Reviews

Dr. 何明昕, He Mingxin, Max

program06 @ yeah.net

Email Subject： CS203B + *ID* + *Name*: *TOPIC*

Sakai: CS203B Fall 2022

数据结构与算法分析B

Data Structures and Algorithm Analysis

Question 1 **Matching** (20×1 point = 20 points)

Fill in each blank represented with a number with parentheses in the sentences using the best LETEER(s) representing the corresponding term(s) from the following alternatives listed below. Or answer a **T** or **F** according to the correctness of each complete statement. Each of which could be re-used (could be the answer for more than one of **(1)~(20)**:

Alternative Answers:

A. computation B. data type C. finite D. delete E. heaps F. false

G. input H. insert I. instructions J. FIFO K. LIFO L. linked lists

M. resizing arrays N. priority queue O. output P. randomized queue Q. queue

R. in-place S. stack T. true X. f(n) = Θ(g(n)) Y. f(n) = O(g(n)) Z. f(n) = Ω(g(n))

An algorithm is a sequence of unambiguous **(1)** for solving a **(2)** problem, i.e., for obtaining a required **(3)** for any legitimate **(4)** in a **(5)** amount of time.

A collection is a **(6)** that stores a group of items. The main operations on a collection are insert and **(7)** items. Which item to delete (remove) determines the nature of a collection. In a stack, removing the item (pop) is done in a **(8)** way. In a queue, removing the item (dequeue) is in a **(9)** way. In a **(10)**, removing the item (deleteMax or deleteMin) is done with the largest (or smallest) item.

(1) I (2) A (3) O (4) G (5) C

(6) B (7) D (8) K (9) J (10) N

Stacks and Queues are normally implemented by **(11)** data structure(s), while priority queues are normally implemented by **(12)**.

A sorting algorithm is **(13)** if it uses $\leqslant$ c log N extra memory.

(14) If $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$, then $T_1(n) = O(T_2(n))$.

(15) Building a heap from an array of n items requires O (n log n) time.

(16) For large input sizes, mergesort will always run faster than insertion sort (on the same input).

(17) To prevent too many recursive call for tiny sized array slice in mergesort or quicksort, in practice to enhance efficiency normally use cutoff to insertion sort when the length of slice is small enough.

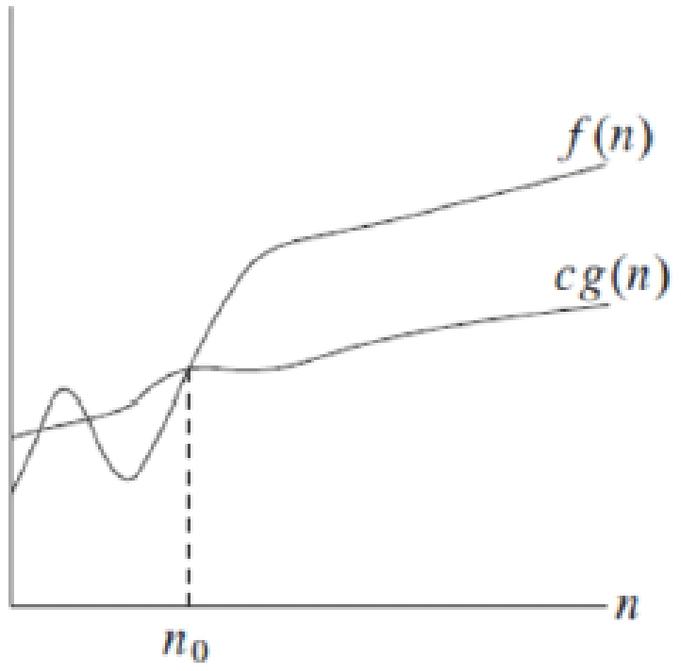
(11) L,M (12) E (13) R (14) F (15) T

(16) F (17) T

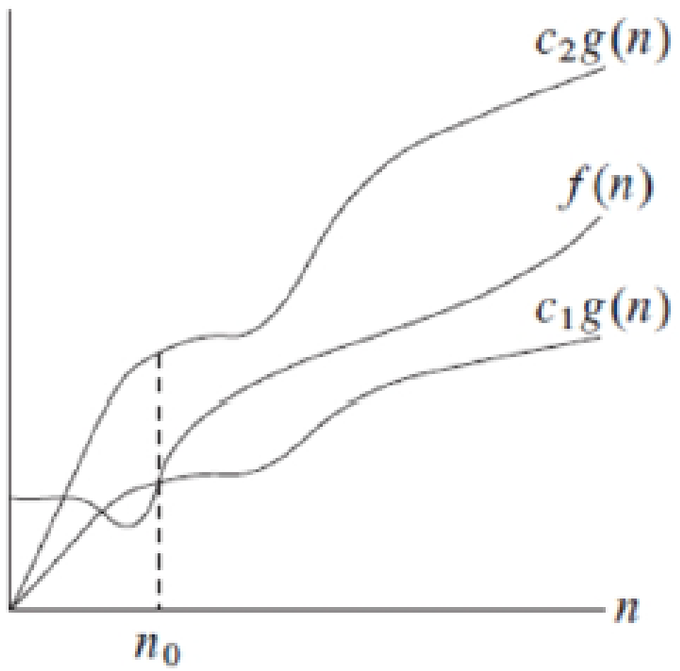
Alternative Answers:

A. computation B. data type C. finite D. delete E. heaps F. false
G. input H. insert I. instructions J. FIFO K. LIFO L. linked lists
M. resizing arrays N. priority queue O. output P. randomized queue Q. queue
R. in-place S. stack T. true X. f(n) = Θ(g(n)) Y. f(n) = O(g(n)) Z. f(n) = Ω(g(n))

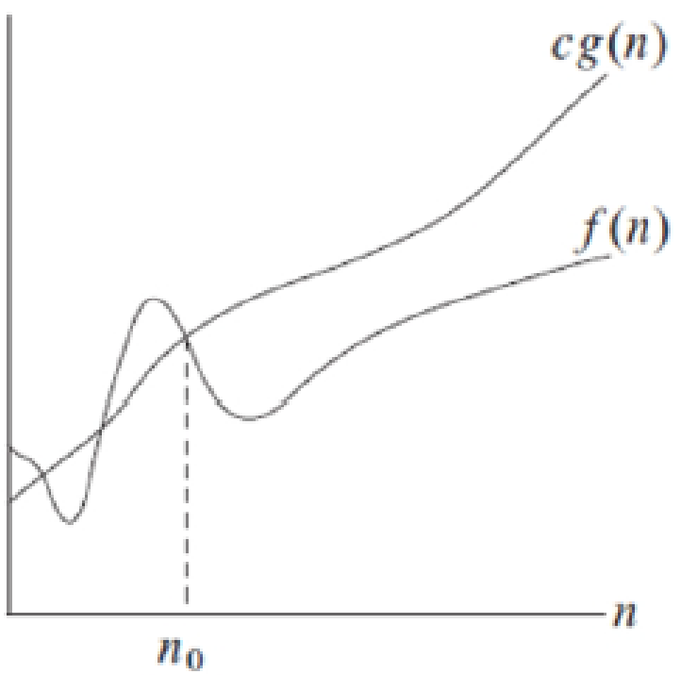
(18)~(20) are based on the following 3 plots. Please identify the corresponding asymptotic notations represented in the following 3 plots.



(18)



(19)



(20)

(18) Z (19) X (20) Y

X. f(n) = Θ(g(n)) Y. f(n) = O(g(n)) Z. f(n) = Ω(g(n))

Question 2 On Big-O (8×1.5 points = 12 points)

For each of the functions f(N) given below, indicate the **tightest bound** possible (in other words, giving $O(2^N)$ as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. **You MUST choose your answer from the following** (not given in any particular order), each of which could be re-used (could be the answer for more than one of **(21)~(28)**:

$O(N^2)$, $O(N^{1/2})$, $O(N^{1/4})$, $O(\log^3 N)$, $O(N)$, $O(N^2 \log N)$, $O(N^5)$, $O(2^N)$, $O(N^3)$, $O(N^8)$, $O(\log^4 N)$, $O(N \log^3 N)$, $O(N^2 \log^2 N)$, $O(\log N)$, $O(1)$, $O(N^4)$, $O(N^N)$, $O(N^6)$, $O(N \log^2 N)$, $O(\log \log N)$

You do not need to explain your answer.

(21) $f(N) = N^2 \log N^2 + 2N \log^2 N$

(22) $f(N) = (\ N\ (100N + 5 + N^3)\ )^2$

(23) $f(N) = 1000 \log \log N + \log N$

(24) $f(N) = \log_{16}(2^N)$

(25) $f(N) = N^2\ (\log N^3 - \log N) + N^2$

(26) $f(N) = (N \log N + 2N)^2$

(27) $f(N) = N^{1/2} + \log^3 N$

(28) $f(N) = 100 \log N + N^{1/4}$

(21) $O(N^2 \log N)$

(22) $O(N^8)$

(23) $O(\log N)$

(24) $O(N)$

(25) $O(N^2 \log N)$

(26) $O(N^2 \log^2 N)$

(27) $O(N^{1/2})$

(28) $O(N^{1/4})$

Question 3 Big-Oh and Run Time Analysis (4×3 points = 12 points)

Describe the worst case running time of the following pseudocode functions in Big-Oh notation in terms of the variable n. Showing your work is not required (although showing work may allow some partial credit in the case your answer is wrong – don't spend a lot of time showing your work.). You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one of **(29)~(32)**):

$O(n^2)$, $O(n^3 \log n)$, $O(n \log n)$, $O(n)$, $O(n^2 \log n)$, $O(n^5)$, $O(2^n)$, $O(n^3)$, $O(\log n)$, $O(1)$, $O(n^4)$, $O(n^n)$

(29)
```
void doWhat(int n, int x, int y) {
    if (x < y) {
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n * i; ++j)
                System.out.println("y = " + y);
    } else {
        System.out.println("x = " + x);
    }
}
```

(29) $O(n^3)$

(30) $O(N)$

(31) $O(N^2)$

(32) $O(N^5)$

(30)
```
int doWhat(int n, int m) {
  if (n < 1) return n;
  if (n < 100) return doWhat(n - m, m);
  return doWhat(n - 1, m);
}
```

(31)
```
void doWhat(int n) {
  int j = 0;
  while (j < n) {
    for (int i = 0; i < n; ++i) {
      System.out.println("j = " + j);
    }
    j = j + 5;
  }
}
```

(32)
```
void doWhat(int n) {
  for (int i = 0; i < n * n; ++i) {
    for (int j = 0; j < n; ++j) {
      for (int k = 0; k < i; ++k)
        System.out.println("k = " + k);
      for (int m = 0; m < 100; ++m)
        System.out.println("m = " + m);
    }
  }
}
```

(29) $O(n^3)$

(30) $O(N)$

(31) $O(N^2)$

(32) $O(N^5)$

Question 4 On Quick-sort (2×5points = 10 points)

Suppose a quick-sort program is in execution. The partition method has receaved a char array slice **a[low..high] = [J U S T F O R E X A M]**, ignore syntax symbols like single quote, please answer question **(33)** and **(34)**:

(33) *(5 points)* If choose **a[low]** as the pivot, write down the result of **a[low..high]** after the partition finished.

(34) *(5 points)* If choose **median3(a[low], a[low + (high-low)/2], a[high])** as the pivot, write down the result of **a[low..high]** after the partition finished.

(33) a[low..high] = [F A E J T O R S X U M]

(Remark: It may have mutiple correct answers. The key points are **J** should be in the 4th position (2 points); **[F A E]** in any order should be on the left of **J** (1.5 points), **[T O R S X U M]** in any order should be on the right of **J** (1.5 points).)

(34) a[low..high] = [J A E F M U S T O R X]

(Remark: It may have mutiple correct answers. The key points are **M** should be in the 5th position(2 points), **[J A E F]** in any order should be on the left of **M** (1.5 points), **[U S T O R X]** in any order should be on the right of **M** (1.5 points).)

Question 5 h-sorting in Shell Sort (3 * 4 points = 12 points)

An h-sorted array is h interleaved sorted subsequences. Shellsort is to h-sort an array sucecessively with a decreasing sequence of values of h. Suppose the content of `char[] a` is `[J U S T M I D T E R M E X A M]`, ignore syntax symbols like single quote, please write down the following results in a sucecessive series :

(35) *(4 points)* 13-sort of array **a**;

(36) *(4 points)* 4-sort of the result of question **(35)**;

(37) *(4 points)* 1-sort of the result of question **(36)**.

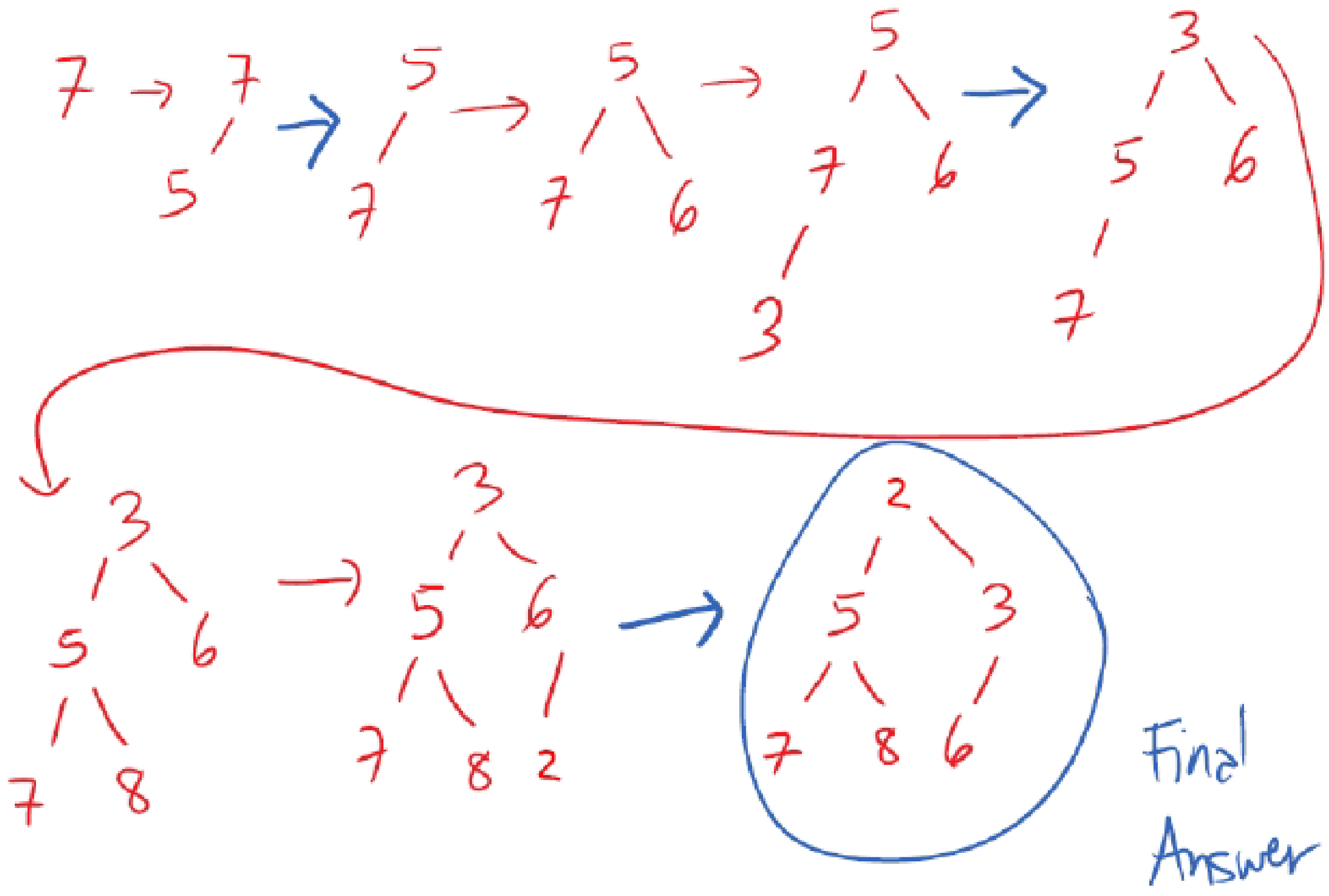
```
(35) 13-sort:  A M S T M I D T E R M E X J U

(36)  4-sort:  A I D E E J M T M M S T X R U

(37)  1-sort:  A D E E I J M M M R S T T U X
```

Question 6 On Binary Min Heaps (7 + 3 = 10 points)

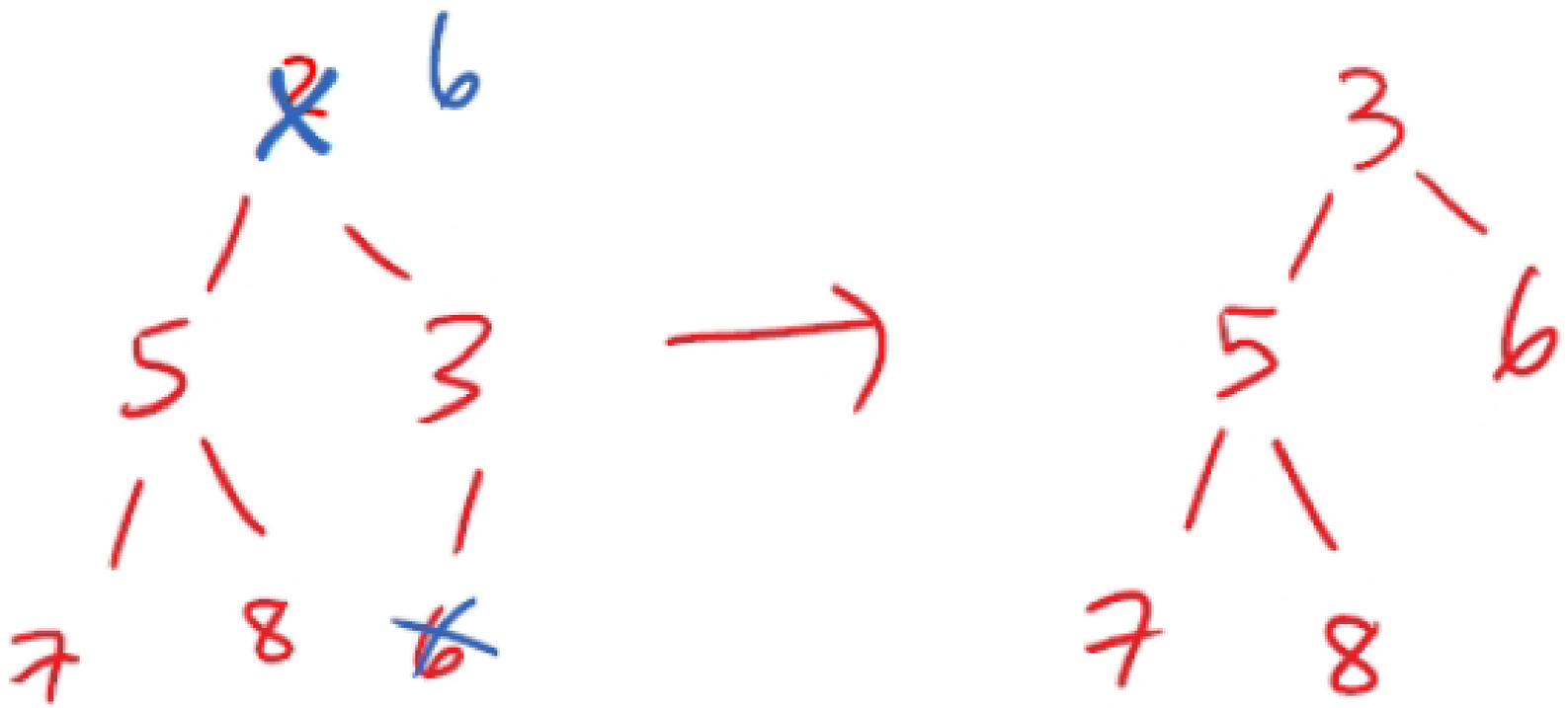
(38) (*7 points)* Draw the binary min heap that results from inserting the integers: 7, 5, 6, 3, 8, 2 in that order into an initially empty binary min heap. You do not need to show the array representation of the heap. You are only required to show the final tree, although drawing intermediate trees may result in partial credit. If you draw intermediate trees, please circle your final result for any credit.

(39) *(3 points)* Draw the result of one deleteMin call on your heap drawn at the end of question **(38) with intermediate trees** .

(38) (*7 points)* The constructing procedure of the binary min tree:



(39) *(3 points)* The operation procedure of a deleteMin from the binary min tree in the result of question **(38)** :



Question 7 **Programming** (3 * 10 points = 30 points)

(40) (*10 points*) Complete the Java method so that it properly implements insertion sort. Make sure the method is complete (no any extra helping method needed) and the result is in-place (do not use another array) .

```
public static int[] insertionSort (int[] a) {
   for (int i = 1; i < a.length; i++) {
      // YOUR CODE HERE
      // …
      // …
   } // end of the for-loop
   return a;
}
```

```
// YOUR CODE HERE
for (int j = i; j > 0; j--)
   if (a[j] < a[j-1]) {
      int temp = a[j-1];
      a[j-1] = a[j];
      a[j] = temp;
   } else break;
// …
```

(41) *(10 points)* Complete the Java method so that it properly implements selection sort. Make sure the method is complete (no any extra helping method needed) and the result is in-place (do not use another array). Notice there are two places to add code.

```
public static void selectionSort (int[] a) {
   for (int i = 0; i < a.length; i++) {
      int idx = i;  // index_of_least
      int j;
      for (j = i+1; j < a.length; j++) {
         // YOUR CODE HERE #1
         // …
      } // end of the inner for-loop

      // YOUR CODE HERE #2
      // …
      // …
   } // end of the outer for-loop
}
```

```
// YOUR CODE HERE #1
if (a[j] < a[idx])
   idx = j;
// …

// YOUR CODE HERE #2
int temp = a[i];
a[i] = a[idx];
a[idx] = temp;
// …
```

(42) *(10 points)* Complete the helping method **binarySearch** so that it helps to properly implement binary search in a sorted array for the method **indexOf**. Both methods return the index found in the array or -1 when not found. Make sure the method **binarySearch** is complete (no any extra helping method needed).

```
public static <E extends Comparable<E>>
int indexOf (E x, E[] a) {
   // a is sorted by comparing with compareTo()
   return binarySearch( x, a, 0, a.length-1 );
}
```

```
private static <E extends Comparable<E>>
int binarySearch (E x, E[] a, int low, int high) {
   // Tips: use x.compareTo(y) to compare 2 Es

   // YOUR CODE HERE
   // …
   // …
}
```

```
// YOUR CODE HERE
if (low > high) return -1;

int lo = low, hi = high;
while (lo <= hi) {
   int mid = (lo + hi) / 2;
   int comp = x.compareTo(a[mid]);
   if (comp == 0) return mid;
   if (comp <  0) hi = mid - 1;
   else           lo = mid + 1;
}
return -1;
// …
```

```
// YOUR CODE HERE
if (low > high) return -1;
int mid = (low + high) / 2;
int comp = x.compareTo(a[mid]);
if (comp == 0) return mid;
if (comp <  0) return binarySearch( x, a, low, mid-1);
else           return binarySearch( x, a, mid+1, high);
// …
```

CS203B-22s Exercise on April 24

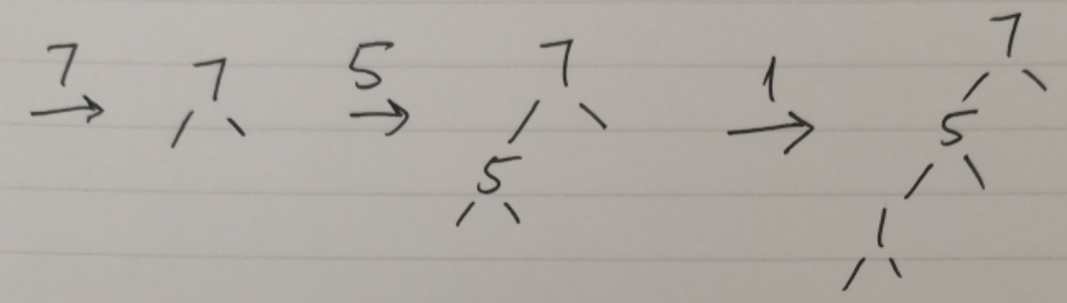
1. Draw the binary search tree (BST) that results from inserting the integers: 7, 5, 1, 6, 3, 4, 8, 2 in that order into an initially empty BST with intermediate trees.

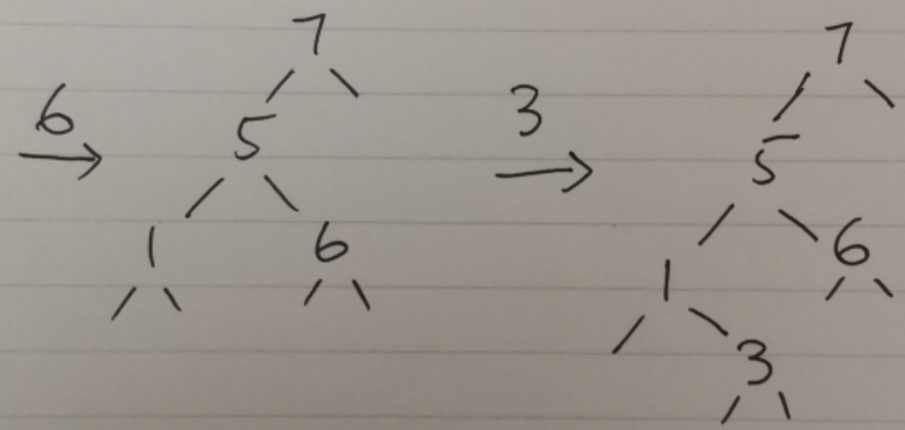
2. Draw the 2-3 tree that results from inserting the integers: 7, 5, 1, 6, 3, 4, 8, 2 in that order into an initially empty 2-3 tree with intermediate trees.

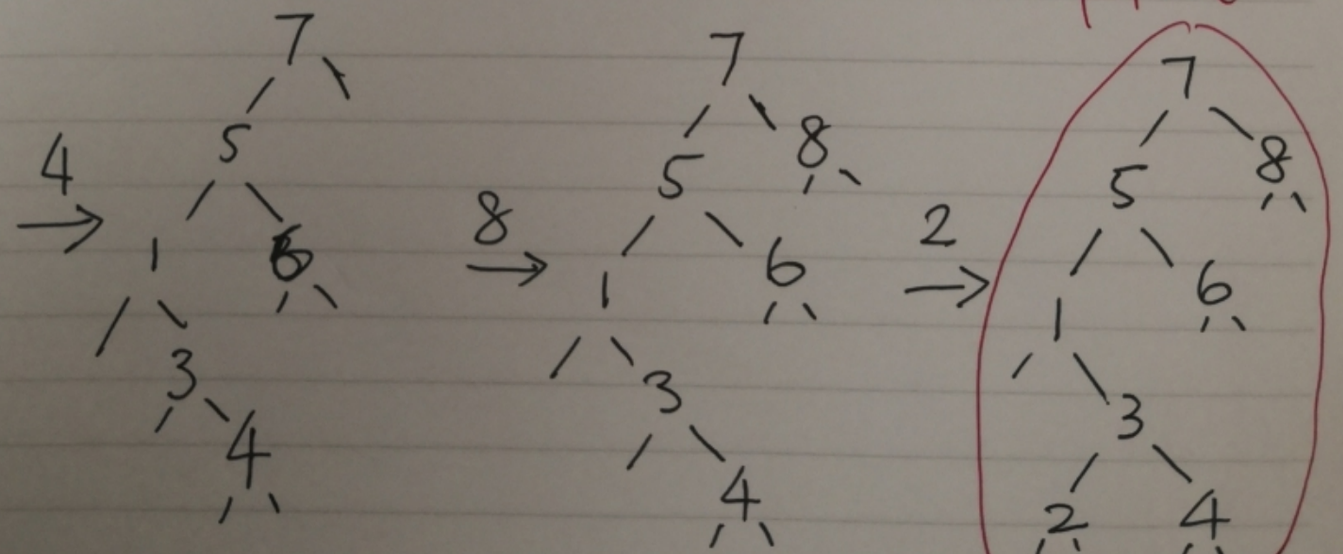
No.

Date

1. Ans. construct BSTs with keys in order 7, 5, 1, 6, 3, 4, 8, 2



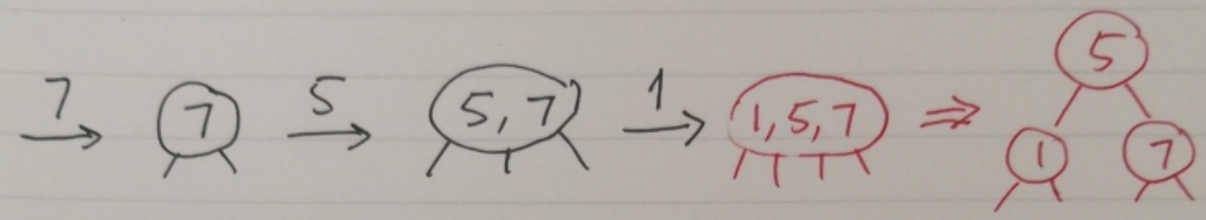


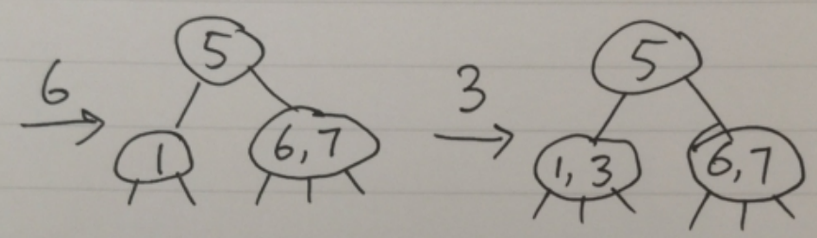


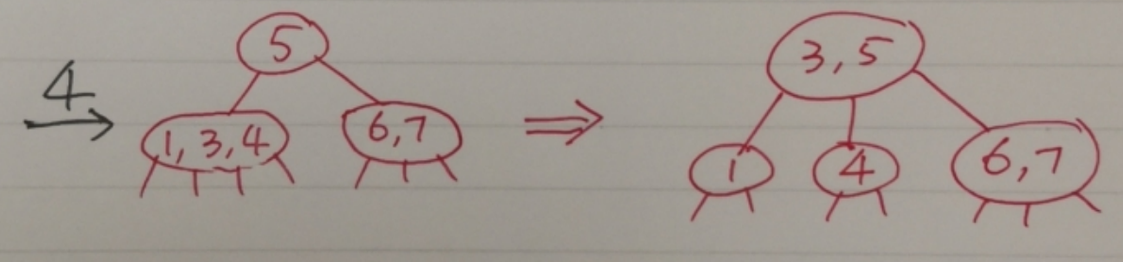
No.

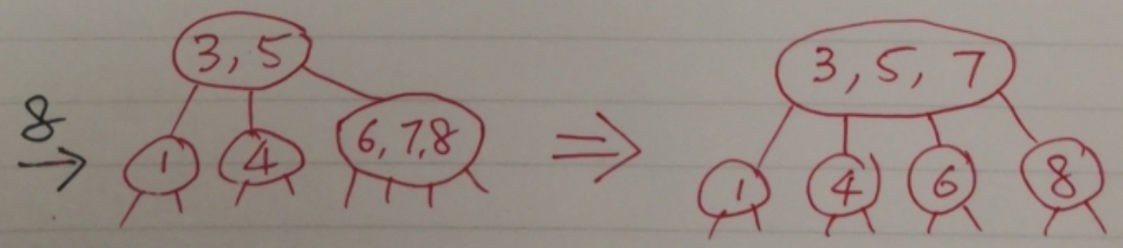
Date

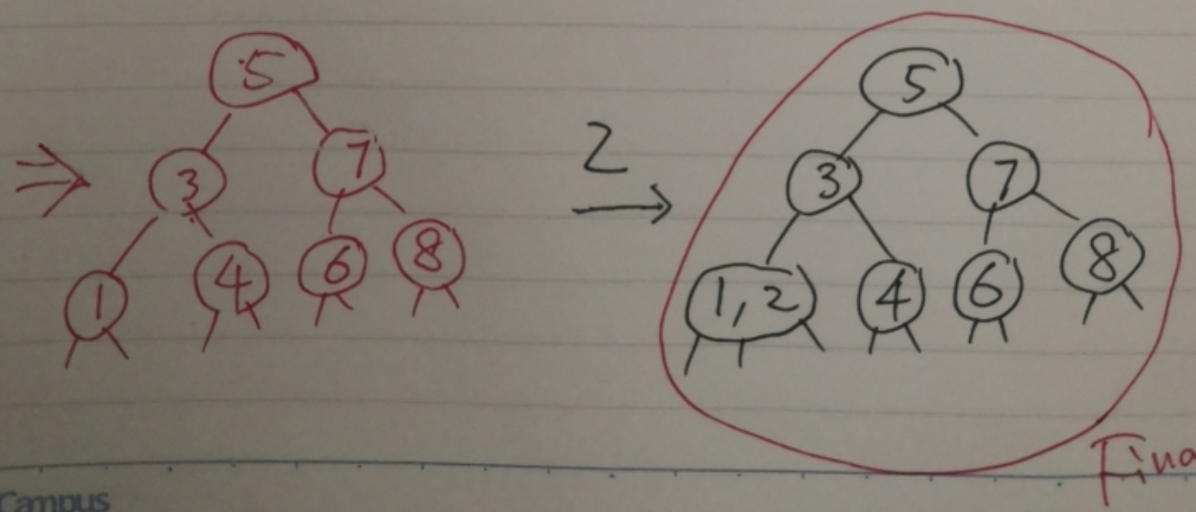
2. Ans. construct 2-3 tree with ~~nodes~~ keys in order 7, 5, 1, 6, 3, 4, 8, 2











Final

Campus

Hashing:

a) Draw the contents of the two hash tables below after inserting the values shown. Show your work for partial credit. *If an insertion fails, please indicate which values fail and attempt to insert any remaining values*. The hash function used is H(k) = k mod tablesize.

Table 1: **Separate chaining**, (where each bucket points to a linked list *sorted from smallest to largest*)

Insert: 14, 23, 2, 19, 20, 5

0	
1	
2	
3	
4	
5	

Table 2: **Linear Probing**

Insert: 9, 1, 17

0	
1	
2	
3	
4	
5	
6	
7	

b) Give the load factor for each table:

Load factor for Table 1:

Load factor for Table 2:

Hashing:

a) Draw the contents of the two hash tables below after inserting the values shown. Show your work for partial credit. *If an insertion fails, please indicate which values fail and attempt to insert any remaining values*. The hash function used is H(k) = k mod tablesize.

Table 1: **Separate chaining**, (where each bucket points to a linked list *sorted from smallest to largest*)

Insert: 14, 23, 2, 19, 20, 5

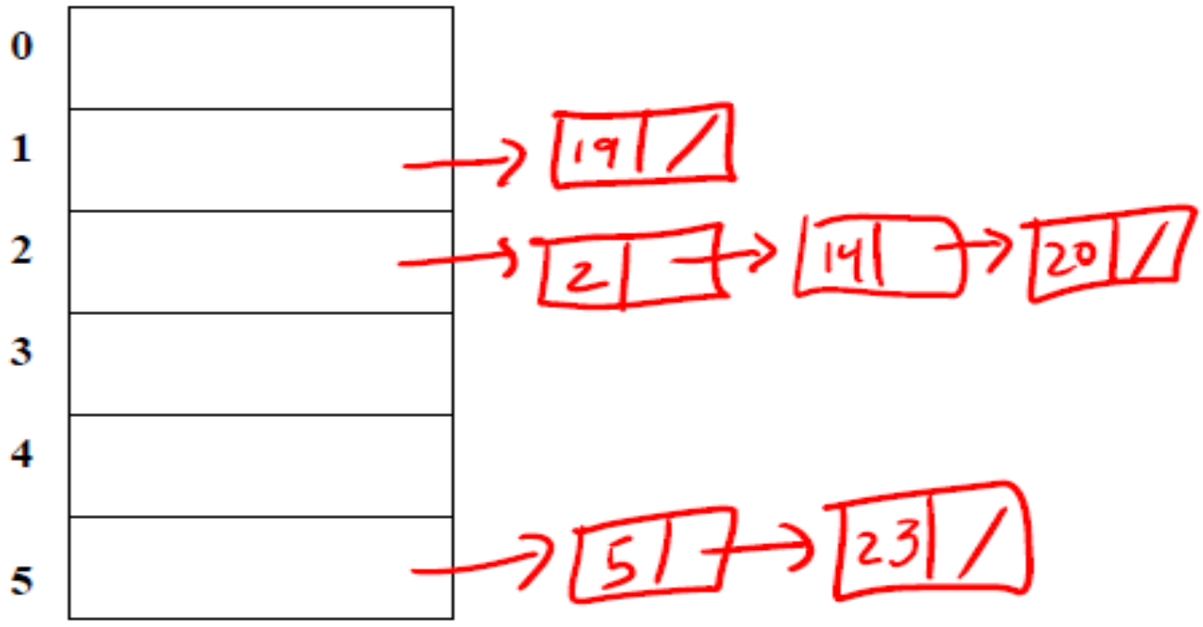
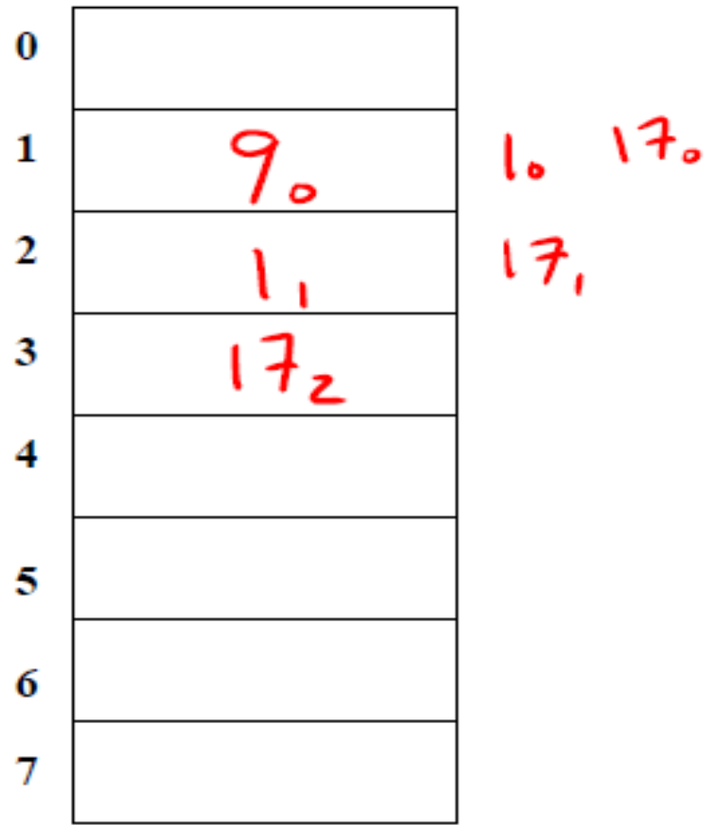


Table 2: **Linear Probing**

Insert: 9, 1, 17



b) Give the load factor for each table:

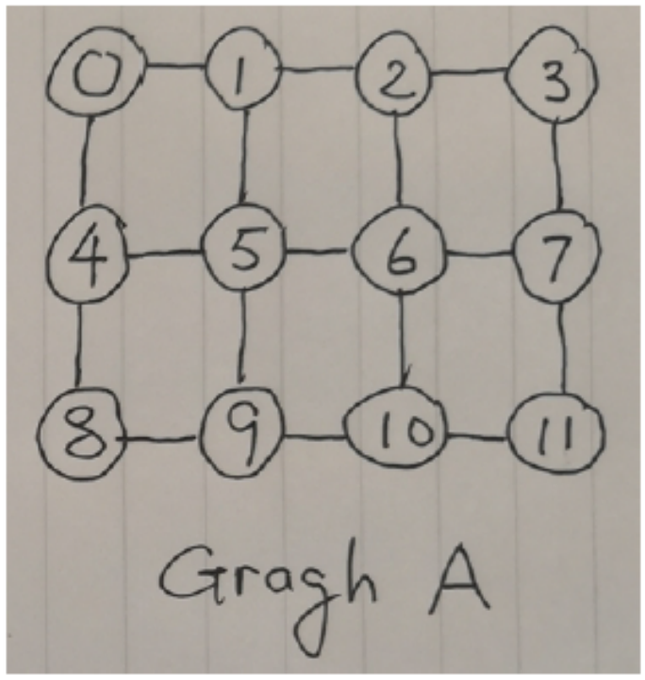
Load factor for Table 1:

6/6 = 1

Load factor for Table 2:

3/8

Question I Give a Graph A as shown:



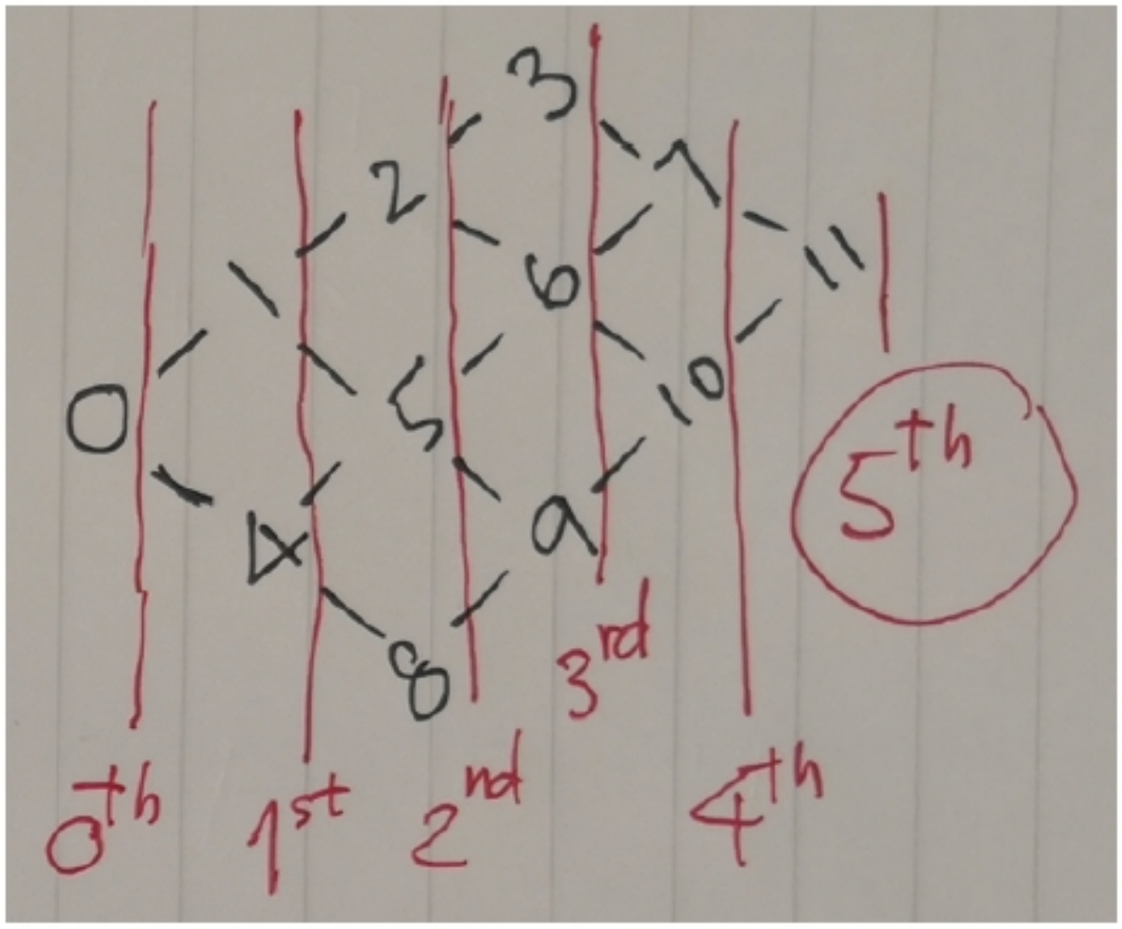
1) What is the diameter of Graph A? (The diameter of a graph is defined as the **largest shortest path distance in the graph**. In other words, it is the maximum value of over all pairs, where denotes the shortest path distance from vertex to vertex.)

2) Write down a depth-first search visiting order starting from 0 to visit all vertices in Graph A.

3) Write down a breadth-first serach visiting order starting from 0 to visit all vertices in Graph A.

Ref. Ans. For Question I:

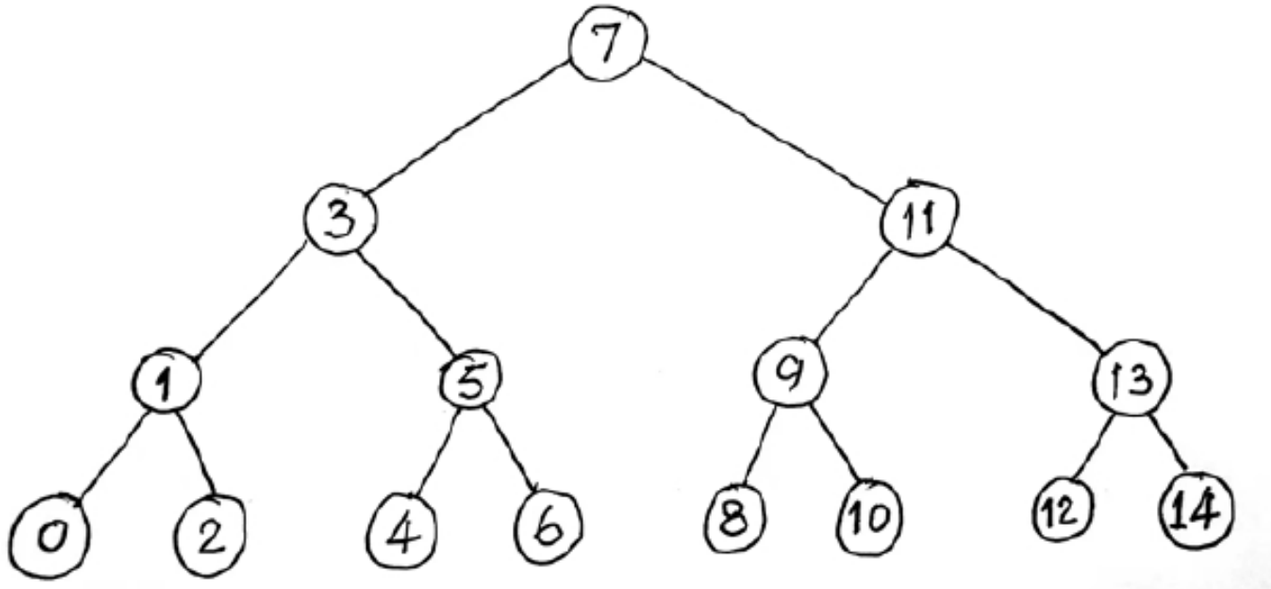
1) The diameter of Graph A is 5, as shown in the following figure:



2) One of the DFS visiting order (not unique): 0, 1, 2, 3, 7, 6, 5, 4, 8, 9, 10, 11.

3) One of the BFS visiting order (not unique): 0, 1, 4, 2, 5, 8, 3, 6, 9, 7, 10, 11.

Question II Give a binary tree as in Graph B.



Graph B A binary tree to be visited

4) Write down the vertex series using depth-first search by pre-order to visit all vertices in Graph B.

5) Write down the vertex series using depth-first search by in-order to visit all vertices in Graph B.

6) Write down the vertex series using breadth-first search by pre-order to visit all vertices in Graph B.

Ref. Ans. For Question II:

4) **DFS by pre-order:** 7, 3, 1, 0, 2, 5, 4, 6, 11, 9, 8, 10, 13, 12, 14.

5) **DFS by in-order:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14.

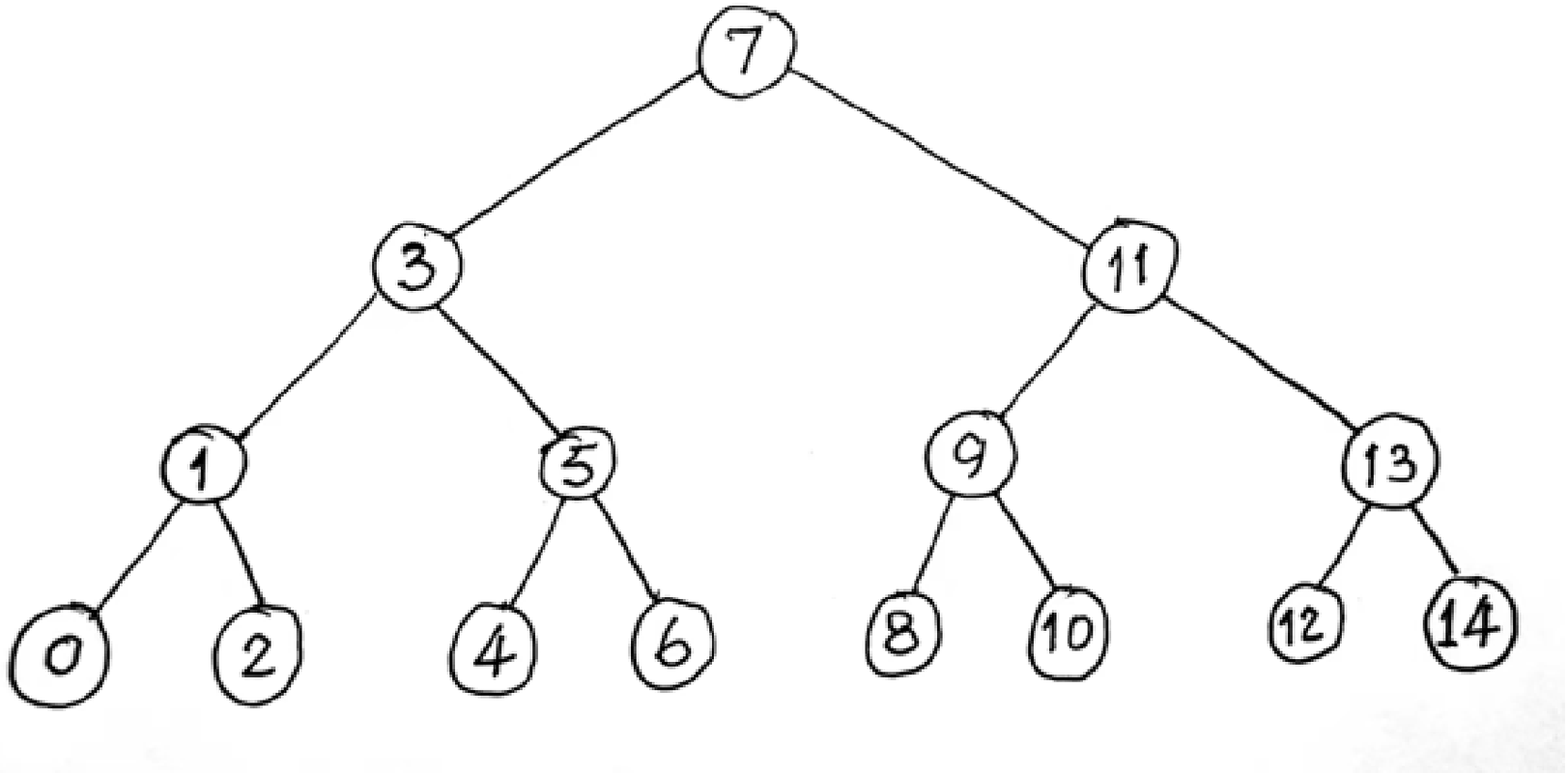
6) **BFS by pre-order:** 7, 3, 11, 1, 5, 9, 13, 0, 2, 4, 6, 8, 10, 12, 14.

BiTreeTraverse.java

```java
    private static <E> void DFS_pre_order (BiTree<E> t) {
        if (t == null) return;
        processNode( t );
        DFS_pre_order( t.left );
        DFS_pre_order( t.right );
    }

    private static <E> void DFS_in_order (BiTree<E> t) {
        if (t == null) return;
        DFS_in_order( t.left );
        processNode( t );
        DFS_in_order( t.right );
    }

    private static <E> void DFS_post_order (BiTree<E> t) {
        if (t == null) return;
        DFS_post_order( t.left );
        DFS_post_order( t.right );
        processNode( t );
    }

```

BiTreeTraverse.java

```java

    private static <E> void BFS_pre_order (BiTree<E> t) {
        Queue<BiTree<E>> q = new LinkedList<BiTree<E>>();
        q.add( t );
        while (q.size() != 0) {
            BiTree<E> n = q.remove();
            if (n.left != null)  q.add( n.left );
            if (n.right != null) q.add( n.right );
            processNode( n );
        }
    }

    private static <E> void processNode (BiTree<E> t) {
        System.out.print( t == null ? "" : t.value.toString() + " " );
    }

```



Graph B A binary tree to be visited

```
H:\work\2022f\BiTreeTraverse>javac BiTreeTraverse.java

H:\work\2022f\BiTreeTraverse>java BiTreeTraverse
DFS_pre_order  : 7 3 1 0 2 5 4 6 11 9 8 10 13 12 14
DFS_in_order   : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
DFS_post_order : 0 2 1 4 6 5 3 8 10 9 12 14 13 11 7
BFS_pre_order  : 7 3 11 1 5 9 13 0 2 4 6 8 10 12 14
```

Depth-first search orders

Observation. DFS visits each vertex exactly once. The order in which it does so can be important.

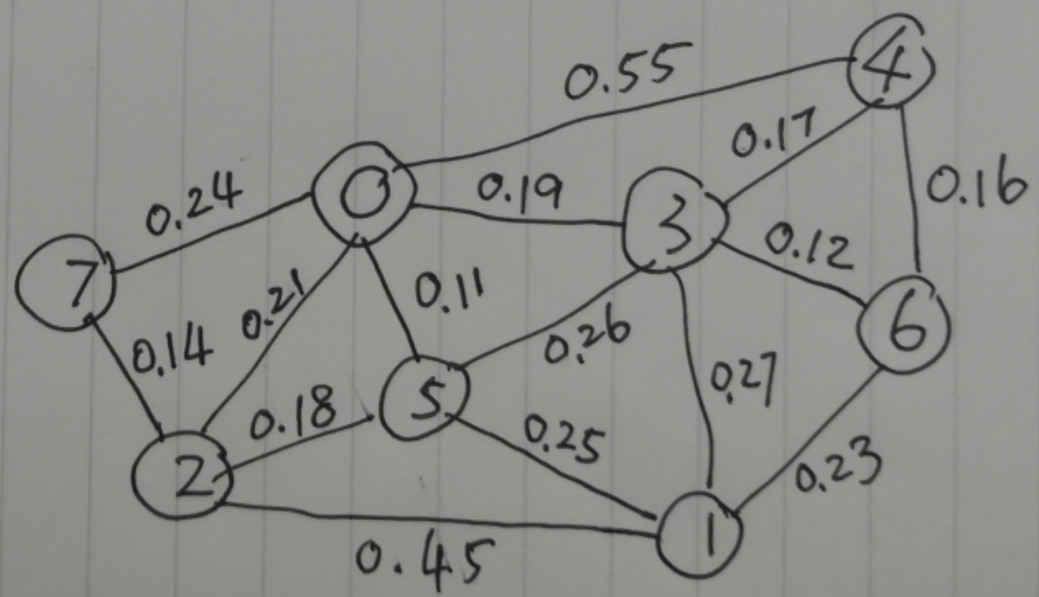
Orderings.

- Preorder: order in which `dfs()` is called.
- Postorder: order in which `dfs()` returns.
- Reverse postorder: reverse order in which `dfs()` returns.

```
private void dfs (Graph G, int v) {

   marked[v] = true;
   preorder.enqueue(v);
   for (int w : G.adj(v))
      if (!marked[w]) dfs(G, w);
   postorder.enqueue(v);
   reversePostorder.push(v);
}
```

CS203B-f21 Quiz on Dec. 14

Use Kruskal's algorithm (Add next edge to tree T unless doing so would create a cycle.) to find the minimum spanning tree (MST) for the following edge-weighted graph. Write down the edges in the MST in their finding order.

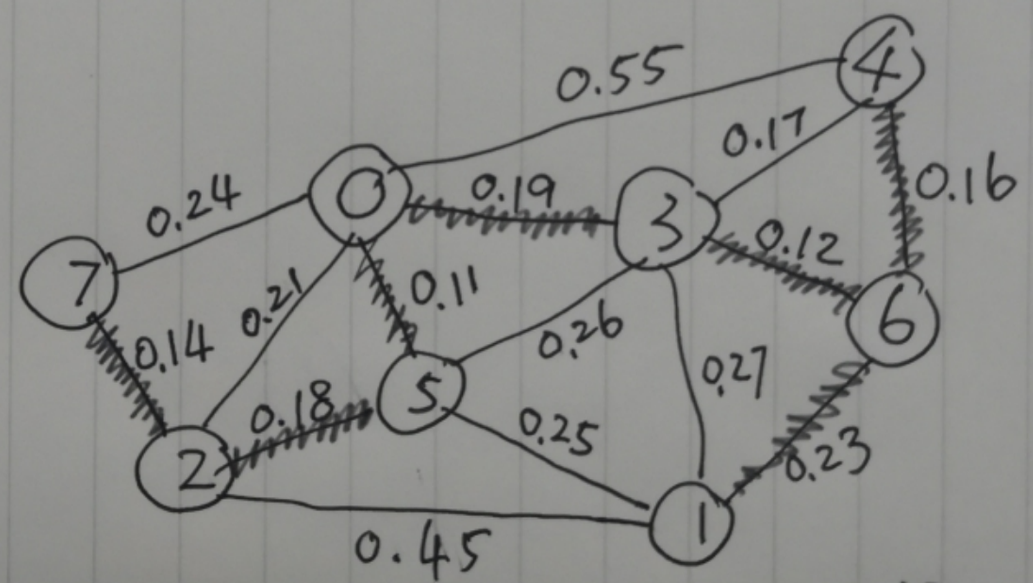


CS203B-f21 Quiz on Dec. 14

Use Kruskal's algorithm (Add next edge to tree T unless doing so would create a cycle.) to find the minimum spanning tree (MST) for the following edge-weighted graph. Write down the edges in the MST in their finding order.

Ref. Answer:

0-5, 3-6, 7-2,
4-6, 2-5, 0-3,
1-6.



Weight = 0.11 + 0.12 + 0.14 + 0.16 + 0.18 + 0.19 + 0.23 = 1.13

LSD string sort.

Suppose we use **LSD string sort** (least-significant-digit-first sort, **radix sort**) to sort the numbers below, using a radix of 10. Show the state of the sort after each of the first two passes, no need to show the state of the third pass when the sorting is complete. If multiple numbers are in a bucket, put the numbers that “come first" closer to the top.

Numbers to sort (in their initial order):

`13, 105, 492, 776, 1, 6, 214, 99, 98, 96, 11, 21`

And the final result sorted should be (It means that digits are right alignmented. That is all `0`s for shorter numbers, e.g. , look `1` as `001` and `13` as `013`):

`1, 6, 11, 13, 21, 96, 98, 99, 105, 214, 492, 776`

(1) Put the result after the **first pass** in the following table:

0	1	2	3	4	5	6	7	8	9

(2) Put the result after the **second pass** in the following table:

0	1	2	3	4	5	6	7	8	9

(3) Put the result after the **third pass** in the following table:

0	1	2	3	4	5	6	7	8	9

LSD string sort

Numbers to sort (in their initial order):

13, 105, 492, 776, 1, 6, 214, 99, 98, 96, 11, 21

And the final result sorted should be (It means that digits are right alignmented. That is all 0s for shorter numbers, e.g. , look **1** as **001** and **13** as **013**):

1, 6, 11, 13, 21, 96, 98, 99, 105, 214, 492, 776.

(1) Put the result after the **first pass** in the following table:

0	1	2	3	4	5	6	7	8	9
	1 11 21	492	13	214	105	776 6 96		98	99

(2) Put the result after the **second pass** in the following table:

0	1	2	3	4	5	6	7	8	9
1 105 6	11 13 214	21					776		492 96 98 99

(3) Put the result after the **third pass** in the following table:

0	1	2	3	4	5	6	7	8	9
1 6 11 13 21 96 98 99	105	214		492			776		

Question On Tidle Notation

By definition, $f(N) \sim g(N)$ means $\lim_{N \to \infty} \frac{f(N)}{g(N)} = 1$

In our text, we use tidle notation f(N) ~ g(N) instead of Big-O notation to estimate running time (or memory) as a function of input size N. The key idea is to keep the leading term in g(N) with coefficient and to ignore lower order terms in f(N). When N is large, those lower order terms are negligible; when N is small, we don't care.

Please write down **the corresponding ~g(N) for the following f(N)**. (*You do not need to explain your answer.)*

(16) $f(N) = 3\ N^2 \log N + 7\ N \log^2 N$

(17) $f(N) = (\ N\ (100\ N + 2\ N^2)\ )^2$

(18) $f(N) = N^2\ (6 \log N^3 - 10 \log N) + N^2$

(19) $f(N) = N\ (4\ N \log N - \log N) + 2\ (N^2)^{3/2}$

Reference Answers:

(16) **~ 3 N^2 log N**

(17) **~ 4 N^6**

(18) **~ 8 N^2 log N**

(19) **~ 2 N^3**