

Everything listed here is covered in CS477/677.

1. Some of the algorithms we'll study this semester.

- (i) Insertion sort.
- (ii) Selection sort.
- (iii) Quicksort.
- (iv) Bubblesort.
- (v) Heapsort.
- (vi) Selection, including medians.
- (vii) Prefix codes and Huffman's algorithm.
- (viii) Dijkstra's algorithm.
- (ix) A* algorithm.
- (x) Floyd/Warshall algorithm.
- (xi) Bellman/Ford algorithm.
- (xii) Johnson's algorithm.
- (xiii) Convex sets and Graham scan.
- (xiv) Levenstein distance and Levenstein's algorithm.
- (xv) Graham scan.
- (xvi) Kosaraju's algorithm.
- (xvii) Union/Find.
- (xviii) Move-to-front. Used for an unsorted list search structure.
- (xix) Recursion.
- (xx) Lower bounds using information theory.

2. Some of the data structures we'll study this semester.

- (i) Priority queues:
 - (I) Stacks.
 - (II) Queues.
 - (III) Heaps.
- (ii) Arrays.
 - (I) k -dimensional arrays.
 - (II) Sparse arrays.
 - (III) Virtual arrays.
- (iii) Graphs and Directed graphs. This is a giant field. We only scratch the surface.
 - (I) Cliques.
 - (II) Connected.
 - (III) Strongly connected.

- (IV) Planar.
 - (V) Cyclic and acyclic.
 - (VI) Transitive closure.
 - (VII) DFS.
 - (VIII) BFS.
 - (IX) Topological order.
 - (X) Inverse of a directed graph.
- (iv) Search Structures.
- (I) Binary search trees.
 - (II) Trees. Binary search trees are a special case.
 - (III) Balanced search structures.
 - A. AVL trees.
 - B. treaps. A combination of two words: “tree” and “heap.”
 - C. 2-3 trees.
 - (IV) Tries. (How do you pronounce that word? It depends on what country you’re from.
 - (V) Unsorted lists. (No kidding!)
3. Other important stuff.
- (i) What is a recursive data structure?
 - (I) linked list.
 - (II) binary tree.
 - (ii) Complexity theory, O , Ω , and Θ .
 - (iii) Recurrences.
 - (iv) Dynamic Programming.
 - (v) Memoization. (There is no “r” in that word.)
 - (vi) Logarithms are, strictly speaking, not part of the course, since you learned them in high school (or whenever). Hopefully, you have not forgotten them, since they come up a **lot**.
4. Fill in the blanks.
- (i) $2^{\log_4(25)} = \text{_____}$ **Hint: it’s an integer.**
 - (ii) A binary tree with height h cannot have more than _____ leaves. (Exact formula, please.)
 - (iii) A planar graph G has 50 edges. G must have at least _____ vertices. (Exact answer please, no partial credit.)
 - (iv) A planar graph with $n \geq 3$ vertices can have no more than _____ edges. (Exact formula, please.)
 - (v) A spanning tree of a connected graph with 10 vertices and 14 edges has _____ edges. (Exact answer, please. No partial credit.)
 - (vi) A _____ hash function fills the hash table exactly with no collisions.

- (vii) Dijkstra's Algorithm is a modification of _____.
- (viii) Heapsort is a fast form of _____.
- (ix) Treesort is a fast form of _____.
- (x) Fill in this blank with **one** letter. If all arc weights are equal, then Dijkstra's algorithm visits the vertices in same order as _____FS.
- (xi) If G is a weighted digraph, it is impossible to solve any shortest path problem on G if G has a _____.
- (xii) If a planar graph has 10 edges, it must have at least _____ vertices. (Exact answer, please. No partial credit.)
- (xiii) If a planar graph has 20 edges, it must have at least _____ vertices. (Exact answer. No partial credit.)
- (xiv) If a _____ hash function is used, there is no collision between any two data items.
- (xv) In closed hashing, if the position at $h(x)$ is already occupied for some new data item x , a _____ sequence is used to find an unoccupied position in the hash table.
- (xvi) In _____ hashing, each item has more than one hash value, but only uses one of them.
- (xvii) In _____ hashing, there can be any number of items at a _____.
- (xviii) The (worst case) asymptotic complexity of Dijkstra's algorithm is _____, where n is the number of vertices and m is the number of arcs.
- (xix) Suppose every vertex of a graph G is given a weight, a positive value. You are allowed to select any set of vertices which does not include any two neighboring vertices. You want to find such a set of maximum total value. Is this _____?
- (xx) The A* algorithm is a modification of _____. (Your answer should use O notation.)
- (xxi) The asymptotic complexity of the Floyd/Warshall algorithm is _____.
- (xxii) The asymptotic expected time to find the median item in an unordered array of size n , using a randomized selection algorithm, is _____. (Use Θ notation.)
- (xxiii) The asymptotic time complexity of Johnson's algorithm is _____. Let n be the number of vertices and m the number of arcs. The answer should be written using O notation.
- (xxiv) The following is pseudo-code for what algorithm? _____

```

int x[n];
input values of x;
for(int i = n-1; i > 0; i--)
    for(int j = 0; j < i; j++) if(x[i] < x[j]) swap(x[i],x[j]);

```

- (xxv) The following is pseudo-code for what algorithm? _____

```

int x[n]; input values of x;
for(int i = n-1; i > 0; i--)
    for(int j = 0; j < i; j++)
        if(x[i] < x[j]) swap(x[i],x[j]);

```

(xxvi) The following is pseudo-code for what algorithm? _____

```
int x[n];
input values of x;
for(int i = n-1; i > 0; i--)
    for(int j = 0; j < i; j++)
        if(x[i] < x[j]) swap(x[i],x[j]);
```

(xxvii) The height of a binary tree with 25 nodes is at least _____. (You must give the exact answer. No partial credit.)

(xxviii) The height of a binary tree with 28 nodes is at least _____. (You must give the best possible answer, exactly. No partial credit.)

(xxix) The height of a binary tree with 45 nodes is at least _____. (You must give the exact answer. No partial credit.)

(xxx) The infix expression $-(a*b-c)$ is equivalent to the postfix expression _____.

(xxxi) The items stored in a priority queue, including stacks, queues, and heaps, represent _____.

(xxxii) The prefix expression $*+ \sim ab \sim -cd$ is equivalent to the infix expression _____ and the postfix expression _____.

(xxxiii) The subproblems of a dynamic program must be worked in _____ order.

(xxxiv) The time complexity of every comparison-based sorting algorithm is _____. (Use Ω notation.)

(xxxv) The worst case asymptotic time complexity of quicksort on a list of length n is _____.

(xxxvi) The worst case asymptotic time complexity of treesort on a list of length n is _____ if an AVL tree is used.

(xxxvii) Treesort is a form of _____.

(xxxviii) _____ algorithm does not allow the weight of any arc to be negative.

(xxxix) _____ algorithm finds an optimal binary code so that the code for one symbol is never a prefix of the code for another symbol.

(xl) _____ and _____ are divide-and-conquer sorting algorithms that we've studied this semester.

(xli) _____ and _____ are greedy algorithms that we've studied this semester.

(xlii) _____ is a fast implementation of selection sort.

(xliii) Suppose you are always having trouble finding money to paying your bills, which have various due dates. Whenever you have some money, you spend it on the bill with the earliest due date. What kind of data structure do you need to store your unpaid bills? _____

5. In each blank, write Θ if correct, otherwise write O or Ω , whichever is correct.

(i) $2^n = \text{_____} (n^2)$

(ii) $3^{n/2} = \text{_____} (2^n)$

- (iii) $\log_2 n = \text{-----} (\log_4 n)$
- (iv) $\log^2 n = \text{-----} (\log n)$
- (v) $\log_3 5 = \text{-----} (\log_4 5)$
- (vi) $\log^* \log n = \text{-----} (\log^* n)$
- (vii) $\log n^2 = \text{-----} (\log n^3)$
- (viii) $\log(n!) = \text{-----} (n \log n)$
- (ix) $n^{0.000000000001} = \text{-----} (\log n)$
- (x) $n^2 = \text{-----} (n^3)$

6. Evaluate. Hint: all answers are integers.

- (i) $\log_2 4 = \text{-----}$
- (ii) $\log_4 2 = \text{-----}$
- (iii) $\log_A A^2 = \text{-----}$, where A is any positive number larger than 1.
- (iv) $x = \text{-----}$, where $\log_x 8 = 3$.
- (v) $x = \text{-----}$, where $3 \log_x^2 9 = 2$
- (vi) $2^{\log_4 25} = \text{-----}$.
- (vii) $\log_4 0.0625 = \text{-----}$
- (viii) $x = \text{-----}$, where $\log_7 x = 0$
- (ix) $9^{\log_3 \sqrt{5}} = \text{-----}$

7. Give the asymptotic complexity, in terms of n , of each of the following code fragments.

- (i)

```
for(int i = 0; i < n; i++)
    for(int j = i; j < n; j = 2*j)
        cout << "hello world" << endl;
```
- (ii)

```
for(int i = 0; i < n; i++)
    for(int j = n; j > i; j = j/2)
        cout << "hello world" << endl;
```
- (iii)

```
for(int i = 0; i < n; i++)
    for(int j = i; j > 1; j = j/2)
        cout << "hello world" << endl;
```
- (iv)

```
for(int i = 0; i < n; i++)
    for(int j = n; j > i; j = j/2)
        cout << "hello world" << endl;
```
- (v)

```
for(int i = 0; i < n; i = i+1);
    for(int j = 0; j < i; j++);
    cout << "Hello world!" << endl;
```

```

(vi)   for(int i = 0; i < n; i = i+1);
        for(int j = i; j < n; j++);
        cout << "Hello world!" << endl;

(vii)  for(int i = 1; i*i < n; i++)
        cout << "hello world" << endl;

(viii) for(int i = 1; i < n; i++)
        for(int j = 1; j < i; j = 2*j)
        cout << "Hello world" << endl;

(ix)   for(int i = 1; i < n; i++)
        for(int j = 1; j < i; j++)
        cout << "hello world" << endl;

(x)    for(int i = 1; i < n; i++)
        for(int j = 2; j < i; j=j*j)
        cout << "Hello world" << endl;

(xi)   for(int i = 1; i < n; i++)
        for(int j = i; j > 0; j = j/2)
        cout << "hello world" << endl;

(xii)  for(int i = 1; i < n; i++)
        for(int j = i; j < n; j = 2*j)
        cout << "hello world" << endl;

(xiii) for(int i = 1; i < n; i++)
        for(int j = i; j < n; j=j+j)
        cout << "hello world" << endl;

(xiv)  for(int i = 1; i < n; i++)
        for(int j = n; j > i; j = j/2)
        cout << "hello world" << endl;

(xv)   for(int i = 1; i < n; i = i/2);
        cout << "Hello world!" << endl;

(xvi)  for(int i = 1; i < n; i = i+i);
        cout << "Hello world!" << endl;

(xvii) for(int i = 1; i < n; i=i+i;
        for(int j = 0; j < i; j++)
        cout << "hello world" << endl;

(xviii) for(int i = 1; i < n; i=i+i;
        for(int j = i; j < n; j++)
        cout << "hello world" << endl;

```

```

(xix)  for(int i = 1; i < n*n; i++)
        cout << "hello world" << endl;

(xx)   for(int i = 2; i < n; i++)
        for(int j = 1; j < log i; j++)
            cout << "hello world" << endl;

(xxi)  for(int i = 2; i < n; i=i*i)
        cout << "Hello world!" << endl;

(xxii) for(int i = n; i > 1; i=i/2;
        for(int j = 0; j < i; j++)
            cout << "hello world" << endl;

(xxiii) for(int i = n; i > 1; i=i/2;
        for(int j = i; j < n; j++)
            cout << "hello world" << endl;

(xxiv) for(int i = n; i > 2; i = sqrt(i))
        cout << "Hello world!" << endl;

```

8. Find asymptotic complexities of values computed by code, each in terms of n .

(i) The asymptotic complexity of mystery1(n) is _____

```

float mystery1(int n)
{
    float totl = 0;
    for(int i = 1; i < n; i++)
        totl += log(i);
    return totl;
}

```

(ii) The asymptotic complexity of mystery2(n) is _____.

```

float mystery2(int n)
{
    if(n <= 1) return 1;
    else return 1 + mystery2(n/2);
}

```

9. List properties of a good hash function.

10. Give the asymptotic solution to each recurrence, using Θ notation.

- (i) $F(n) = F\left(\frac{n}{2}\right) + n$
- (ii) $F(n) = 2F\left(\frac{n}{2}\right) + n$
- (iii) $F(n) = 2F\left(\frac{n}{2}\right) + n^2$

- (iv) $F(n) = 2F\left(\frac{n}{4}\right) + 1$
- (v) $F(n) = 2F\left(\frac{n}{2}\right) + n$
- (vi) $F(n) = 2F\left(\frac{n}{2}\right) + n^2$
- (vii) $F(n) = 2F\left(\frac{n}{4}\right) + \sqrt{n}$
- (viii) $F(n) = 3F\left(\frac{n}{9}\right) + 1$
- (ix) $F(n) = 3F\left(\frac{n}{3}\right) + 3F\left(\frac{2n}{3}\right) + n^2$
- (x) $F(n) = 4F\left(\frac{n}{2}\right) + n^2$
- (xi) $F(n) = 4F\left(\frac{n}{2}\right) + n$
- (xii) $F(n) = F\left(\frac{3n}{5}\right) + 4F\left(\frac{2n}{5}\right) + 1$
- (xiii) $F(n) = F\left(\frac{3n}{5}\right) + 4F\left(\frac{2n}{5}\right) + n$
- (xiv) $F(n) = F\left(\frac{3n}{5}\right) + 4F\left(\frac{2n}{5}\right) + n^2$
- (xv) $F(n) = F\left(\frac{3n}{5}\right) + 4F\left(\frac{2n}{5}\right) + n^3$
- (xvi) $F(n) = F(\log n) + 1$
- (xvii) $F(n) = F\left(\frac{n}{3}\right) + 2F\left(\frac{2n}{3}\right) + n$
- (xviii) $F(n) = F(n - \log n) + \log n$
- (xix) $F(n) = F\left(\frac{n}{2}\right) + 2F\left(\frac{n}{4}\right) + n$
- (xx) $F(n) = F\left(\frac{n}{2}\right) + n$
- (xxi) $F(n) = F(\sqrt{n}) + 1$
- (xxii) $F(n) = F(n - 1) + n^2$
- (xxiii) $F(n) = F(n - \sqrt{n}) + n$
- (xxiv) $F(n) = F(n - \log n) + \log^2 n$

11. Explain how to compute the n^{th} Fibonacci number $F(n)$ in $O(\log n)$ time using the recurrence

$$F(1) = F(2) = 1$$

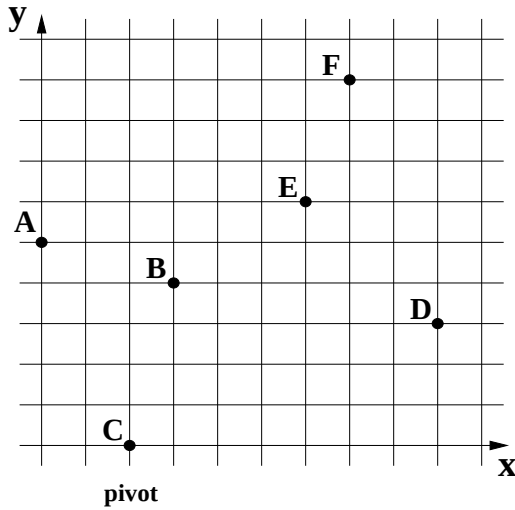
$$F(n) = F\left(\frac{n-1}{2}\right) * F\left(\frac{n}{2}\right) + F\left(\frac{n+1}{2}\right) * F\left(\frac{n+2}{2}\right) \text{ for } n > 2$$

where, for an integer n , the expression $n/2$ is always rounded down to an integer, as in C++. Be sure to state the asymptotic space complexity of your algorithm.

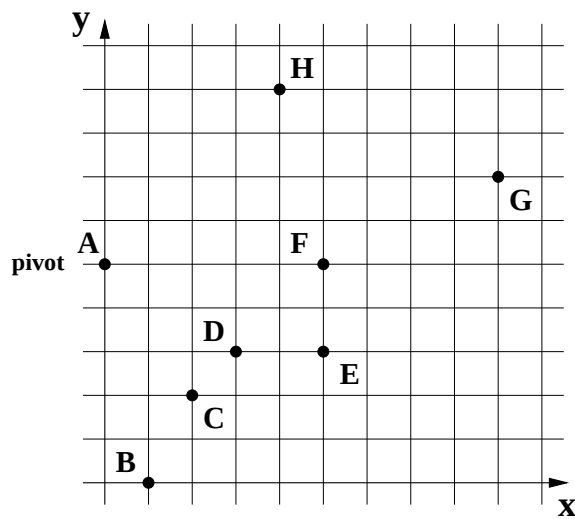
12. Find the maximum length strictly monotone subsequence of 4, 9, 2, 7, 3, 6, 5. Show your work.

13. Find the convex hull of each of the sets of points in the plane shown in each of these examples. Show your steps. To delete anything, do not erase, but simply cross it out.

(i)

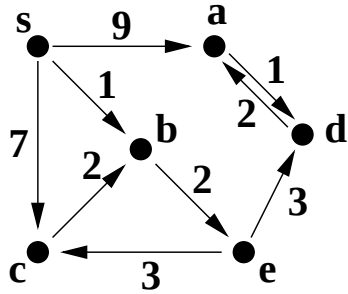


(ii)

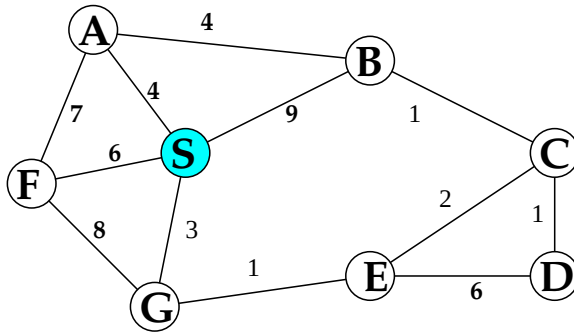


14. Use Dijkstra's algorithm to solve the single source shortest path problem for each of the following weighted directed graphs, where s is the source. Show weights, backpointers, and minheap at each step. Do not erase anything; simply cross it out.

(i)

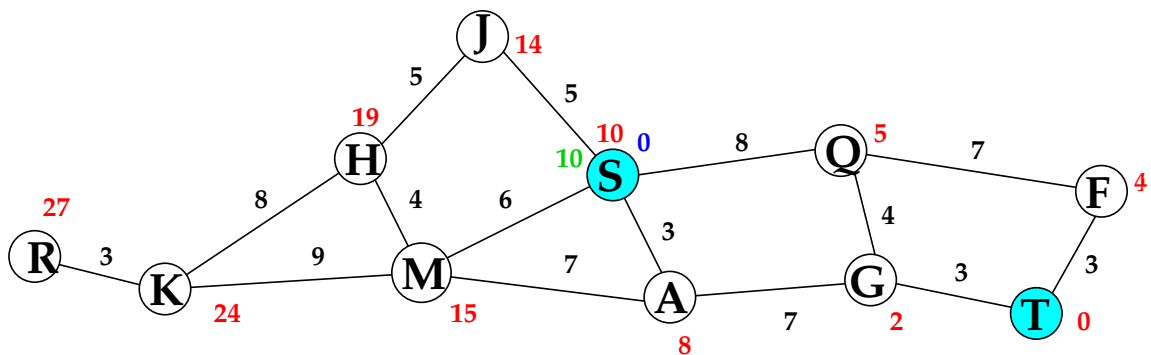


(ii)

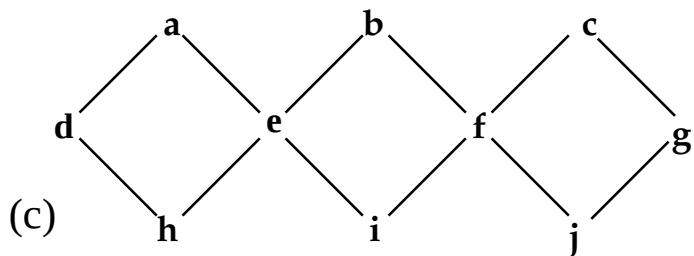
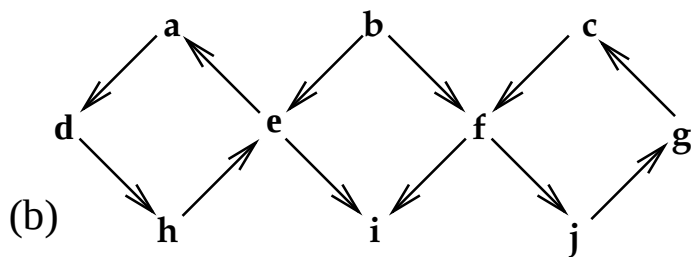
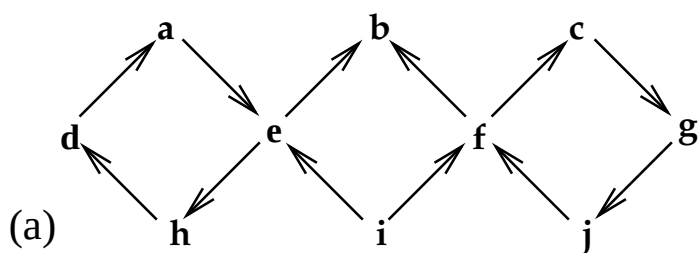


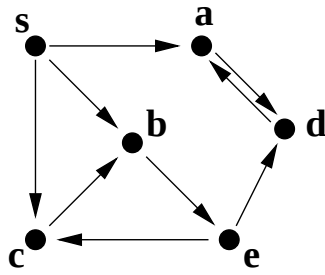
15. Sketch a circular linked list with dummy item which implements a queue. The queue has four items. From front to rear, these are R, Q, T, L, and show the insertion of A into the queue. Show the steps. Don't erase deleted links; instead, simply cross them out.

16. Walk through the A^* algorithm shortest path from S to T in the weighted graph shown below. The heuristics are indicated by red numerals.

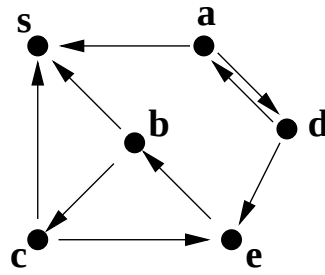


17. Use the DFS (Kosaraju's) method to find the strong components of each of the directed graphs below. Show your steps. In each case, the original digraph is (a). Use the remaining figures for your work.

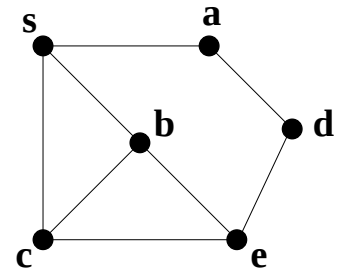




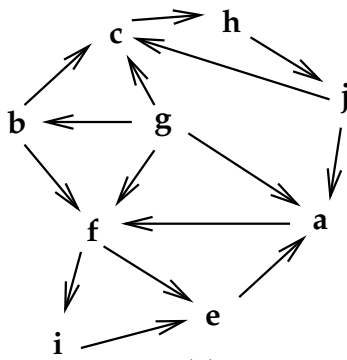
(a)



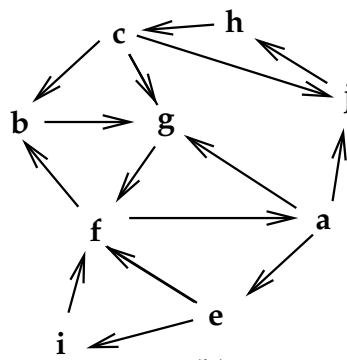
(b)



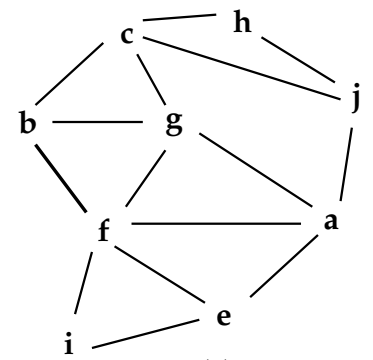
(c)



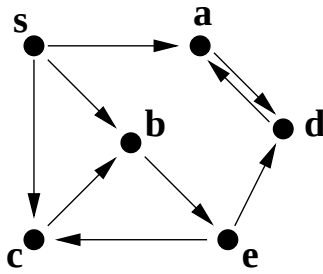
(a)



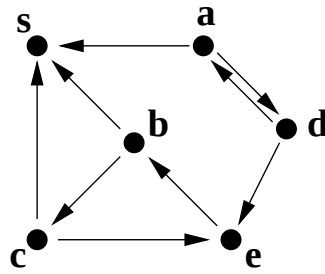
(b)



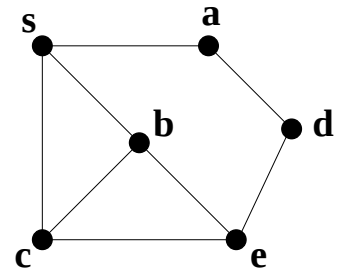
(c)



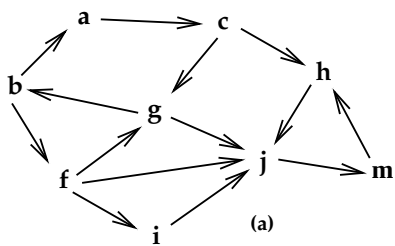
(a)



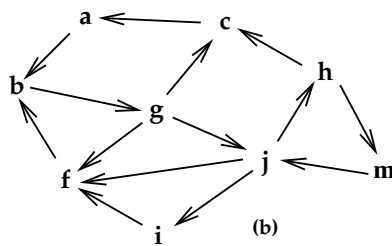
(b)



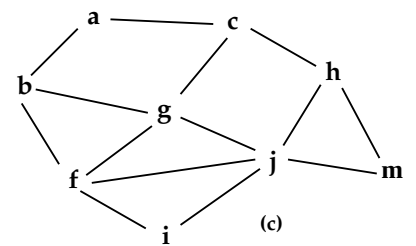
(c)



(a)



(b)



(c)

18. Dynamic Programming.

(i) ' Given an array $A[N]$ the values of an array $B[N]$ are defined as follows:

$$B[0] = A[0]$$

$$B[1] = A[1]$$

For any $1 < i < N$, $B[i] = \max(B[i-1], B[i-2] + A[i])$

Finish the C++ program.

```
// header

const int N = 20;
int A[N];
int B[N];

void getA()
{
    for(int i = 0; i < N; i++)
        cin >> A[i];
}

void computeB() // compute the values of B
{

}

void writeB()
{

}

int main()
{
    getA();
    computeB();
    writeB();
    return 1;
}
```

- (ii) You are given a row of coins of values, $x_1, x_2, \dots, x_n$. Give a dynamic program which computes the maximum total value of a set of coins no two of which are adjacent in the original row. Your program does not have to print the optimal set.

- (iii) You are given an acyclic directed graph $G = (V, E)$. Describe a dynamic programming algorithm which calculates the maximum number of edges of any directed path through G . Let $\text{Innbr}(x)$ be the set of in-neighbors of a vertex x , and $\text{Outnbr}(x)$ its set of out-neighbors.

19. Use Huffman's algorithm to compute an optimal prefix code for each alphabet, given the relative frequency of each symbol.

(i)

<i>a</i>	3	
<i>b</i>	6	
<i>c</i>	4	
<i>d</i>	5	
<i>e</i>	8	
<i>f</i>	9	

(ii)

<i>a</i>	7	
<i>b</i>	3	
<i>c</i>	5	
<i>d</i>	8	
<i>e</i>	12	
<i>f</i>	1	

(iii)

<i>a</i>	8	
<i>b</i>	5	
<i>c</i>	3	
<i>d</i>	7	
<i>e</i>	12	
<i>f</i>	1	

20. Write pseudocode for the Bellman/Ford algorithm.
21. Write pseudocode for the Floyd/Warshall algorithm.
22. Write pseudocode for a function

float power(float x, int n)

that returns x^n . You may assume that $x \neq 0$ and $n \geq 0$. It is not necessary to use an algorithm given in class; use any $O(\log n)$ time algorithm.

23. Walk through mergesort, given the array below.

XORWYFHBVESUP

24. Walk through polyphase mergesort, given the array below.

ACPREHJMQSOZTUNXVB

25. True, False, or Open, meaning that the answer is not known to science at this time.

- (i) ----- A complete graph of order n is planar.
- (ii) ----- A dynamic program with n subproblems, each of which can be worked in constant time, can always be worked in polylogarithmic time by dividing up the task among polynomially many processors working in parallel.
- (iii) ----- Closed hashing uses open addressing.
- (iv) ----- A good programmer would never use an unordered list as a search structure.
- (v) ----- Computers are so fast nowadays that there is no longer any point to analyzing the time complexity of a program.
- (vi) ----- Kruskal's algorithm uses dynamic programming.
- (vii) ----- No good programmer would ever implement a search structure as an unordered list.
- (viii) ----- Open hashing uses open addressing.
- (ix) ----- There is a mathematical statement which is true, yet cannot be proven.
- (x) ----- The subproblems of a dynamic program form a directed acyclic graph.
- (xi) ----- Treesort is an efficient implementation of insertion sort.

26. Levenshtein distance between strings.

- (i) Compute the Levenshtein distance between *abcdba* and *acdbce*. Show the matrix.
- (ii) Compute the Levenshtein distance between *abcdafg* and *agbccdfc*. Show the matrix.
- (iii) Compute the Levenshtein distance between *bdef* and *abce*. Show the matrix.

27. Memoization

- (i) Consider the following C++ code.

```
int martha(int n)
{
    if(n <= 1) return 1;
    else return martha(n/2)+martha((n+1)/2)+n;
}
```

(I) What is the asymptotic complexity of *martha(n)*? **Hint: Bently-Blostein-Saxe**

(II) What is the time complexity of the recursive code given above? (This is **not** the same question.) Assume that each arithmetic operation takes one time unit.

(III) Using memoization, the time complexity of the computation can be greatly decreased. What is that time complexity?

(ii) Consider the following C++ code.

```
int george(int n)
{
    if(n <= 1) return 1;
    else return george(n/2) + george(n/3) + george(n/6);
}
```

(I) What is the asymptotic complexity of `george(n)`? **Hint: Akra-Bazzi**

(II) What is the time complexity of the recursive code given above? (This is **not** the same question.) Assume that each arithmetic operation takes one time unit.

(III) Using memoization, the time complexity of the computation can be greatly decreased. What is that time complexity?

28. A compiler stores an array $A[9][5][12]$ into main memory in row major order, with base address 1024. Each entry of A requires one unit of memory. What is the address of $A[5][3][7]$ in main memory?

29. You need to store Pascal's triangle in row-major order into a 1-dimensional array P whose indices start at 0. The triangle is infinite, but you will only store $\binom{n}{k}$ for $n < N$. Write a function I such that $P[I(n, k)] = \binom{n}{k}$ for $0 \leq k \leq n < N$. For example, $I(3, 2) = 8$.

```

      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
```

```
int I(int n, int k)
{
    // the position of n choose k in the linear array
    assert(k >= 0 and n >= k and n < N);
    int indx =

    return indx;
}
```

30. Implementing a stack.

(i) Consider an array implementation of a stack of integers, as given below. Fill in the code which implements the needed operators of a stack.

```

const int N = // whatever
struct stack
{
    int item[N];
    int size; // number of items in the stack
    // bottom of the stack is at item[0];
};

void initialize(stack&s)
{

}

void push(stack&s,int i)
{

}

bool empty(stack&s)
{

}

int pop(stack&s)
{

}

```

- (ii) A stack can also be implemented as a linked list: Instead of being limited by the value of N, such a stack is theoretically any size, not chosen in advance.

```

struct stacknode;
typedef stacknode*stack;
struct stacknode
{
    int item;
    stack link;
};

void initialize(s&stack)
{

}

```

```

void push(s&stack,int i)
{

}

bool empty(s&stack)
{

}

int pop(s&stack)
{

}

```

Do you see any errors in the above code? If so, fix them.

31. Here is an implementation of binary search tree of integer. Fill in the missing code in the code for the operators.

```

struct treenode;
typedef treenode*tree;
struct treenode
{
    int item;
    tree left;
    tree right;
};

void insert(tree&root,int n)
    // inserts n into the binary search tree. Duplicates are not inserted.
{
    if(root)
        // missing code
    else; // do not insert duplicate
    else
    {
        root = new treenode;
        root->item = n;
    }
}

bool find(tree root, int n)
    // returns true if some node contains n, false otherwise.

```

```

{
    if(root)
        if(root->item == n) return true;
        else if(n < root->item) return find(root->left,n);
        else return // missing code here
    else return false;
}

tree location(tree root, int n)
    // returns pointer to the node which contains n, if any, nullptr otherwise.
{
    if(root)
        if(root->item == n) return root;
        else if(n < root->item) return location(root->left,n);
        else return // missing code here
    else return nullptr;
}

```

Do you see any errors in the above code? If so, fix them.