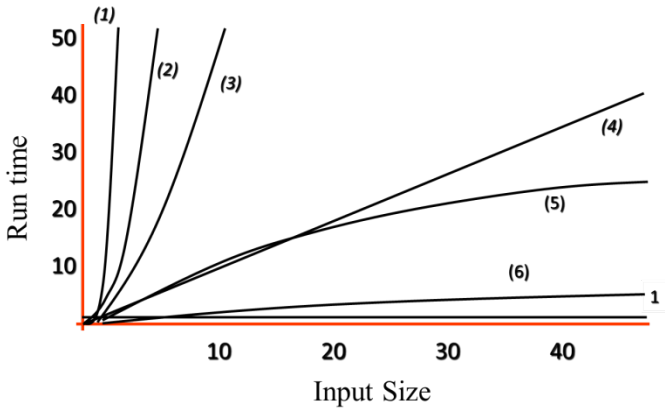
1. **Complexity**

Match each curve in the graph with one of the O() functions -- $N\log_2 N$, N^3, $(\log_2 N)^2$, N, N^2, $\log_2 N$.



Curve	O() function
(1)	N^3
(2)	N^2
(3)	$N\log_2 N$
(4)	N
(5)	$(\log_2 N)^2$
(6)	$\log_2 N$

Solution: see slides.

2. **Recursive algorithm.** Write pseudo-code to solve the following two problems.

(1) Print elements of a linked list in reverse order by using same single linked list.
Solution: See homework.

(2) Definition: to rotate an array by k times, we shift all items in the array to the right by k slots, and put the most right k items in the original array to the left of the rotated array. For instance:

Original array: **1, 3, 6, 9, 12, 15**
Rotated by 2: **12, 15, 1, 3, 6, 9**

Find the minimum element in a rotated sorted array (we don't know how many time it has been rotated).
Pay attention to the "stopping condition"
What if we have duplicated items?

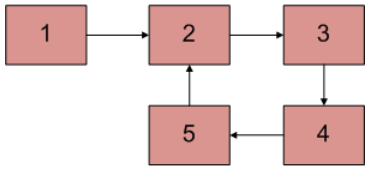
See http://www.geeksforgeeks.org/find-minimum-element-in-a-sorted-and-rotated-array/
See https://stackoverflow.com/questions/8532833/find-smallest-number-in-sorted-rotatable-array

3. **Singly Linked List.** In text (bulleted list) or pseudo-code, describe the algorithms. Assume the length of the list is N, state the space and time complexity.
(1) Write an algorithm to retrieve the middle element in a singly linked list.

Method 1: Scan the list once to get the length L; then use a pointer to iterate over L/2 items.

Method 2: using two pointers, one fast (each time going over 2 items) and one slow (each time going over 1 item). Once the fast one reaches the end, the slow one points to the middle point.

(2) A malformed singly linked list with a loop (see the mini example below) causes iteration over the list to fail because the iteration will never reach the end of the list. Write an algorithm to detect loop in a singly linked list.



The "removal" part is more difficult.
Solution: http://www.geeksforgeeks.org/detect-and-remove-loop-in-a-linked-list/

4. **Sorting**. Sort an array containing the digits 71808294 in ascending order. Show how the order of the digits changes during each step of the following sorting algorithms. Clearly state what data structures are used in each algorithm (tree, array, list, etc).

(1) insertion sort

(2) selection sort

(3) mergesort

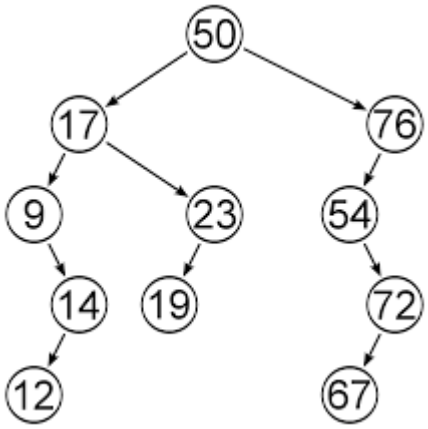
(4) quicksort (always choosing the first element of any subarray to be the pivot)

(5) quicksort (always choosing the last element of any subarray to be the pivot)

(6) heapsort

See Lecture Slides

5. **Trees**. Traverse the following tree.



(1) In-order traversal: show the sequence of nodes being visited.
9,12,14,17,19,23,50,54,67,72,76

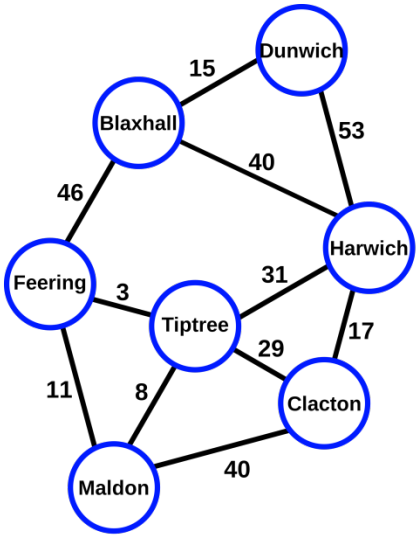
(2) In-order + iterative: Illustrate the stack content in each step.
See Lecture slides

(3) In-order + iterative: write down the pseudo-code that implements the traversal.
See Lecture slides

(4) Post-order traversal: write down the sequence of nodes being visited.
12,14,9,19,23,17,67,72,54,76,50

How about post-order iterative traversal?

6. **Graphs.** Here is a map of the East Anglia towns. Edge weight indicates physical distance.



(a) Represent the graph in adjacency matrix and adjacency lists. You may list the nodes in alphabetic order.

See Lecture slides

(b) Run BFS starting from Tiptree. Assume that the Visit() procedure considers all neighbors of a given vertex in alphabetic order. In each step: mark the current vertex; mark vertices whose neighbors have all been considered; illustrate the queue Q.

Tiptree, Clacton, Feering, Harwich, Maldon, Blaxhall, Dunwich

(c) You are setting up high-speed fibers to connect all towns. Run Prim's to identify MST. In each step, highlight the current MST. Initial node: Harwich.

See lecture slides related to Prim's for detail

(d) You are traveling from Maldon to Dunwich. Run Dijkstra's to identify the shortest path.

Maldon, Tiptree, Feering, Blaxhall, Dunwich
Or: Maldon, Feering, Blaxhall, Dunwich

Total path length 72

7. **Hashtable.** Design algorithms in text (bulleted list) or pseudo-code. Describe your data structures and the time/space complexity.

(1) Given an array of integers, identify the most common integer(s).

Use hashing. Count the frequency in the hashtable with (key,frequency).
Find the key with max frequency.

Time O(n), Space O(n)

(2) Given two arrays of integers, identify integer(s) that show up in both arrays.

Store the first array in hashtable.
Do the same process for each element in the second array, if find in the hashtable, it means it is repeated.

Time O(n), Space O(n)

8. **Graph.** You are given a weighted directed graph G = (V, E, w) and the shortest path distances $\delta(s, u)$ from a source vertex s to every other vertex in G. However, you are not given $\pi(u)$ (the predecessor pointers). With this information, give an algorithm to find a shortest path from s to a given vertex t in O(V + E) time.

Solution: Start at u. Of the edges that point to u, at least one of them will come from a vertex v that satisfis $\delta(s, v)+w(v, u) = \delta(s, u)$. Such a v is on the shortest path. Recursively find the shortest path from s to v. This algorithm hits every vertex and edge at most once, for a running time of O(V + E)

9. **Applications.** Design algorithms in text (bulleted list) or pseudo-code. Describe your data structures and the time/space complexity.

(1) Numbers are randomly generated and passed to a method. Write a program to find and maintain the median value as new values are generated.
Solution: One solution is to use two priority heaps: a max heap for the values below the median, and a min heap for the values above the median. The median will be largest value of the max heap. When a new value arrives it is placed in the below heap if the value is less than or equal to the median, otherwise it is placed into the above heap. The heap sizes can be equal or the below heap has one extra. This constraint can easily be restored by shifting an element from one heap to the other. The median is available in constant time, so updates are O(lg n).

(2) You have 10 giga integers, each of which is 8 bytes. All original numbers are stored in a hard drive; your computer has a physical memory of 4 giga bytes. Design an algorithm to find the largest 1 million numbers. Why you choose the algorithms over others? (1 giga = 10^9; 1 million = 10^6)
Solution: similar to the one in midterm2. Consider mergesort; max heap; selection. Note that you should consider locality: you cannot put all 10 giga integers in to memory. Slice the entire dataset into chunks. Sort each chunk. Build a heap out of the head elements of individual sorted chunks.