
 2                            |  5   |
                              +-----+
+                             |  7   |  5 + 2 = 7 is pushed
                              +-----+
                              |  8   |
8                             |  7   |
                              +-----+
```

Expression Tree and Stack

```
                        |  3  |
                        |  8  |
3                       |  7  |
                        +-----+
                        |  5  |  8 – 3 = 5 is pushed
-                       |  7  |
                        +-----+
```

Expression Tree and Stack

```
                      | 35  |  7 * 5 = 35 is pushed
                      |     |
                      +-----+
                      |  7  |
 7                    | 35  |
                      +-----+
                      |  5  |  35 / 7 = 5 is pushed
/                     +-----+
```

Expression Tree and Stack

As we reach to the end of expression, the content of the stack determines the result.

Stack and Queue

Let Q be a non-empty queue and let S be an empty stack. Using only the stack and queue ADT functions and a single element variable X, write an algorithm to reverse the order of the elements in Q.

5 minutes to discuss.

Stack and Queue

```
Algorithm reverse(Queue Q,
                    Stack S) {
    Let X be an element
    While queue Q is not empty {
        Dequeue an element to X.
        Push the element X to the
            stack S.
    }
    While stack S is not empty {
        Pop an element to X.
        Enqueue the element X to
    the
            queue Q.
    }
}
```

```
void reverse(Queue &Q,
                Stack &S) {
    ELEM X;
    while (!Q.isEmpty()) {
        X = Q.dequeue();
        S.push(X);
    }
    while (!S.isEmpty()) {
        X = S.pop();
        Q.enqueue(X);
    }
}
```

What is the running time complexity of this algorithm? $2n = O(n)$

Array

Suppose we are maintaining a collection C of elements such that, each time we add a new element to the collection, we copy the contents of C into a new array list of just the right size. What is the running time of adding n elements to an initially empty collection C in this case? Justify your answer.

5 minutes to discuss.

Array

- Adding the first element to the collection implies to copy one element to the array (i.e. one operation). Insertion of the second element to the collection, however, implies to copy two elements to the array (i.e. two operations). With similar argument, insertion of the i^{th} element implies to copy i elements to the array (i.e. i operations). So, the total number of operations after adding the nth element to the collection, will be

$$1+2+\ldots+n=\frac{n(n+1)}{2}$$

- Thus the running time of the algorithm is, $O(n^2)$.

Heap

What is a max-heap?

A max-heap is a specialized tree such that

- Each element (item) must be **>=** all of its descendants.
- All levels are **full**, except possibly the last level.

  If the last level (bottom level) is not full, all of its nodes must be as far left as possible.

What is a min-heap?

A max-heap is a specialized tree such that

- Each element (item) must be **<=** all of its descendants.
- All levels are **full**, except possibly the last level.

  If the last level (bottom level) is not full, all of its nodes must be as far left as possible.

Heap

Is there a heap T storing seven entries with distinct keys such that a preorder traversal of T yields the entries in increasing or decreasing order by key? How about an inorder traversal? How about a postorder traversal? If so, give an example; if not, say why.

5 minutes to discuss.

Heap

- A heap with 7 entries is a complete binary tree of height 2, which has a root (at depth 0), two internal nodes (at depth 1), and four external nodes (at depth 2) –see Figure 1.

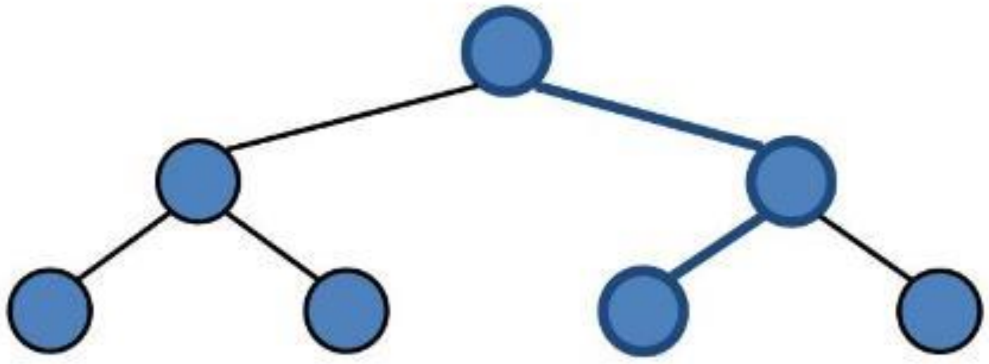


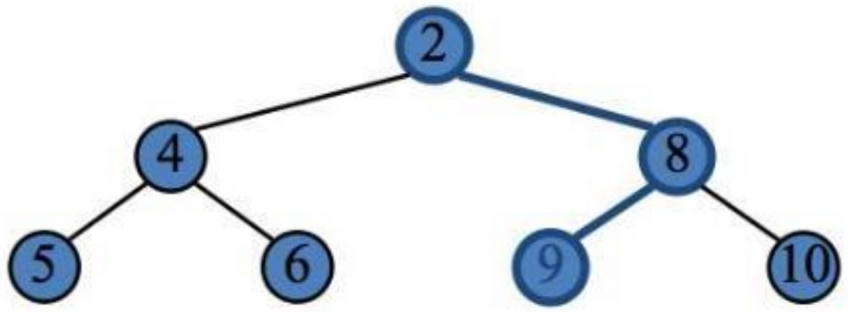
Figure 1: A heap, T, with 7 entries

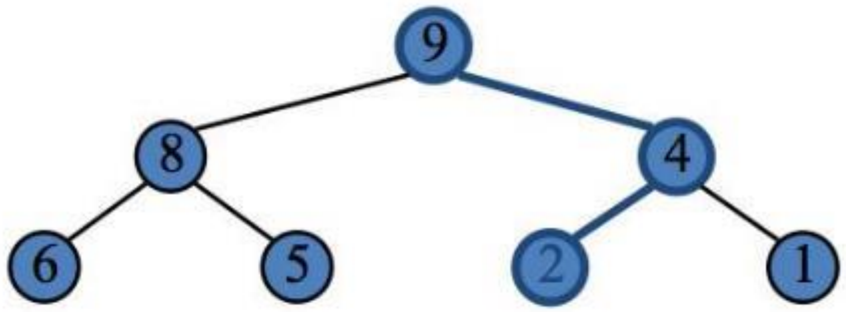
Heap

- In an inorder traversal, we visit the left subtree before visiting the root and then we visit the right subtree. If T is a min-heap then the key value of the root is smaller than the key values of its children, and if T is a max-heap then the key value of the root is larger than the key values of its children. In both cases, it is impossible that the inorder traversal of T yields the entries in increasing or decreasing order by key values.

Heap

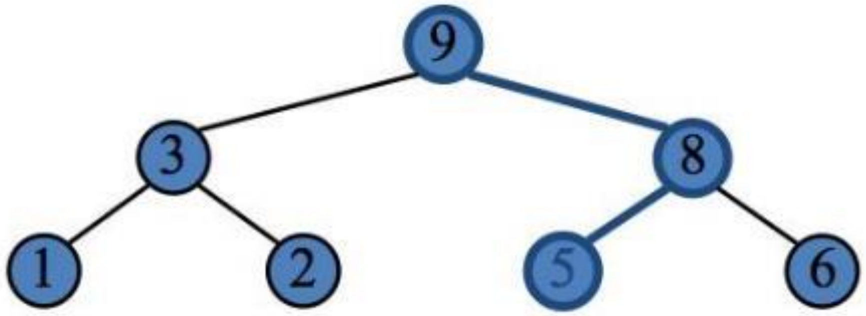
- In a preorder traversal, we visit the root, the left subtree, an then the right subtree in order. So, it is possible to have an instance of the heap T, in which its preorder traversal yields the entries in increasing (or decreasing) order by keys. See, e.g. the heaps (a min-heap and a max-heap) of Figure

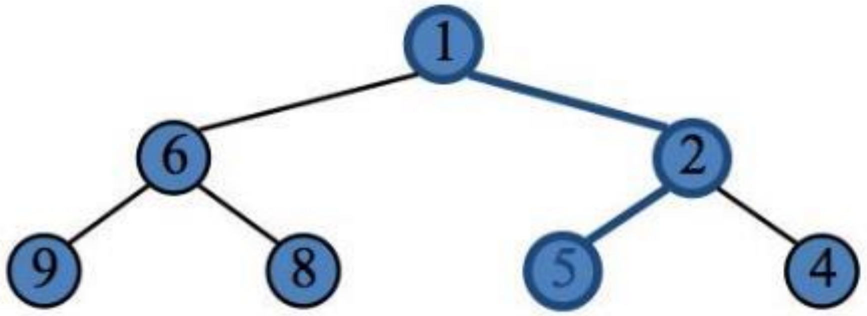




Heap

- In a postorder traversal of T , we visit the left and right subtrees before visiting the root. So it is possible to have instances of T in which their postorder traversal yield entries in increasing or decreasing order by keys. Such heaps are shown in Figure 3.





Heap

- In a binary heap, for an item in position i, where are the parent, left child, and right child located?

The parent is $\left\lfloor \frac{n}{2} \right\rfloor$

$Note: The\ symbol \lfloor\quad\rfloor denotes\ floor\ or\ truncated\ or\ round - down.$

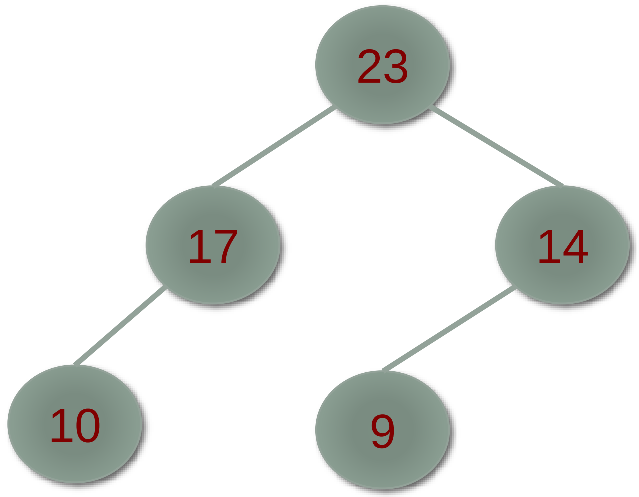
The left-child is 2i, and

The right-child is 2i+1.

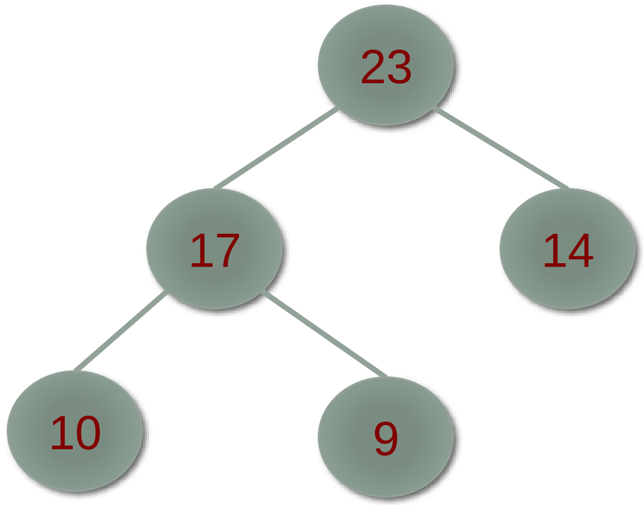
(Note: This is true if the root node starts as node 0)

Heap

- Is the following a heap? Explain or justify your answer.



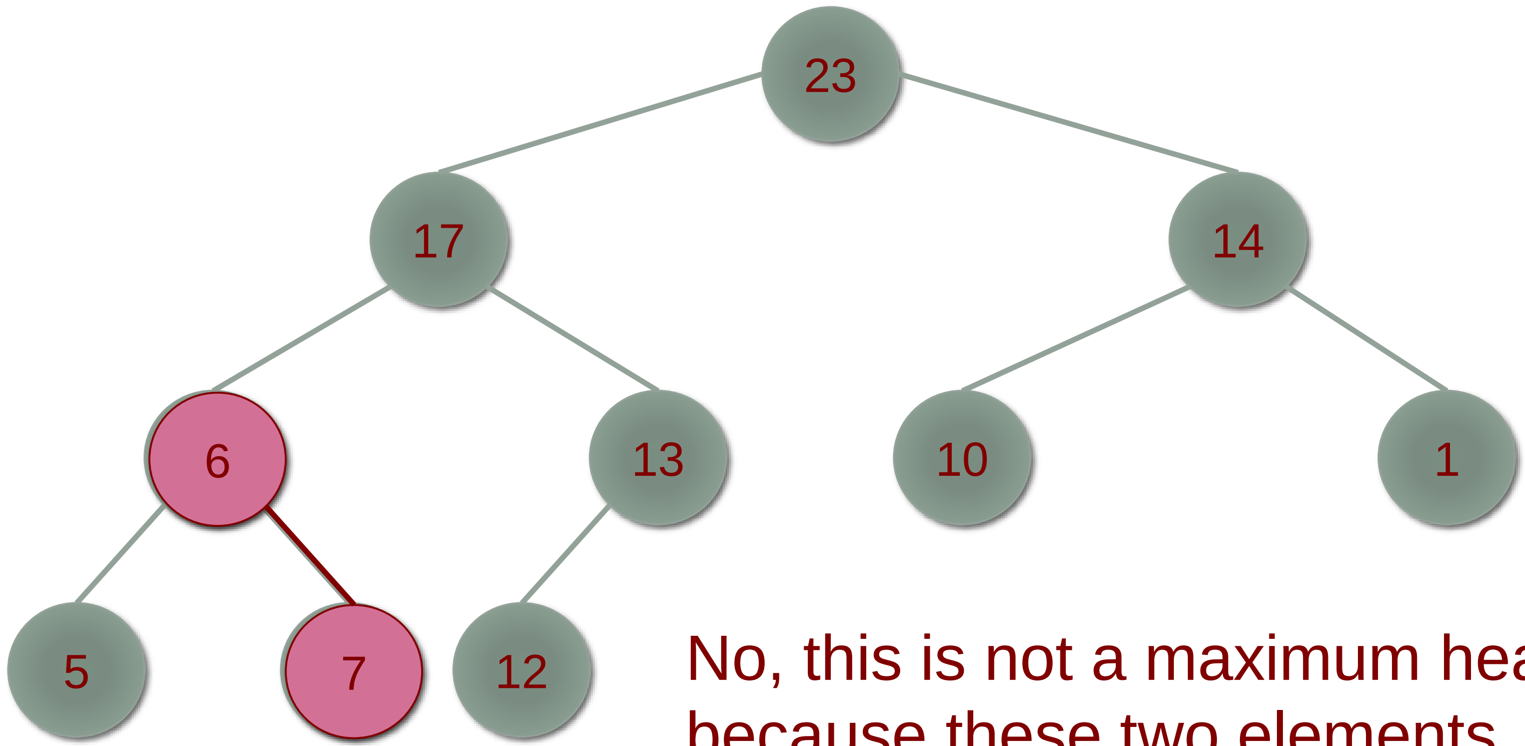
No, this is NOT a heap, because the binary tree is NOT a complete or nearly-complete binary tree.



Yes, this is a heap, because both the structure and value constraints are met; that is, binary tree is a nearly-complete binary tree and each node value is higher or equal to all its descendants.

Heap

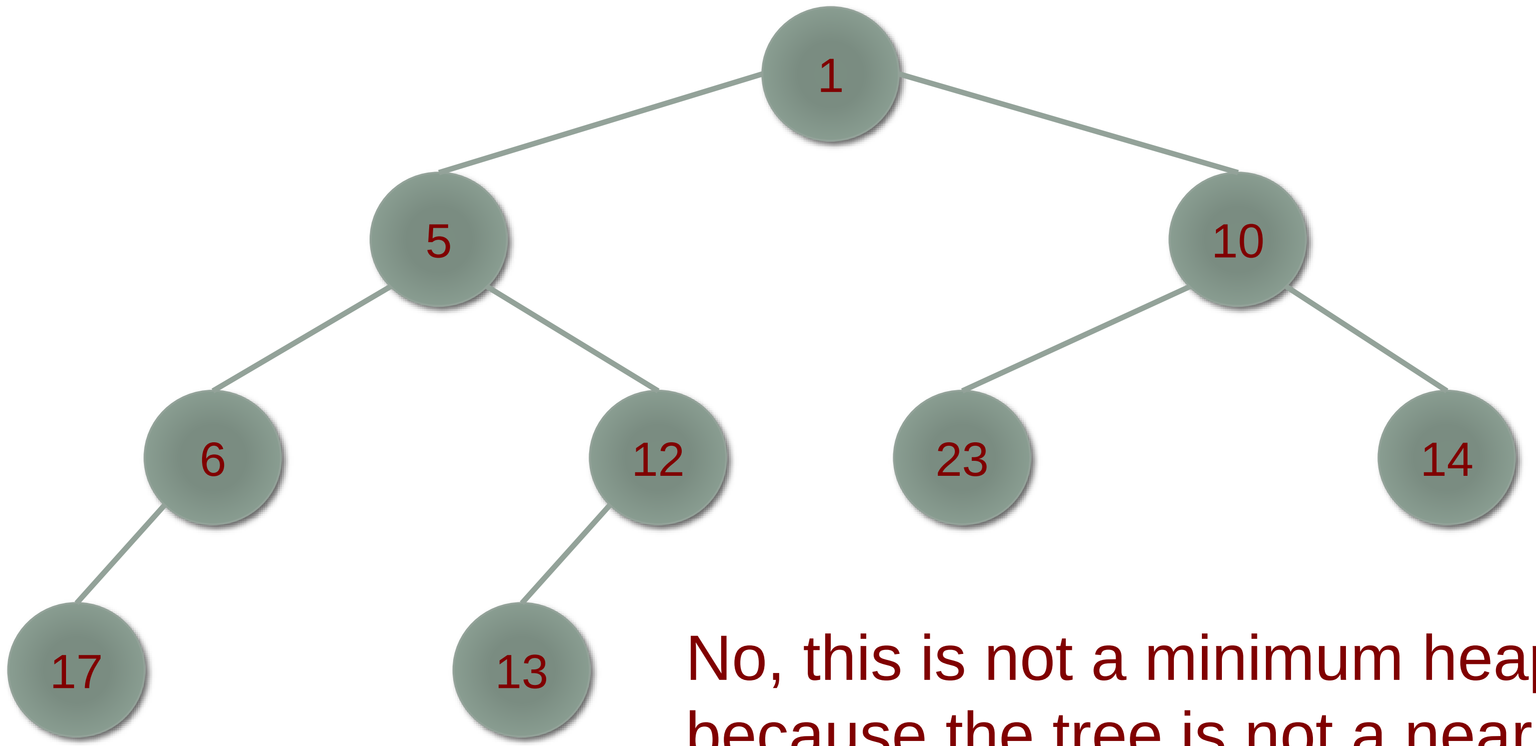
- Is the following a maximum heap?



No, this is not a maximum heap, because these two elements violate the property of a maximum heap.

Heap

- Is the following a minimum heap?



No, this is not a minimum heap, because the tree is not a nearly completed binary tree.

Heap

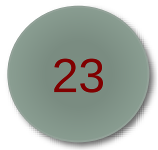
- Construct a minimum-heap from an empty heap using the following sequence of numbers: 23, 17, 14, 6, 13, 10, 1. (Note: the numbers are available one at a time in the specified sequence.)

5 minutes to discuss.

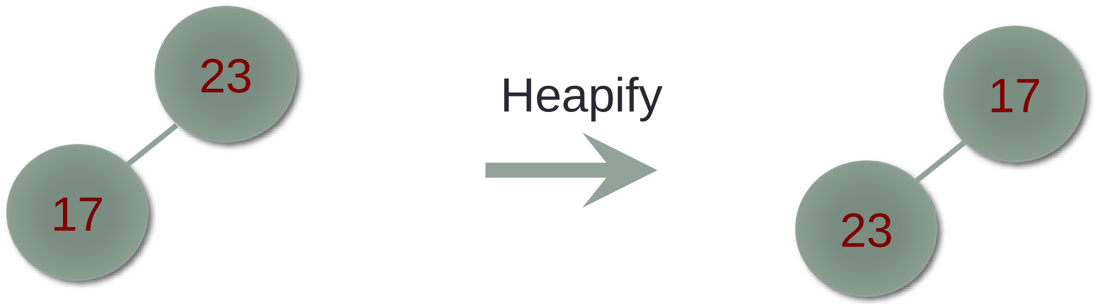
Heap

- Construct a minimum-heap from an empty heap using the following sequence of numbers: 23, 17, 14, 6, 13, 10, 1. (Note: the numbers are available one at a time in the specified sequence.)

Insert 23:



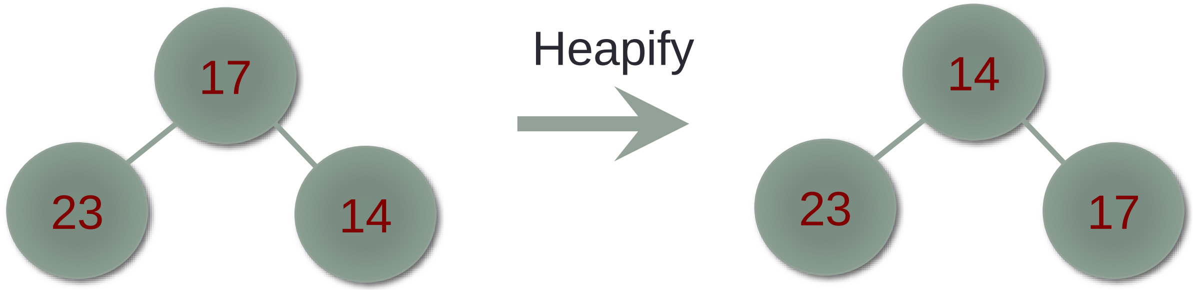
Insert 17:



Heap

23, 17, 14, 6, 13, 10, 1

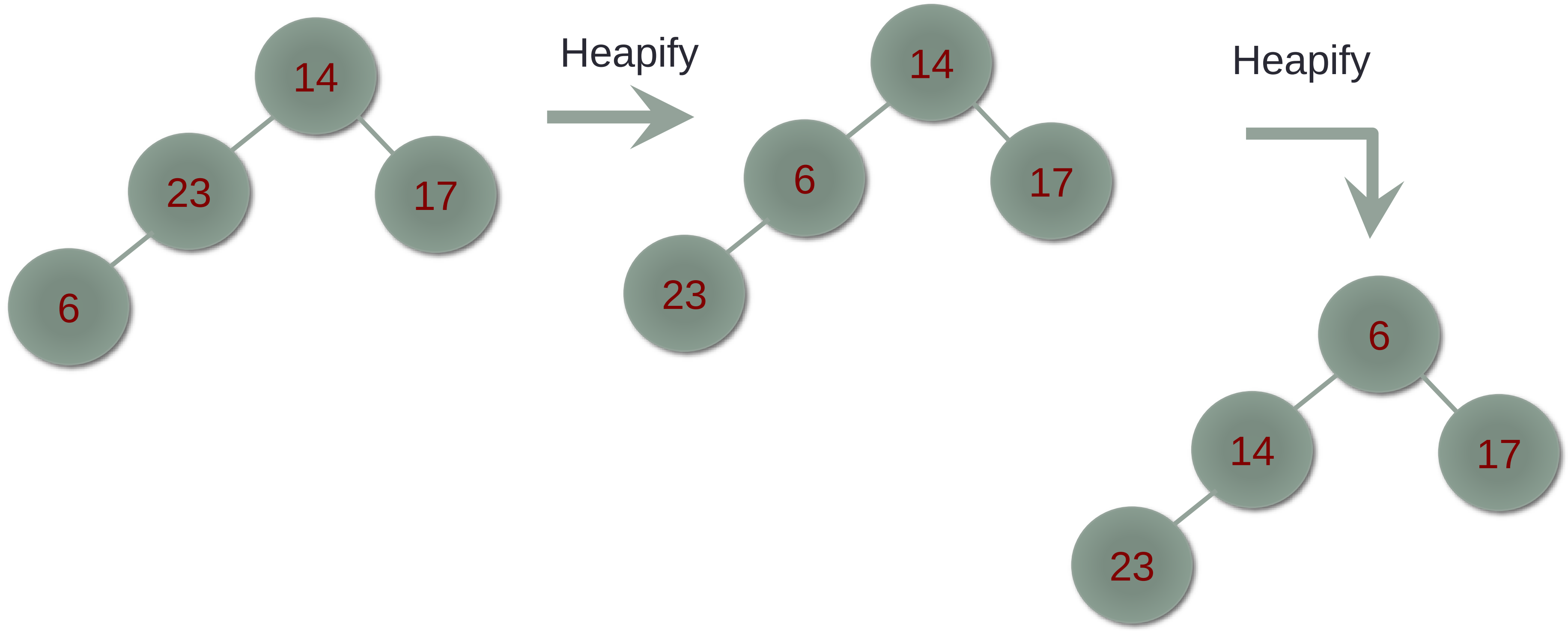
Insert 14:



Heap

23, 17, 14, 6, 13, 10, 1

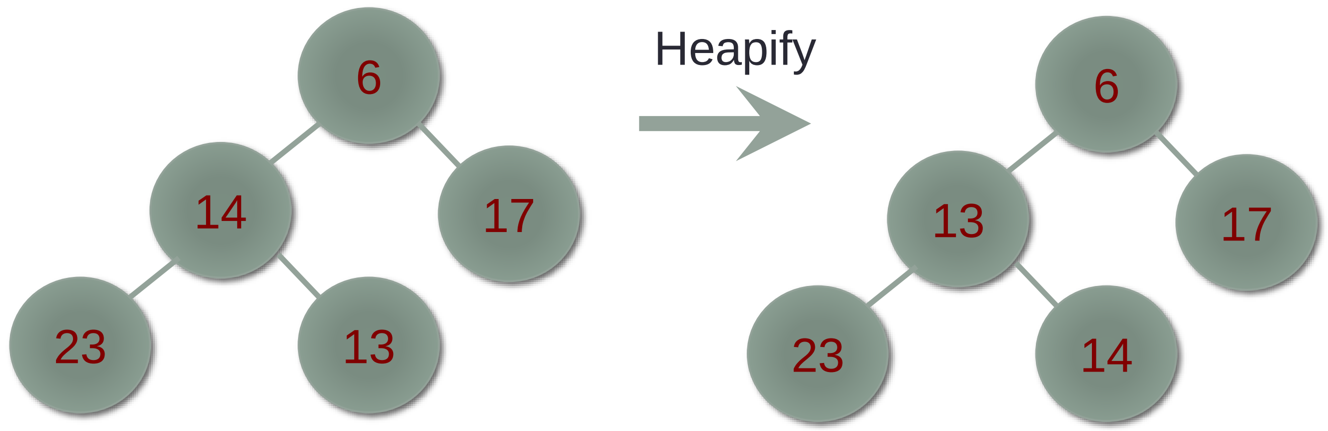
Insert 6:



Heap

23, 17, 14, 6, 13, 10, 1

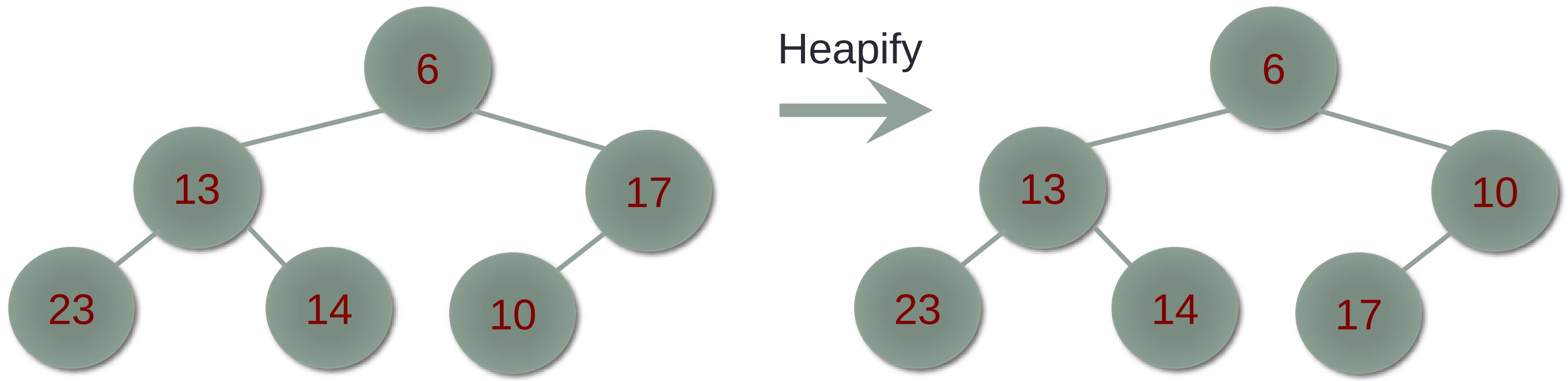
Insert 13:



Heap

23, 17, 14, 6, 13, 10, 1

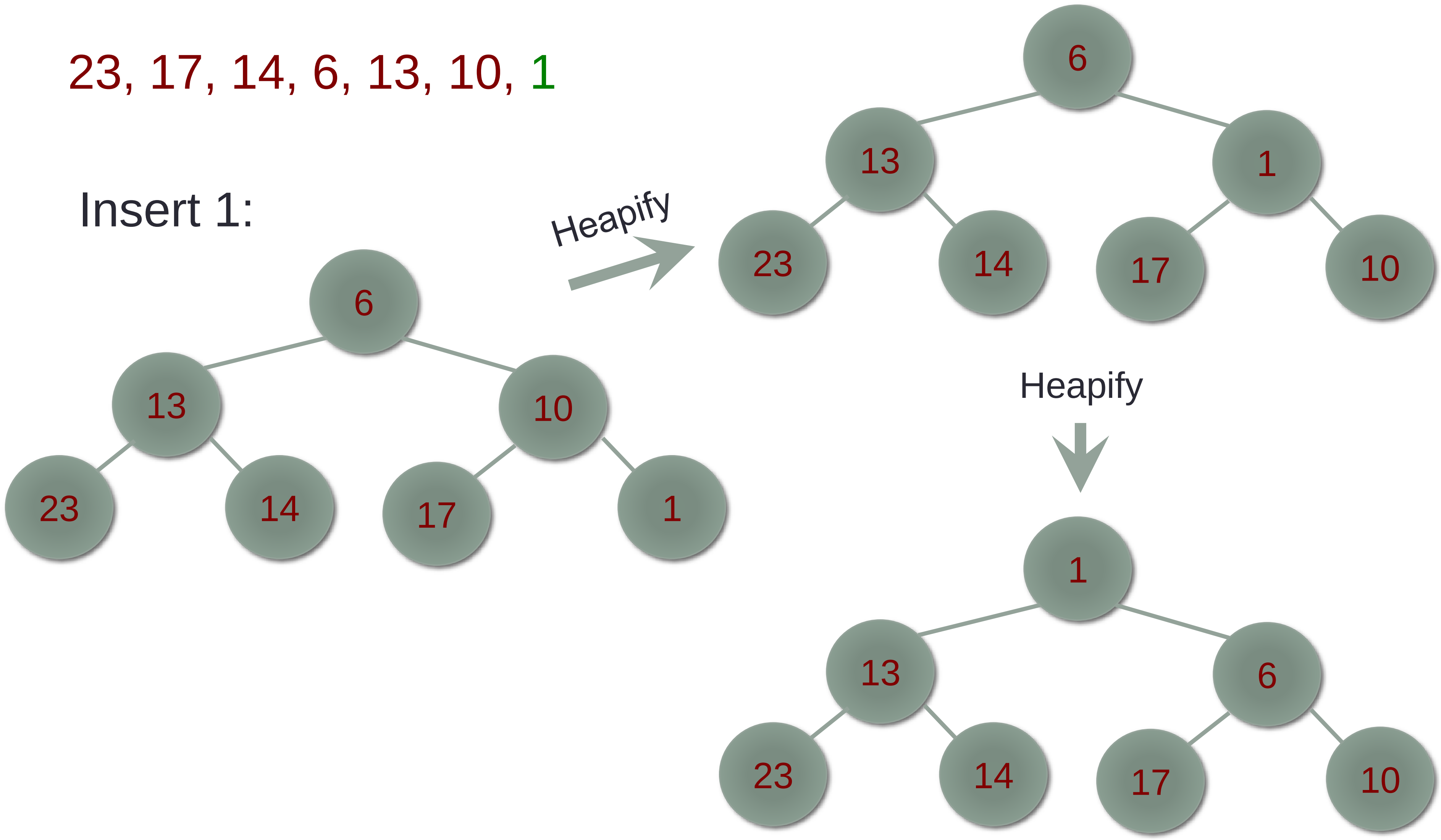
Insert 10:



Heap

23, 17, 14, 6, 13, 10, 1

Insert 1:



Makeheap

Makeheap

```
Procedure makeheap(T[1..n])
  for i = n ÷ 2 to 1 step –1 do
      siftdown(T, i)

Procedure siftdown(T[1 .. n], i)
  k = i
  repeat
      j = k
      if 2j <= n and T[2j] > T[k] then k = 2j
      if 2j + 1 <= n and T[2j + 1] > T[k] then k = 2j + 1
      swap T[j] and T[k]
  until j = k
```

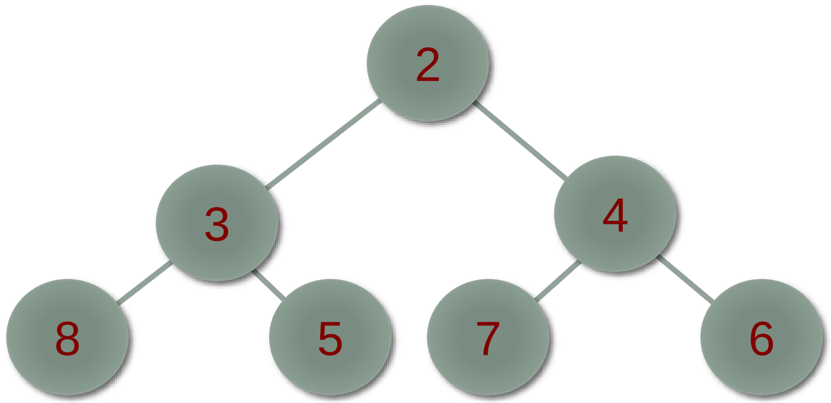
Heap

- Next, show how if the minimum heap is implemented using array from the same sequence of numbers; that is, 23, 17, 14, 6, 13, 10, 1.

5 minutes to discuss.

Heap

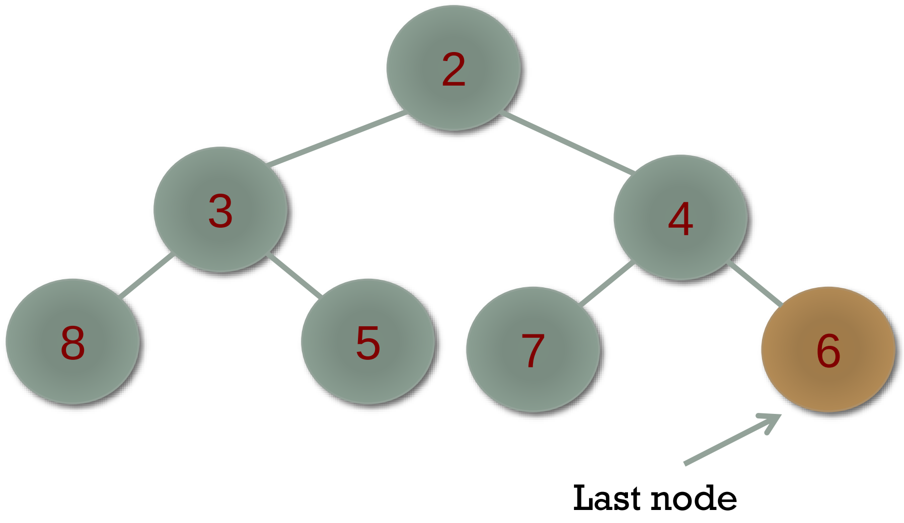
Given the following heap:

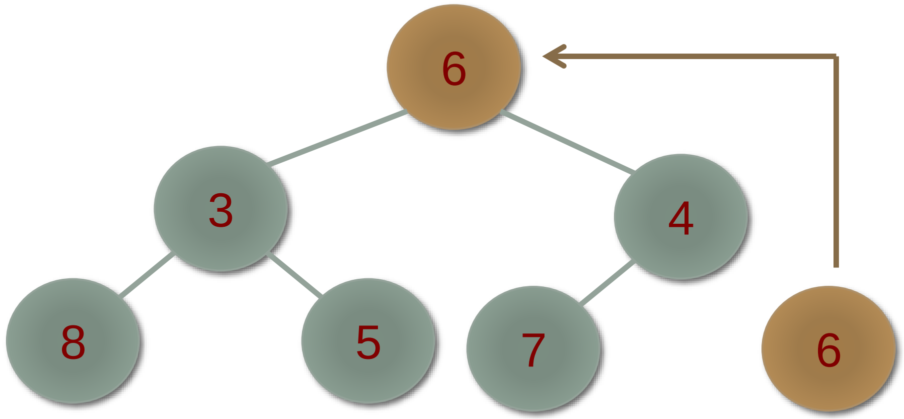


Show all the steps of the algorithm for removing key 2 from the heap.

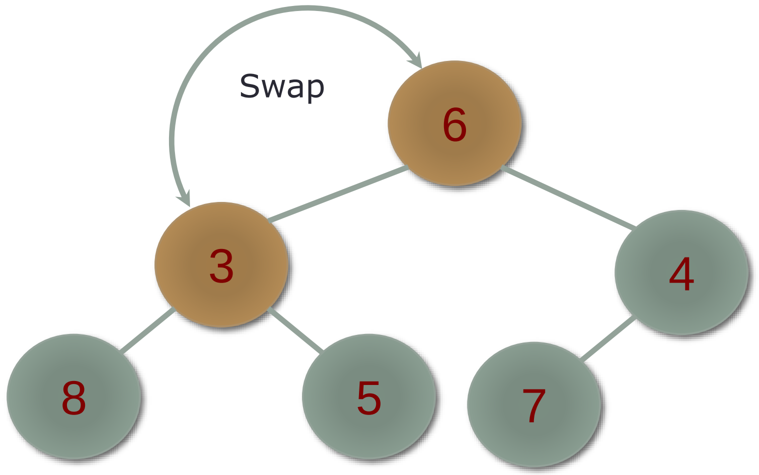
5 minutes to discuss.

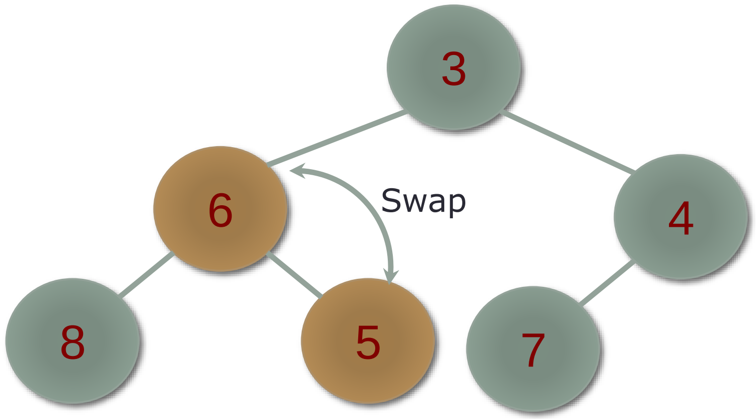
Heap

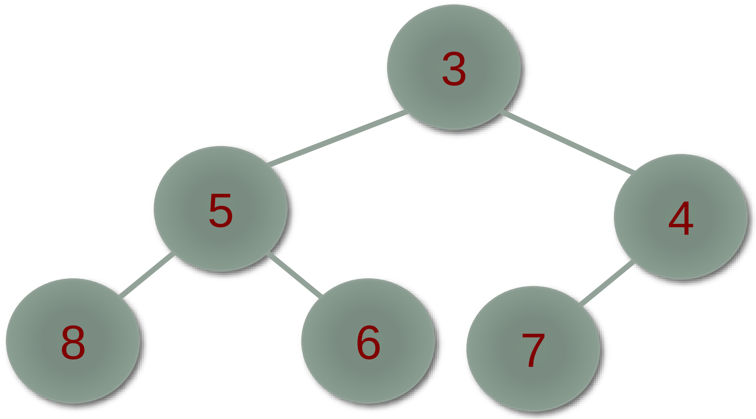




Heap







Associative Table

University of Wollongong

SIM GLOBAL EDUCATION

Associative Table

For the input 30, 20, 56, 75, 31, 19 and hash function *h(K) = K mod 11*:

i. Draw the 6-entry hash table that results from using the above mentioned hash function, assuming collision is handled by linear probing.
ii. Find the largest number of key comparison in a successful search in this table.
iii. Find the average number of key comparisons in a successful search in this table. You can assume that a search for each of the six keys is equally likely.

Associative Table

i. Construction of the hash table.

The list of keys: **30, 20, 56, 75, 31, 19**

The hash function: $h(k) = k \bmod 11$

The hash address:							
	k:	30	20	56	75	31	19
The hash address:	h(k):	8	9	1	9	9	8

Associative Table

The hash table using linear probing resolution:

0	1	2	3	4	5	6	7	8	9	10
31	56	19						30	20	75

ii. The largest number of key comparisons in a successful search in this table is 6, that is, searching for k = 19.

Associative Table

iii. The average number of key comparisons in a successful search in this table, assuming that a search for each of the six keys is equally likely, is

$$\frac{1}{6} \times 1 + \frac{1}{6} \times 1 + \frac{1}{6} \times 1 + \frac{1}{6} \times 2 + \frac{1}{6} \times 3 + \frac{1}{6} \times 6$$

$$= \frac{14}{6} \approx 2.33$$

Associative Table

For the input 30, 20, 56, 75, 31, 19 and hash function *h(K) = K mod 11*:

i. Draw the 6-entry hash table that results from using the above mentioned hash function, assuming collision is handled by chaining.
ii. Find the largest number of key comparison in a successful search in this table.
iii. Find the average number of key comparisons in a successful search in this table. You can assume that a search for each of the six keys is equally likely.

Associative Table

i. Construction of the hash table.

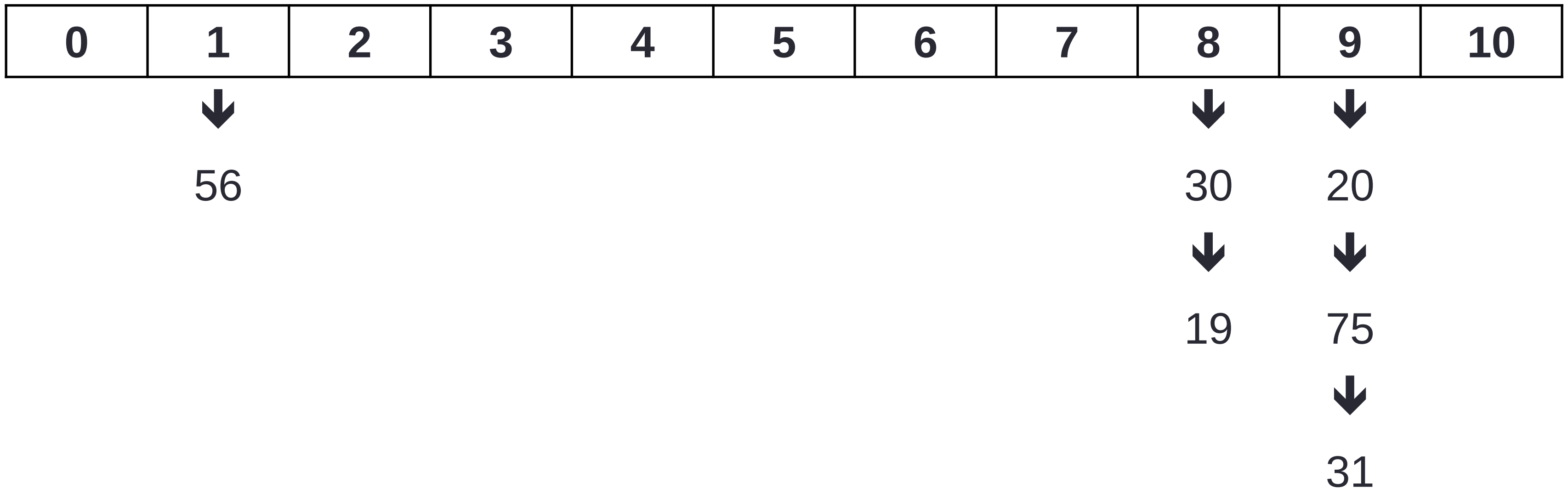
The list of keys: **30, 20, 56, 75, 31, 19**

The hash function: $h(k) = k \bmod 11$

The hash address:							
	k:	30	20	56	75	31	19
The hash address:	h(k):	8	9	1	9	9	8

Associative Table

The hash table using chaining collision resolution:



ii. The largest number of key comparisons in a successful search in this table is 3, that is, searching for k = 31.

Associative Table

iii. The average number of key comparisons in a successful search in this table, assuming that a search for each of the six keys is equally likely, is

$$\frac{1}{6} \times 1 + \frac{1}{6} \times 1 + \frac{1}{6} \times 1 + \frac{1}{6} \times 2 + \frac{1}{6} \times 3 + \frac{1}{6} \times 2$$

$$= \frac{10}{6} \approx 1.7$$

Associative Table

Insert keys 18, 41, 22, 44, 59, 32, 31, 73 in this order to an associative table, using double-hashing resolution technique, where $h(x) = x(\bmod 13)\ and\ h'(x) = 5 - x(\bmod 5).$

The hash addresses:	x	18	41	22	44	59	32	31	73
	H(x)	5	2	9	5	7	6	5	8
	H'(x)				1			4	

Since h(44) = 5, which has been occupied, collision occurs. To resolve the collision, a second hash function is applied, in this case, h'(44) = 5 – 44 mod 5, which give you 1, a new location and this location is not occupied. Hence the key 44 will be inserted to location 1.

Similarly, for key 31, second hash function is used to resolve collision.

Associative Table

- The hash table using double-hashing collision resolution is as follow:

0	1	2	3	4	5	6	7	8	9	10	11	12
	44	41		31	18	32	59	73	22			

Associative Table

- What happen if the second time hashing collide again? How to solve the problem?

$$dh(x) = \left(\underbrace{(x \; mod \; 13)}_{\text{First hash}} + j \times \underbrace{\left((5 - x) \; mod \; 5\right)}_{\text{Second hash}}\right) mod \; 13$$

where $j = 0,1,2,3, \ldots$