Comp 2540: Data Structures and Algorithms: Review

Prof. Jianguo Lu

April 2, 2025

This course is about "good" program and "bad" programs

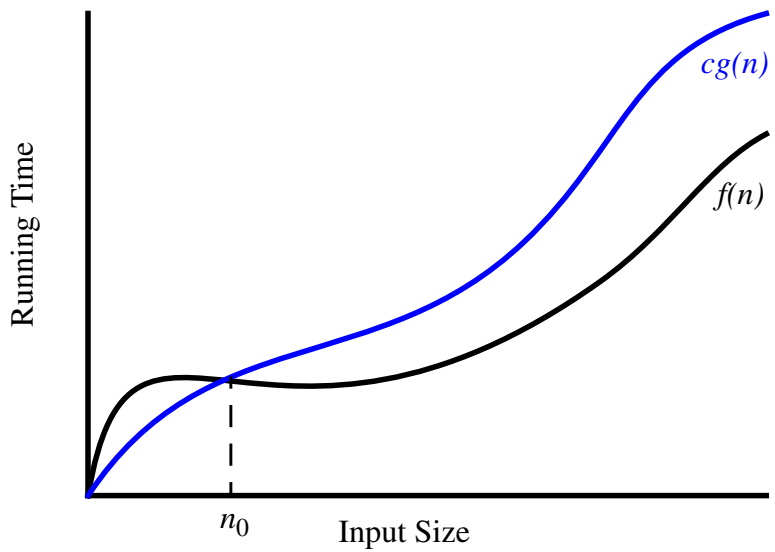
- "Bad" programs are slow programs
- Examples:
  - good vs bad string concatenation
  - good vs bad sorting programs,
  - good vs bad searching programs,
  - good vs bad tree height programs,
  - good vs bad Fibonacci programs,
  - good vs bad shortest path programs.
  - good vs bad word counting programs,
    - A1: $O(nm^2)$, $O(nm)$, (use linked list, m: vocabulary size)
    - A2: $O(n^2)$, $O(n \log n)$ (use sorting)
    - A4: $O(n)$
  - ...
- How do we know whether a program is good?

How do we know whether a program is a good one?

- We can measure the running time
  - Running time is data and machine dependent
  - There are big variations when data are small
- We can analyze the complexity of algorithms theoretically
  - Asymptotic analyses

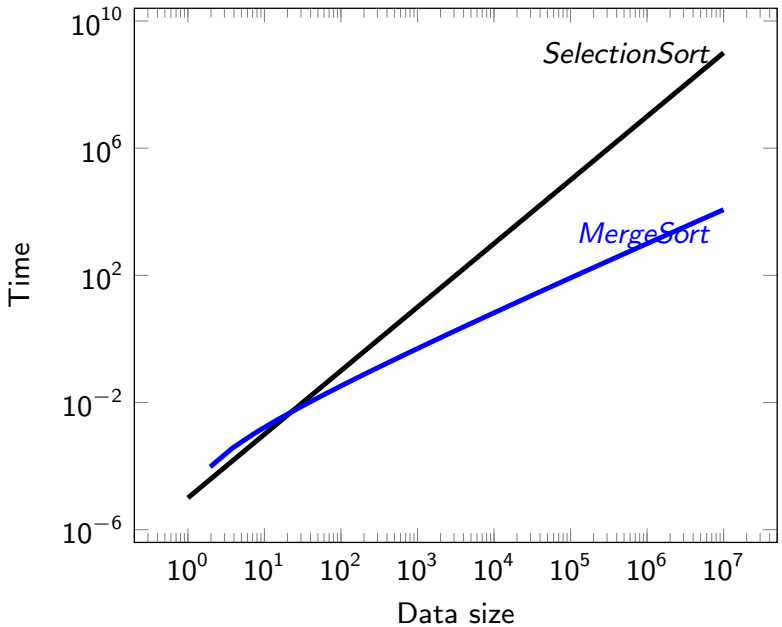
Definition: O(g(n))

$$O((g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\} \quad (1)$$



Selection sort vs. Merge Sort

- It is about the growth rate



How do we design (good) algorithms

Algorithms design strategies

- Divide and Conquer: breaking a problem into smaller ones.
  - e.g., Merge sort and Quick sort.
- Greedy algorithm: often used in optimization problems,
  - e.g., Dijkstra's shortest path algorithm
- Dynamic programming:
  - e.g., Fibonacci
  - Dijkstra's algorithm also uses memorization.

```
public static Entry<String,Integer> count_LINKED_LIST_BAD(String[]
    tokens) {
  LinkedList<Entry<String, Integer>> list = new
      LinkedList<Entry<String, Integer>>();
  for (int j = 0; j < tokens.length; j++) {
        String word = tokens[j];
        boolean found = false;
        for (int i = 0; i < list.size(); i++) {
                Entry<String, Integer> e = list.get(i);
                if (word.equals(e.getKey())) {
                        e.setValue(e.getValue() + 1);
                        list.set(i, e);
                        found = true;
                        break;
                }
        }
        if (!found)
            list.add(new AbstractMap.SimpleEntry<String,
                Integer>(word, 1));
}
```

- Time complexity depends not only on the layers of nested loops
- We also need to know the time complexity of the operations

A2

- String concatenation, read text,
- Sorting algorithms
- Overall better programs (reading and counting)

A3

- Stack and dynamic array
- Time complexity for amortized algorithms, doubling strategy
- Time and space efficiency

A4: Heap and Hashing

- How to implement Heap
- How to implement Hash
- How to use Hash to count word frequency

A4: Heap and Hashing

- How to implement Heap
- How to implement Hash
- How to use Hash to count word frequency
- API provides those implementations, Why do I need to learn the implementation?

A5: Dijkstra's algorithm

Algorithm 1: Dijkstra's algorithm

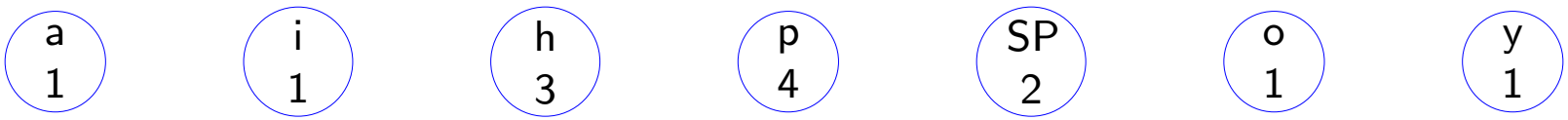
Input: Graph G=(V, E), starting node *s*
Output: Distance *dist* to all remaining nodes

```
1  dist[s] =0;
2  dist[v]=∞ for all other nodes;
3  PriorityQueue Q=V with their distances;
4  while Q not empty do
5      u =Q.removeMin();
6      for all v ∈ neighbors[u] and v ∈ Q do
7          if dist[v] > dist[u] + w(u, v) then
8              dist[v]=dist[u] + w(u, v);
9              update Q with new distance
10 return dist
```

- We need to have fast removeMin() operation and fast update operation.
- Heap is good for removeMin
- Hash is good for update
- We need to modify heap for our purpose

Bonus Assignment

Given the letters and their frequencies as below. Create the corresponding Huffman tree.



Overview of topics

Week 1	Introduction, Array and Linked List
Week 2, 3	Algorithm Analysis.
Week 3	Divide and conquer, merge sort.
Week 4	Recursion and algorithm analysis of recursive programs
Week 5, 6	Stacks and Queues, Priority queues and Heaps, Heap sort
Week 7	Trees, Midterm Exam.
Week 8	Maps and Hash tables
Week 9	More on Sorting algorithms
Week 10	Graph algorithms, graph traversal, shortest path.
Week 11	Binary search trees, AVL trees, Red-black trees
Week 12	Red-black tree, Huffman tree, Review

Abstract Data Types (ADT)

Data structures are often represented by Abstract Data Types

Definition of ADT

- Define a data type using its operations and behavior
- Ignore (abstract away) the details of the implementation
- Examples: Stack, Queue, ...

Stack ADT

boolean empty()	Tests if this stack is empty.
E peek()	Looks at the object at the top of this stack without removing it.
E pop()	Removes the object at the top of this stack and returns it.
E push(E item)	Pushes an item onto the top of this stack.

Practice Questions

The height of a red-black tree that contains n nodes is:

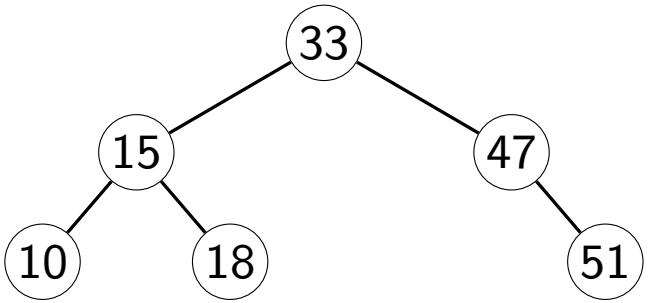
1. At most twice the height of its equivalent (2,4)-tree.
2. n log n
3. Asymptotically the same as the height of a binary search tree
4. Twice the height of an AVL tree.

Practice Questions

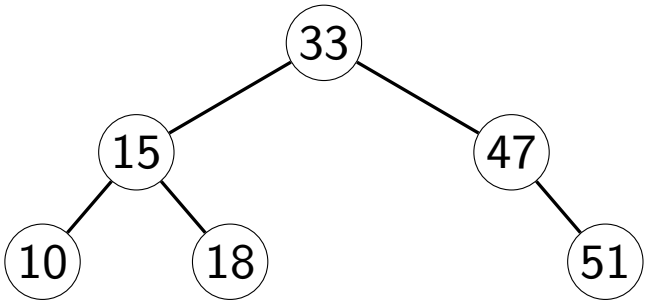
The height of a red-black tree that contains n nodes is:

1. At most twice the height of its equivalent (2,4)-tree.
2. n log n
3. Asymptotically the same as the height of a binary search tree
4. Twice the height of an AVL tree.

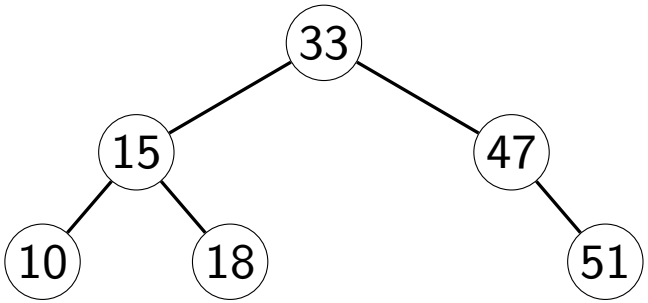
Answer: 1



- Write the pre-order traversal of the tree.



- Write the pre-order traversal of the tree.
- *Answer: 33, 15, 10, 18 47, 51*
- Write the in-order traversal of the same tree.



- Write the pre-order traversal of the tree.
- *Answer: 33, 15, 10, 18 47, 51*
- Write the in-order traversal of the same tree.
- *Answer: 10, 15, 18, 33, 47, 51*

What is the major benefit of QuickSort over MergeSort when both are implemented efficiently?

What is the major benefit of QuickSort over MergeSort when both are implemented efficiently?

Answer: Quick sort can be implemented as an in-place sorting, hence more space-efficient compared with Merge-sort

The height of a tree is the maximum depth of all the nodes in a tree. A bad algorithm to compute the height is listed below:

```
private int heightBad( ) {
        int h = 0;
        for (Node node : nodes( ))
                if (isExternal(node))
                        h = Math.max(h, depth(node));
        return h;
}
```

- Write a more efficient algorithm in pseudocode that has the time complexity $O(n)$, where n is the number of nodes in the tree.

The height of a tree is the maximum depth of all the nodes in a tree. A bad algorithm to compute the height is listed below:

```
private int heightBad( ) {
        int h = 0;
        for (Node node : nodes( ))
                if (isExternal(node))
                        h = Math.max(h, depth(node));
        return h;
}
```

- Write a more efficient algorithm in pseudocode that has the time complexity $O(n)$, where n is the number of nodes in the tree.

Answer:

```
public int height(Node node) {
  int h = 0;
  for (Node c: children(node))
      h = Math.max(h, 1 + height(c));
  return h;
}
```

Given a Hash table of size N = 9 as follows:

		11		31			61	80
0	1	2	3	4	5	6	7	8

where the hash function is $h(k) = k \mod N$. Suppose that linear probing is used to handle collisions.

(a) If we insert 5 now, write in the following table where it should be.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Given a Hash table of size N = 9 as follows:

		11		31			61	80
0	1	2	3	4	5	6	7	8

where the hash function is $h(k) = k \mod N$. Suppose that linear probing is used to handle collisions.

(a) If we insert 5 now, write in the following table where it should be.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Answer:

		11		*31*	*5*		*61*	*80*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

(b) After inserting 5, if we want to insert 4, write the result in the following table.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Given a Hash table of size N = 9 as follows:

		11		31			61	80
0	1	2	3	4	5	6	7	8

where the hash function is $h(k) = k \mod N$. Suppose that linear probing is used to handle collisions.

(a) If we insert 5 now, write in the following table where it should be.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Answer:

		11		*31*	*5*		*61*	*80*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

(b) After inserting 5, if we want to insert 4, write the result in the following table.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Answer:

		11		*31*	*5*	*4*	*61*	*80*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

(c) After a) and b), we want to insert 16. Write the result.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Given a Hash table of size N = 9 as follows:

		11		31			61	80
0	1	2	3	4	5	6	7	8

where the hash function is $h(k) = k \mod N$. Suppose that linear probing is used to handle collisions.

(a) If we insert 5 now, write in the following table where it should be.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Answer:

		11		*31*	*5*		*61*	*80*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

(b) After inserting 5, if we want to insert 4, write the result in the following table.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Answer:

		11		*31*	*5*	*4*	*61*	*80*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

(c) After a) and b), we want to insert 16. Write the result.

		11		31			61	80
0	1	2	3	4	5	6	7	8

Answer:

16		*11*		*31*	*5*	*4*	*61*	*80*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

Given the following binary search algorithm:

```
boolean binarySearch(int[] data,int target,int low,int high) {
  if (low>high) return false;
  else {
      int mid = (low + high) / 2;
      if (target == data[mid]) return true;
      else if (target < data[mid])
          return binarySearch(data, target, low, mid - 1);
          else
               return binarySearch(data, target, mid + 1, high);
  }
}
```

- Write the recurrence relation for its time complexity

Given the following binary search algorithm:

```
boolean binarySearch(int[] data,int target,int low,int high) {
  if (low>high) return false;
  else {
      int mid = (low + high) / 2;
      if (target == data[mid]) return true;
      else if (target < data[mid])
          return binarySearch(data, target, low, mid - 1);
          else
               return binarySearch(data, target, mid + 1, high);
  }
}
```

- Write the recurrence relation for its time complexity
  Answer:

$$T(n) = \begin{cases} \Theta(1), & \text{if } n=1 \\ T(n/2) + \Theta(1), & \text{if } n > 1 \end{cases} \quad (2)$$

Solve the equation using the *substitution method* discussed in class.

Solve the equation using the *substitution method* discussed in class.

Answer: We guess that $T(n)$ is $O(\log n)$. Need to prove it using induction.

$$T(n) \leq c \log n \tag{3}$$

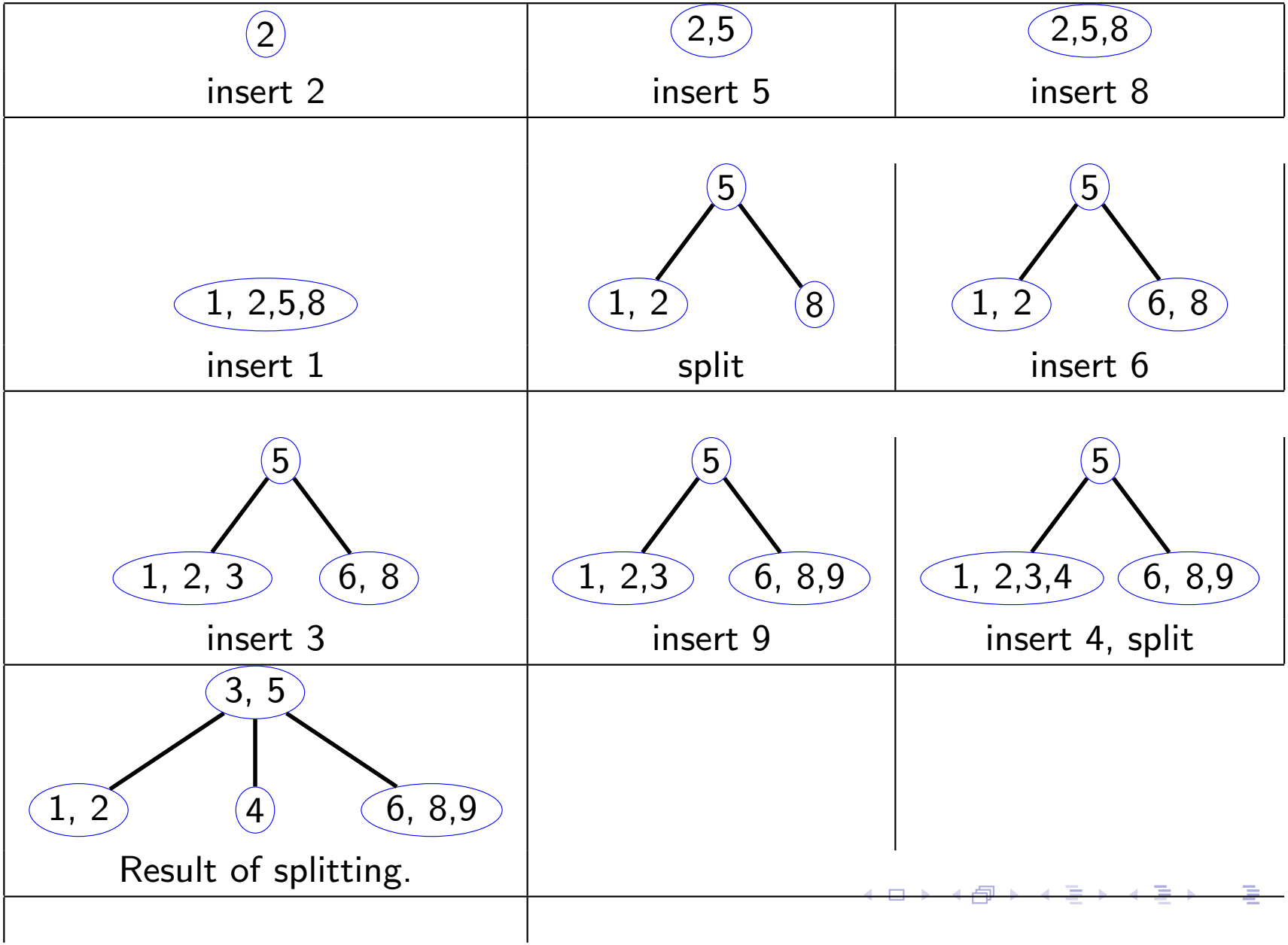
$$T(n/2) \leq c \log n/2 \tag{4}$$

$$\begin{aligned} T(n) &= T(n/2) + 1 && (5)\\ &\leq c \log n/2 + 1 && (6)\\ &= c \log n - c \log 2 + 1 && (7)\\ &= c \log n - c + 1 && (\log 2 = 1)\\ &\leq c \log n && (\text{when } c \geq 1) \end{aligned}$$

Construct a (2,4) tree by inserting keys (2,5,8,1,6,3,9,4) in that sequence. You need to draw a tree after each insertion. You do not need to write justifications for each operation.

Construct a (2,4) tree by inserting keys (2,5,8,1,6,3,9,4) in that sequence. You need to draw a tree after each insertion. You do not need to write justifications for each operation.

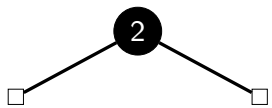
Answer:

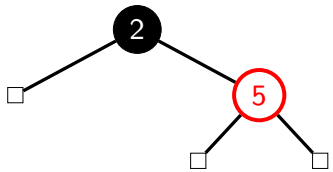


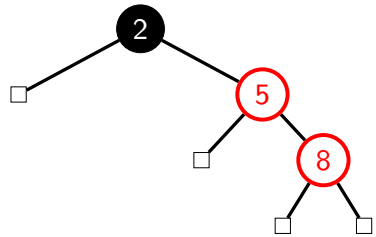
Construct a red-black tree by inserting keys (2,5,8,1,6,3,9) in that sequence. You need to draw a tree after each insertion. You do not need to write justifications for each operation.

Construct a red-black tree by inserting keys (2,5,8,1,6,3,9) in that sequence. You need to draw a tree after each insertion. You do not need to write justifications for each operation.

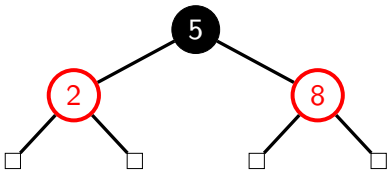
Answer:



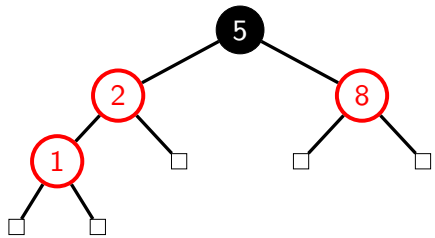




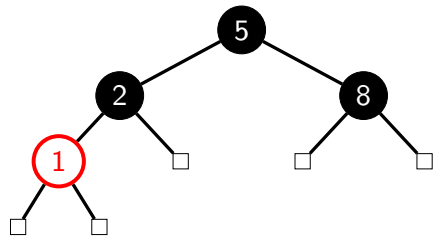
double red. uncle is black. restructure.



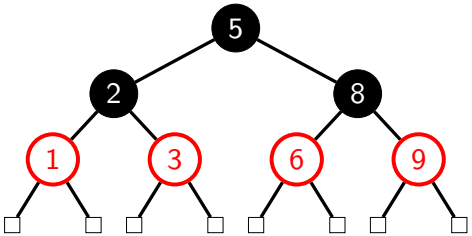
Result of restructuring



Insert 1. double red. uncle is red. recolor.



5 is root. color it black.



Insert 6, 3, 9.

This question is about Heap Construction

- Given a array of elements

A=

	14	21	12	32	96	51	61	28
0	1	2	3	4	5	6	7	8

.

Draw the corresponding heap represented by this array without heapification. Suppose that the top element of the heap starts from A[1].

This question is about Heap Construction

- Given a array of elements

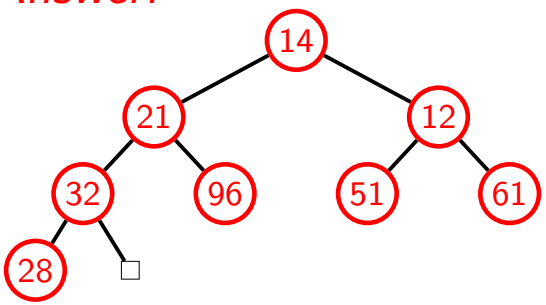
A=

	14	21	12	32	96	51	61	28
0	1	2	3	4	5	6	7	8

.

Draw the corresponding heap represented by this array without heapification. Suppose that the top element of the heap starts from A[1].

Answer:



- Write the pseudo code for constructing a maxHeap bottom-up (i.e., the better approach discussed in class)
- What is the complexity of the bottom-up heap construction algorithm?

This question is about Heap Construction

- Given a array of elements

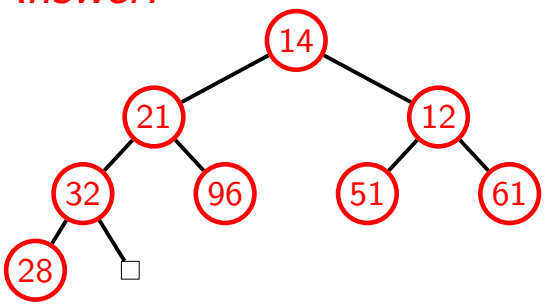
A=

	14	21	12	32	96	51	61	28
0	1	2	3	4	5	6	7	8

.

Draw the corresponding heap represented by this array without heapification. Suppose that the top element of the heap starts from A[1].

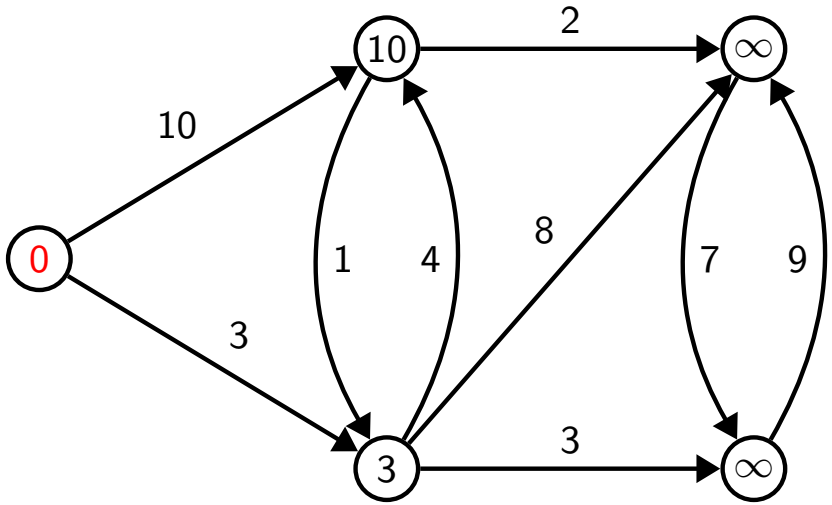
Answer:



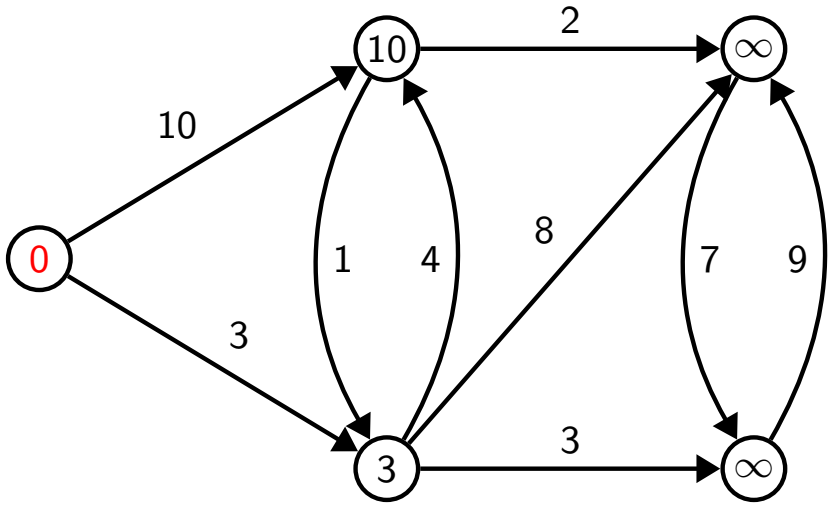
- Write the pseudo code for constructing a maxHeap bottom-up (i.e., the better approach discussed in class)
- What is the complexity of the bottom-up heap construction algorithm?

Answer: refer the slides and book

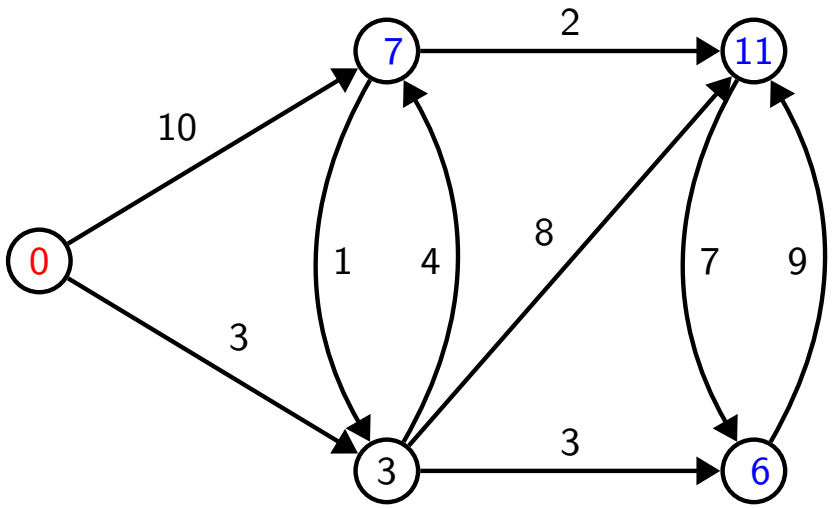
The Dijkstra's algorithm allows edges can be weighted. When we run the Dijkstra's algorithm on the following graph. Suppose that the current node is the one with distance 3. Write the distances of the neighbouring nodes to be updated in this step.



The Dijkstra's algorithm allows edges can be weighted. When we run the Dijkstra's algorithm on the following graph. Suppose that the current node is the one with distance 3. Write the distances of the neighbouring nodes to be updated in this step.



Answer:




Audi Sport
Audi Sport
Audi Sport
BOSCH
ORIS
Swiss Watches
MICHELIN
MICHELIN
Castrol EDGE
itk
7