Quiz 6 notes

1. Binary search of an array – O(n)
   1. Find position of a key in a sorted array
   2. binarySearch of key
      1. low = 0, high = size - 1
      2. while high >= low
         1. if middle = key, return true
         2. if middle > key, low = middle + 1
         3. if middle < key, high = middle – 1
      3. otherwise, return false
2. Binary-search tree property
   1. Key in each node must be greater than all keys store in the left sub-tree, not greater than all keys in the right sub-tree
3. Comparison based sort (integers)
   1. mergeSort – recursive divide and conquer – O(nlogn)
   2. if length of a is 1, do nothing
   3. otherwise, split into two halves, and recursively sort them
   4. merge into a
   5. algorithm
      1. if array length <= 1, return
      2. a0 = copy range a, 0, length/2
      3. a1 = copy range a, length/2, length
      4. mergeSort a0, c
      5. mergeSort a1, c
      6. merge a0, a1, a, c
         1. merge
            1. i0 = 0, i1 = 0
            2. loop through length of a

if i0 = length of a0

a[i] = a1[i1++]

else if i1 = length of a1

a[i] = a0[i0++]

else if compare a0[i0], a1[i1] < 0

a[i] = a0[i0++]

else

a[i] = a1[i1++]

* 1. quickSort – another divide and conquer algorithm
     1. pick random pivot element x
     2. partition array into sets of less than, equal to, and greater than x
     3. recursively sort first and third sets in partition
     4. algorithm
        1. quickSort a, c
           1. quickSort a, 0, length, c
        2. quickSort a, i, n, c
           1. n <= 1 return
           2. x = a[i + rand.nextInt(n)]
           3. p = i - 1, j = I, q = i + n
           4. while(j < q)

comp = compare(a[j], x)

if comp < 0

swap a, j++, ++p // move to beginning

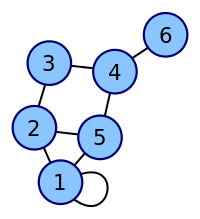
else if comp > 0

swap a, j, --q // move to end

else

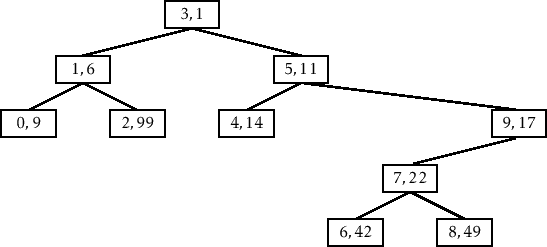
j++ // keep in the middle

* + - * 1. quickSort(a, i, p-i+1, c)
        2. quickSort(a, q, n-(q-i), c)

1. Complete binary tree
   1. Binary tree where every level, except possibly last, is filled, all nodes are far left as possible
2. Connected component in undirected graph
   1. Sub-graph in which any two vertices are connected to each other by paths, connected to no additional vertices in super-graph
3. Connected undirected graph
   1. There is a path from every vertex to every other vertex
4. Cycle in undirected graph
   1. When an unexplored edge leads to a node visited before
5. Breadth-first search of graph
   1. Push the root in Queue
   2. Loop until queue empty
   3. Remove the vertex from the Queue
   4. If removed node has unexplored vertices
      1. Mark them as explored
      2. Insert unexplored vertices in queue
6. Degree of a vertex (undirected graph)
   1. Number of edges incident on it
7. Depth-first search of graph
   1. Push root in Stack
   2. Loop until Stack empty
   3. Peek the node of Stack
   4. If node has unexplored vertices
      1. Get unexplored vertex, mark as explored, push it on Stack
   5. If node does not have any unexplored vertices
      1. Pop the vertex from Stack
8. Directed graph
   1. Set of vertices and collection of directed edges that each connects an ordered pair of vertices
9. Full binary tree
   1. Every node other than the leaves has two children
10. Graph representation: adjacency lists
    1. Vertex-indexed array of lists of vertices connected by an edge to each vertex
11. Graph representation: adjacency matrix
    1. Square matrix used to represent a finite graph
    2. Example graph
       1. 
    3. Corresponding matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |

1. Graph traversal
   1. Process of visiting each vertex in a graph by checking or updating
2. indegree of vertex (directed graph)
   1. Number of edges pointing to it
3. max-heap property
   1. Complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node
4. outdegree of vertex (directed graph)
   1. Number of edges pointing from it
5. Path in graph
   1. Sequence of edges which connect a sequence of vertices
6. Treap
   1. A type of binary tree
   2. Node has data value, x, and unique numerical property, p
   3. Nodes obey heap property
      1. Each parent node has a priority smaller than that of its two children



1. Undirected graph
   1. Graph where edges are bidirectional