# 241 Exam Review

## Algorithms, recursion and algorithmic analysis

What is an algorithm?

Algorithm is a description of a series basic steps which achieve a specified result.

How do we describe it?

Plain English, Pseudocode, computer program implementation.

What (and why) is recursion?

Who is the big-O? What does he mean?

It is a representation of the time efficiency of some algorithm.

Definition: we write f(n) = O(g(n)) if, for some Constant C, f(n) <= Cg(n) for ALL sufficiently large C.

Common efficiency analyses: nested loop, divide and conquer!

## Array, sorting and searching

Using subarrays (particularly in recursive method for array processing)

Searching in unsorted array and in sorted array

Sorting method (selection, insertion, Heap sort)

Selection sort:

Pick the 1st smallest element at the first position

Pick the 2ed smallest element at the second position

…

Pick the Nth element at the Nth position

Insertion sort:

The insertion algorithm sorts an array of values by repetitively inserting a particular value into a subset of the array which is already sorted.

Heap sort (see Heap section)

## Random number generator and their use

Why are random numbers important in computation?

What is the difference between truly random and pseudo-random numbers?

Truly random number come from by observing unpredictable physical process. Pseudo-random numbers are generated mathematically, even they appear to be random, but if we know the seed and the algorithm they use, then the random number can be fully predicted.

How are pseudo-random numbers generated?

S <- (s \* a + b) % c

How can we pick a winner? Shuffle a deck? Choose from a collection of know or unknown size? What are some of the efficiency issue?

## ADT principles, data structures

What are the advantages of using ADTs?

ADT is a data type whose intended use is specified by its interface, so this abstraction can hide the detail implementation from user, and user can focus on important issue.

What is the relationship between ADT and data structure?

A data structure is the collection of programming structure used to implement an ADT.

Common ADTs: stack, list, queue – similarities and differences

## Stack and queue ADTs, linked data structures

What is a stack? What is a queue? How are they similar? How are they different?

How are lined data structures implemented in Java?

It is implement with recursive data structure in which, one of the attribute of the class, its type is the class itself.

## Tree:

* A balanced n-ary tree with m elements will have a height of lognm.
* A full balanced binary tree with depth = d, can have at most 2^[d+1] – 1 nodes and at least 2^[d] nodes.

If the binary tree is poor balanced, then it can have at least d + 1 nodes.

* the meaning of full and complete
* Traversal:

1. Preorder: visit root, then traverse the subtree from left to right.
2. Inorder: travers the left subtree, then root, then the remaining subtrees from left to right.
3. Postorder: travers the subtree from left to right, then visit root.

* Binary Search Tree

1. The definition of BST: A binary search tree is a binary tree that, for each node n, the left subtree of n contains element less than the element stored in n, and the right subtree of n contains element that are greater than or equal to the element stored in n.
2. Adding element into binary search tree
3. If root is empty, then root = element.
4. If element is less then root, then add to left subtree.
5. Else add to right subtree.
6. Deleting node from binary search tree
7. If node has no children, just delete it.
8. If node has only one child, replace the deleted node with its child.
9. If the node has two children, an appropriate node is founded from lower in the tree and used it to replace the node, the children of the removed node become the children of replacement node.

A good choice for the appropriate node is the deleted node’s inorder successor (the next highest value).

1. The height () or depth () of Binary search tree.

## Heap and Priority Queue

* Definition of Heap: A heap is a complete binary tree in which every element is greater than or equal to both of its children.
* A heap keeps the largest value of a set of elements readily available. This is useful in any situation in which the value is the dominant criteria.
* Pseudo-code for adding and removal algorithms.
* Using heaps of sorting, compare with other sorting?

Have the advantage of in place sort (insertion sort) and the O(nlogn) time efficiency (merge sorting).

* Heap operation:

1. Adding an element into the heap

The strategy to add an element is to add the element in the heap as a new leaf, keep the tree complete; then moving the element up toward the root.

1. Finding the maximum value in the heap

It is easy, it is just the root.

1. Remove the maximum value

The strategy to delete maximum value, first delete the root, then reconstruction the tree. During the reconstruction, move the “last leaf” of the tree (rightmost leaf on the last level) to the root position, then move it down through the tree as needed until the relationships among the elements are appropriate.

* Heap sort: A heap sort sorts a set of elements by adding each of them into a heap, then removing them one at a time.
* Two ways of organizing heaps:

1. Top down approach: the efficiency is O (nlogn)
2. Bottom up approach: the efficiency is O (n)

* Priority Queue definition: A priority queue is a collection that follows two rules:

1. Item with higher priority go first
2. Items with same priority use first in, first out method to determine their ordering.

* Priority Queue Implementation:

1. Could be implemented by using a list of queues where each queue represents items of a given priority.
2. Using heap to implement Priority Queue, all we need to do is define the priority on the priority queue node, then apply the heap operation based on priority.

Define the PriorityQueueNode that stores, what is the priority:

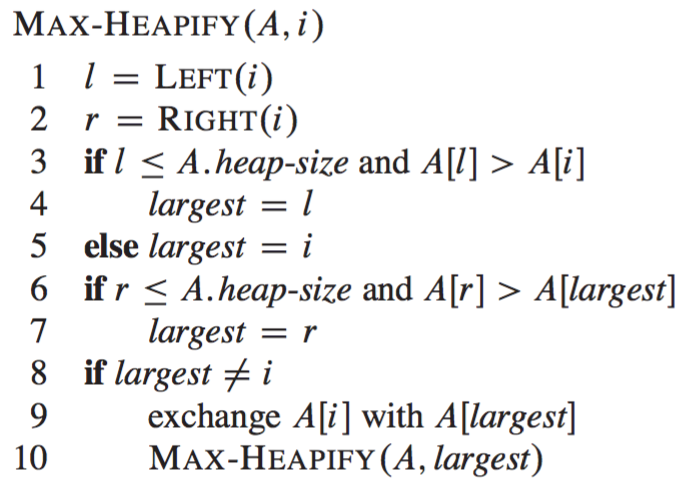
1. the element to be placed on the queue.
2. the priority of the element.
3. the order in which element are placed on the queue.

Then define compareTo () method for PriorityQueueNode class to compare priorities first and then compare the order if there is a tie.

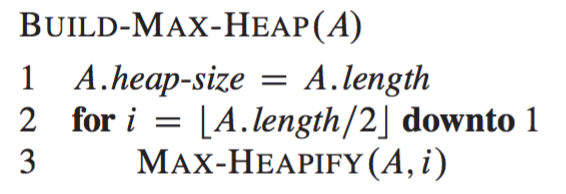
* Implementing heap using array:

A[i] => its ParentIndex = i/2, LeftChildIndex = 2\*i, RightChildIndex = 2\*i + 1

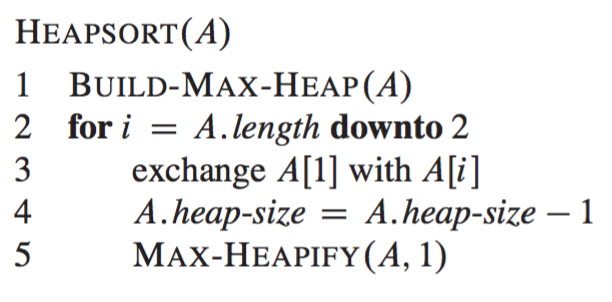
* Core method: Max-Heapify(A, i)



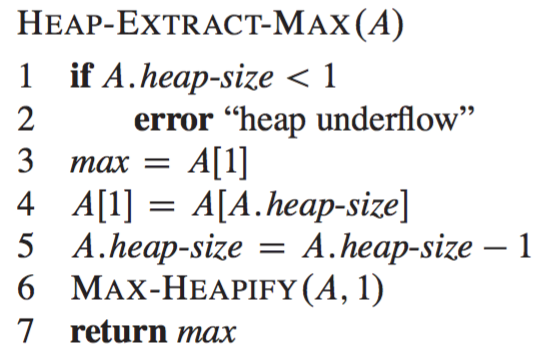
* Build-Max-Heap(A)



* HEAPSORT(A)



* HEAP-EXTRAC-MAX(A)



* MAX-HEAP-INSERT(A, KEY)

A.heap-size = A.heap-size + 1

i = A.heap-size

A[i] = key

While i>1 and A[Parent(i)] < A[i]

Exchange A[i] with A[Parent(i)]

i = Parent(i)

* Two approach to implement heap
  1. Top-down approach

Treat a growing initial sub-array as heap and add element onto it one at a time, and let it float as high as necessary.

* 1. Bottom-up approach

Image there is already a tree structure over the array, and fix the non-leaf node which violate the heap property from bottom to top.

## Graph

* An undirected graph is considered complete if it has the maximum number of edges connecting vertices.
* A path is a sequence of edges which connects two vertices in a graph.
* The length of a path is the number of edges in the path (or the number of vertices – 1).
* An undirected graph considered to be connected if for any two vertices there is a path between them.
* A cycle is a path in which the first and last vertices are the same and none of the edges are repeated.
* An undirected tree is a connected, acyclic, undirected graph with one element designated as root.
* **The traversal of graph**

1. Breadth first

Use a queue to manage the traversal nodes, and use an iterator to build the result

1. Enqueue the starting vertex into the queue and mark the enqueued vertex as visited.

Begin a loop that will continue until the queue is empty.

1. Within this loop, dequeue a vertex from the queue, and add that vertex into the iterator.
2. Enqueue each of the vertices that are adjacent to the current one, and have not already been marked as visited, into the queue. And mark they as visited vertices.

After the loop, the iterator contains the vertices in breath-first order from the giving starting vertex.

1. Depth first

Similar to breath first search with two differences:

1. Replace the queue with stack
2. Mark the vertex as visited not when we push it into stack, but when it has been added into the iterator.

* Testing for connectivity

A graph is connected if and only if for each vertex v in a graph containing n vertices, the size of the result of a breadth-first traversal starting at v is n.

* Minimum Spanning Tree
* Determining the shortest path
* Distance Algorithm: Determine the distance from a given vertex to any other vertices

**d[home] <- 0;**

**enqueue home**

**while queue is not empty do:**

**v <- dequeue**

**for w, each of the neighbors of v:**

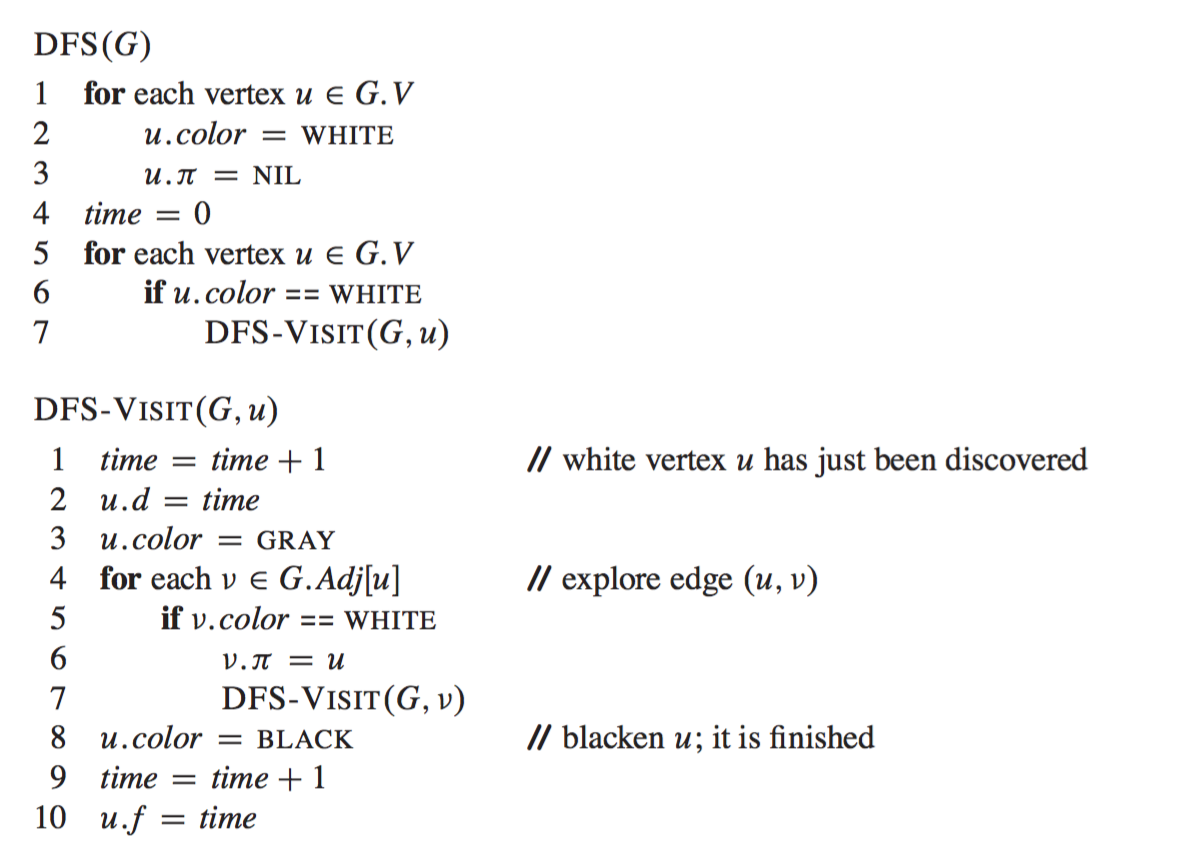
**if(d[w] is not seen before)**

**d[w] = d[v] + 1;**

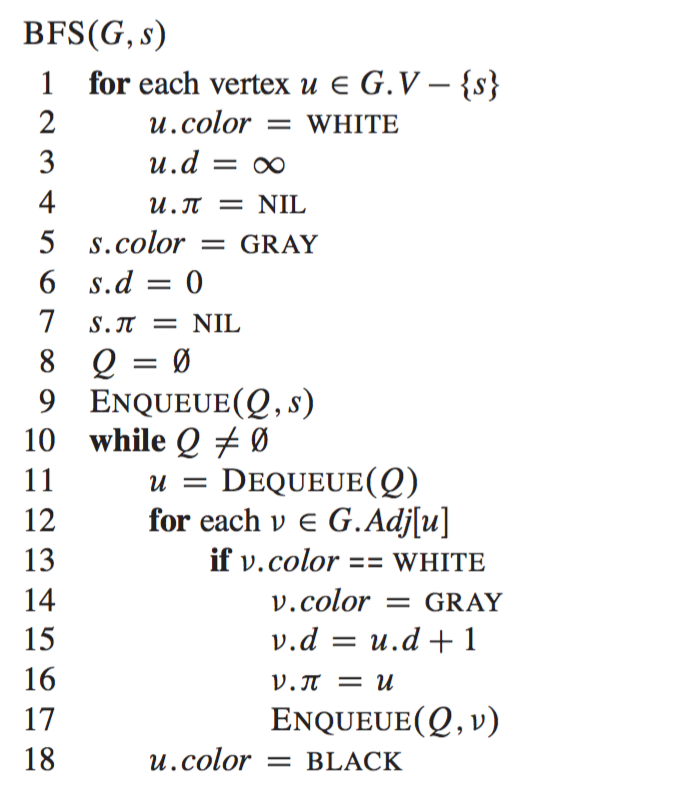
**enqueue w;**

* **Pseudo Code For Traversal of Graph:**

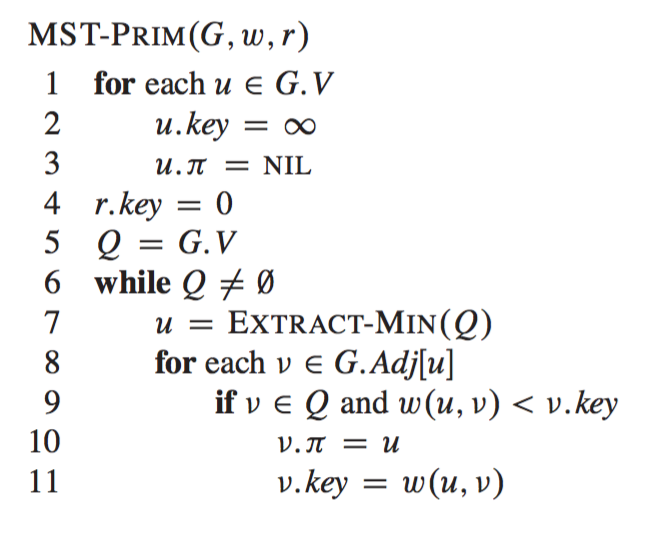
1. **Depth First Search**

****

1. **Breath First Search**

****

* **Minimum Spanning Tree: For each edge (u, v) belongs to E, there is a cost or weight associated with it, w(u, v). The MST is an acyclic subset T belongs to E that connect all the vertices and whose total weight w(T) = Total w (u, v) where (u, v) belongs to T, is minimized.**
* **Pseudo code for Prim’s algorithm**

****