University of Nevada, Las Vegas Computer Science 477/677 Spring 2026

Answers to Study for Final Examination May 11, 2026

Apology: I have tried to eliminate duplicate questions, but there are probably some I missed.

1. omegatheta1 In each blank, write Θ if correct, otherwise write O or Ω, whichever is correct.

   (i) $n^2 = O(n^3)$

   (ii) $\log(n^2) = \Theta\ (\log(n^3))$

   (iii) $\log(n!) = \Theta\ (n \log n)$

   (iv) $\log_2 n = \Theta\ (\log_4 n)$

   (v) $n^{0.000000000001} = \Omega\ (\log n)$

   (vi) $\log^* \log n = \Theta\ (\log^* n)$

2. True or False. Write "O" if the answer is not known to science at this time.

   (i) **F** No good programmar would ever implement a search structure as an unordered list.

   (ii) **T** There is a mathematical statement which is true, yet cannot be proven.

   (iii) **T** The subproblems of a dynamic program form a directed acyclic graph.

   (iv) **F** Kruskal's algorithm uses dynamic programming.

   (v) **T** Heapsort can be considered to be an efficient implementation of selection sort.

   (vi) **F** Computers are so fast nowadays that there is no longer any point to analyzing the time complexity of a program.

   (vii) **T** A complete graph of order 4 is planar.

   (viii) **T** Binary tree sort (also called "treesort") can be considered to be a sophisticated implementation of insertion sort.

   (ix) **F** Open hashing uses probe sequences.

   (x) **F** You can avoid collisions in a hash table by making the table twice as large as the data set.

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

   (i)
   ```
   for(i = 0; i < n; i = i+1)
   ```
   $\Theta(n)$

   (ii)
   ```
   for(int i = 0; i < n; i++)
    for(int j = i; j < n; j = j*j)
   ```
   $\Theta(n)$

   (iii)
   ```
   for(int i = n; i > 1; i--)
    for(int j = 1; j < i; j = 2*j)
   ```
   $\Theta(n \log n)$

   (iv)
   ```
   for(int i = 0; i < n; i++)
    for(int j = i; j > 0; j = j/2)
   ```
   $\Theta(n \log n)$

(v)
```
for(int i = 0; i < n; i++)
 for(int j = n; j > i; j = j/2)
```
$\Theta(n)$

(vi)
```
for(int i = 1; i*i < n; i++)
```
$\Theta(n^2)$

(vii) Duplicate

(viii) Duplicate

(ix)
```
for(int i = 1; i < n; i = i+i)
```
$\Theta(\log n)$

(x)

(xi)
```
for(int i = 2; i < n; i = i*i)
```
$\Theta(\log \log n)$

(xii) Duplicate

(xiii)
```
for(int i = n; i > 2; i = sqrt(i))
```
$\Theta(\log \log n)$

(xiv)
```
for(int i = 1; i < n; i++)
 for(int j = 1; j < i; j++)
$\Theta(n\log n)$
```

(xv)
```
for(int i = 1; i < n; i++)
 for(int j = i; j < n; j++)
```
$\Theta(n)$

(xvi)
```
for(int i = 1; i < n; i = 2*i)
  for(int j = 2; j < i; j = j*j)
```
$\Theta(n \log \log n)$

4. Solve the recurrences. Give the asymptotic value of $F(n)$ in terms of n, using Θ notation.

(i) $F(n) = 2F(3n/4) + F(n/2) + 2F(n/4) + 2n^3$
$2(3/4)^3 + 2(1/4)^3 + 2(1/4)^3 = 1$
$F(n) = \Theta(n^3 \log n)$

(ii) $F(n) = 2F(n/2) + n^2$
$F(n) = \Theta(n^2)$

(iii) $F(n) = 2F(n/2) + n$
$F(n) = \Theta(n^2 \log n)$

(iv) $F(n) = 2F(n/4) + \sqrt{n}$
$2(1/4)^{\frac{1}{2}} = 1$
$F(n) = \Theta(\sqrt{n} \log n$

(v) $F(n) = 3F(n-1) + 1$
$F(n) = \Theta(3^n)$

(vi) $F(n) = 3F(n/2) + n^2$

$3(n/2)^2 < 1$

$F(n) = \Theta(n^2)$

(vii) Duplicate

(viii) $F(n) = 3F(n/9) + 1$

$\delta = \frac{1}{2}$

$F(n) = \Theta(\sqrt{n})$

(ix) $F(n) = 4F(n/2) + n$

$F(n) = \Theta(n^2)$

(x) $F(n) = 4F(n/2) + n^2$

$F(n) = \Theta(n^2 \log n)$

(xi) $F(n) = 4F(n/2) + n$

$F(n) = \Theta(n^2)$

(xii) $F(n) = F(3n/5) + 4F(2n/5) + n^2$

$(3/5)^2 + 4(2/5)^2 = 1$

$F(n) = \Theta(n^2 \log n)$

(xiii) $F(n) = F(\log n) + 1$

$F(n) = \Theta(\log^*(n))$

(xiv) $F(n) = F(n-1) + n^2$

$$\frac{F(n) - F(n-1)}{1} = n^2$$

$F'(n) = n^2$

$F(n) = \Theta(n^3)$

(xv) $F(n) = F(n/2) + 2F(n/4) + n$

$F(n) = \Theta(n \log n)$

(xvi) $F(n) = F(n/2) + F((n-1)/2) + 3n$

$F(n) = \Theta(n \log n)$

(xvii) $F(n) = F(n/3) + F(n/2) + n$

$F(n) = \Theta(n)$

(xviii) $F(n) = F(n/5) + F(7n/10) + n$

$F(n) = \Theta(n)$

(xix) Duplicate

(xx) $F(n) = F\left(\frac{n}{2}\right) + n$

$F(n) = \Theta(n)$

(xxi) $F(n) = F(n/2) + 2F(n/4) + n$

$F(n) = \Theta(n \log n)$

(xxii) $F(n) = F(\sqrt{n}) + 1$

$F(n) = \Theta(\log \log n)$

5. Fill in the blanks.

(i) **heapsort** is a fast implementation of selection sort.

(ii) **treesort** is a fast implementation of insertion sort.

(iii) The asymptotic complexity of the Floyd/Warshall algorithm for a directed graph of size n with m arcs is $\Theta(n^3)$.

(iv) **Huffman's** algorithm finds a binary code for a weighted alphabet such that the code for one symbol is never a prefix of the code for another symbol.

(v) The asymptotic expected height of a treap with n nodes is $\Theta(\log n)$.

(vi) If G is a weighted digraph, it is impossible to solve any shortest path problem on G if G has a **negative cycle**

(vii) The following is pseudo-code for what algorithm? **selection sort**

```
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]);
```

(viii) **Dijkstra's** algorithm does not allow the weight of any arc to be negative.

(ix) The asymptotic time complexity of Johnson's algorithm on a weighted directed graph of n vertices and m arcs is $O(nm \log n)$.
(Your answer should use O notation.)

(x)

(xi) 10 The prefix expression $*a+ \sim b*-c\,d \sim e$ is equivalent to the infix expression $a*(\sim b+(c-d)* \sim e)$ and the postfix expression $a \sim bcd- \sim e*+*$

(xii) In closed hashing, if the position at $h(x)$ is already occupied for some data item x, a **probe** sequence is used to find an unoccupied position in the hash table.

(xiii) A planar graph with $n \geq 3$ vertices can have no more than $3n - 6$ edges. (Exact formula, please.)

(xiv) The height of a binary tree with 17 nodes is at least **4**. (You must give the best possible answer, exactly. No partial credit.)

(xv) The following is pseudo-code for what algorithm? **bubblesort**

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

(xvi) The items stored in a priority queue (that includes stacks, queues, and heaps) represent **unfulfilled obligations**.

(xvii) The asymptotic complexity of Dijkstra's algorithm algorithm is $O(m \log n)$.

(xviii) **Huffman's** and **Kruskal's** are greedy algorithms that we've studied this semester.

(xix) An acyclic directed graph with 9 vertices must have at least **9** strong components. (Must be exact answer.)

(xx) In **Open hashing** there can be any number of items at a given index of the hash table.

(xxi) If a planar graph has 10 edges, it must have at least **6** vertices.

(xxii) Fill in this blank with one letter. If all arc weights are equal, then Dijkstra's algorithm visits the vertices in same order as **BFS**.

(xxiii) The height of a binary tree with 45 nodes is at least **5**. (You must give the exact answer. No partial credit.)

(xxiv) The following is pseudo-code for what algorithm? **selection sort**, building from the high end.

```
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) A planar graph with $n \geq 3$ vertices can have no more than $3n - 6$ edges. (Exact formula, please.)

(xxvi) Duplicate

(xxvii) A binary tree with 8 nodes cannot have height less than **3**. (Exact answer.)

(xxviii) **binary search** is a divide-and-conquer searching algorithms.

(xxix) **quicksort** and **merge sort** are divide-and-conquer sorting algorithms.

(xxx) **perfect** hashing, which can be used by a compiler to identify reserved words, does not have collisions.

(xxxi) In an open hash table, there is a **search structure** at each table index.

(xxxii) A directed graph is defined to be **strongly connected** if, given any two vertices x and y, the graph contains a path from x to y.

(xxxiii) In order for there to exist a topological order of the vertices of a digraph, the graph must be **acyclic**.

(xxxiv) Duplicate

(xxxv) Duplicate

(xxxvi) A **perfect** hash function has no collisions.

(xxxvii) Duplicate

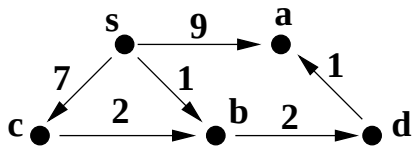
(xxxviii) Duplicate

(xxxix) The asymptotic expected time to find the median item in an unordered array of size n, using a randomized selection algorithm, is $\Theta(n)$

(xl) Duplicate

(xli) Duplicate

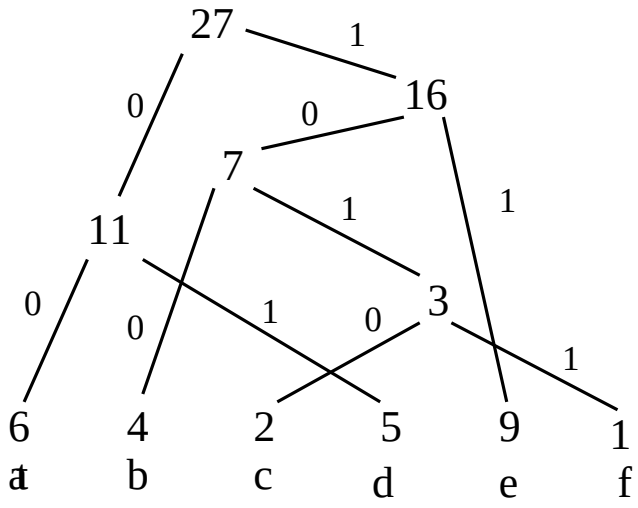
6. Use Dijkstra's algorithm to solve the single source shortest path problem for the following weighted directed graph, where **s** is the source. Show the steps.



	s	a	b	c	d
V	0	~~9~~ 4	1	7	3
back		~~s~~ d	s	s	b

7. Find an optimal prefix code for the alphabet $\{a, b, c, d, e, f\}$ whose frequencies are given:

a	6	00
b	4	100
c	2	1010
d	5	01
e	9	11
f	1	1011



8. What is the asymptotic complexity of the function **george**(n), in terms of n?

```
int george(int n)
 {
  // input condition: n >= 0
  if(n < 1) return 1;
  else return george(n-1)+george(n-1);
 }
```

$\Theta(2^n)$

9. What is the asymptotic complexity of the function **martha**(n) in terms of n?

```
int martha(int n)
 {
  // input condition: n >= 1
  if(n == 1) return 0;
  else return n + martha(n/2);
 }
```

$\Theta(n)$

10. What is the asymptotic time complexity of above code which computes **martha**(n), in terms of n?

    Hint: This is **not** the same question as the previous one!

    $\Theta(\log n)$

11. Write a C++ function which solves the simplest coinrow problem we have discussed, namely, given a sequence of positive numbers, find the subsequence of maximum total, subject to the condition that the subsequence may not contain any two consecutive terms of the sequence.

    Let $x_1, x_2, \ldots x_n$. Let $A[n]$ be the greatest total of any legal subsequence of the first n terms of the sequence, and let $B[n]$ be the greatest total of any legal subsequence

    There is more than one correct algorithm. Here is one.

    $A[0] = 0$
    $B[1] = x_1$
    $A[1] = \max(0, B[1])$
    For all i from 2 to n
      $B[i] = x_i + A[i-2]$
      $A[i] = max(B[i], A[i-1])$
    Return $A[n]$

12. The usual recurrence for Fibonacci numbers is:
    $F[1] = F[2] = 1$
    $F[n] = F[n-1] + F[n-2]$ for $n > 2$

    However, there is another recurrence:
    $F[1] = F[2] = 1$
    $F[n] = F\left[\frac{n-1}{2}\right] * F\left[\frac{n}{2}\right] + F\left[\frac{n+1}{2}\right] * F\left[\frac{n+2}{2}\right]$ for $n > 2$
    where integer division is truncated as in C++.

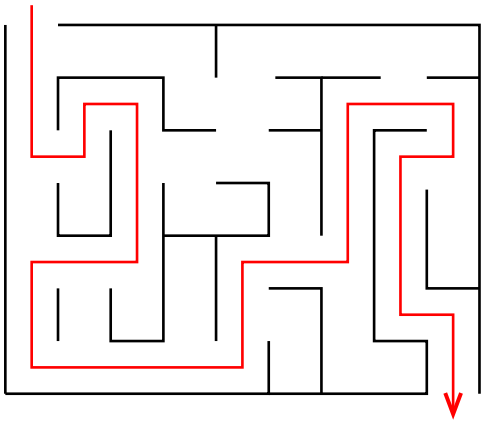
    Using that recurrence, Describe a $\Theta(\log n)$-time memoization algorithm which reads a value of n and computes $F[n]$, but computes only $O(\log n)$ intermediate values.

```
int F(int n)
 {
  assert(n > 0);
  int fn;
  if(the memo (n,F) has been saved) fn = F;
  else
   if(n < 3) fn = 1;
   else fn = F((n-1)/2)+F(n/2)+F((n+1)/2)+F((n+1)/2);
  store the memo (n,fn);
  return fn;
 }
```

    Values of $F(m)$ are stored in layers; each layer saves at most four values of $F(m)$ for m close to $\frac{n}{2^k}$. Thus the number of memos stored during an execution of $F(n)$ is $O(\log n)$.

13. The figure below shows an example maze. The black lines are walls. You need to find the shortest path, avoiding the walls, from the entrance at the upper left and the exit at the lower right. The red path shows one such path, although it is not the shortest. Describe a program to find the shortest path from the entrance of such a maze, not necessarily this one, to the exit. You do not need to write pseudocode. Your answer should contain the word, "graph," and should state which search method and which data structure(s) you need to use.

    Define a graph $G = (V, E)$ where V is the set of all squares in the figure, and E is the set all pairs of adjacent squares which are not separated by a line. We need to find the shortest path in the graph from the upper left square to the lower right square. The most efficient method is BFS. The red path in the figure is not optimal.



14. 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 $0 \le k \le n < N$. For example, $I(3,2) = 8$.
    $I(n,k)$ is the number of predecessors of the entry whose value is $\binom{n}{k}$

```
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 = n(n-1)/2 + k;
  return indx;
 }
```

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

15. A compiler stores an array `A[8][10][18]` into main memory in row major order, with base address B, and each entry of A requires one place in main memory. Write a formula for the main memory address of `A[i][j][k]` for integers i, j, and k within range.

    $B + 180 * i + 18 * j + k$

16. A 3-dimensional $8 \times 9 \times 6$ rectangular array X is stored in main memory in column major order, and its base address is 4096. Each item of X takes two words of main memory, that is, two address location. Find the address, in main memory, of X[3][7][4].

    The number of predecessors of $X[3][7][4]$ is $4 * 8 * 9 + 7 * 8 + 3 = 347$. The address of that entry in main memory is $2 * 347 + 4096 = 4590$,

17. You are implementing a 3D triangular array A where $A[i][j][k]$ is defined for $0 \leq k \leq j \leq i \leq 4$, a total of 35 entries (Is that correct?), and is stored as a one-dimensional subarray of main memory in row-major order, with base address 1024. Each term of A takes one place in main memory. What would be the address, in main memory, of $A[4][2][1]$?

    The list of items in main memory, up to 421, is:
    000,100,110,111,200,210,211,220,221,222,300,310,311,320,321,322,330,331,332,333 400,410,411,420,421
    The number of predecessors of $A[4][2][1]$ is (1+3+6+10) + (1+2) + 1 = 24.
    The address in main memory of $A[4][2][1]$ is $24 + 1024 = 1048$

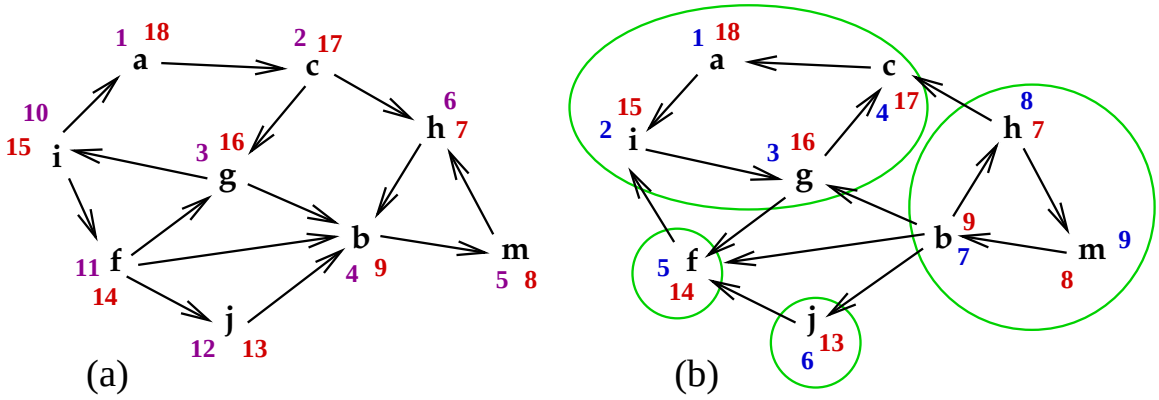
18. You are given an acyclic directed graph $G = (V, E)$ where each arc is weighted. If (x, y) is an arc, we write $w(x, y)$ for the weight of that arc. Describe a dynamic programming algorithm which calculates the directed path through G of maximum weight.

    Hint: Subproblem: given a vertex x, what is the maximum weight $A[x]$? of any directed path that ends at x? We compute all values of $A[x]$ in topological order.

    Assume that $V = \{1, 2, \ldots n\}$, in topological order, and that $In(x)$ is the set of in-neighbors of a vertex x. Here is an algorithm (which is not the only correct algorithm for the problem.)

    For $x = 1$ to n, $A[x] = \max\{A[y] + w(y, x) : y \in In(x)\}$
    Return $\max\{A[i]\}$

19. Use the DFS method to find the strong components of the digraph shown below as (a). Use the other figures to show your steps.



20. List properties of a good hash function. There are many ways to express this Here are some properties you could list.

    (a) deterministic.

    (b) fast to compute;

    (c) avoid locality.

    (d) evenly spead over hash values.

21. Walk through mergesort with the array given below.

    VJATNLDQMEFSPWGL

    VJATNLDQ MEFSPWGL

    VJAT NLDQ MEFS PWGL

    VJ AT NL DQ ME FS PW GL

    JV AT LN DQ EM FS PW GL

    AJTV DLNQ EFMS GLPW

    ADJLNQTV EFGLMPSW

    ADEFGJLLMNPSTVW

22. Write pseudcode for the simple coin-row problem we discussed in class. You are given a row of n coins of various non-negative values. The problem is to select a set of coins of maximum total value, subject to the condition that no two adjacent coins are selected. ~~Your code should allow recovery of the selected set of coins which are selected.~~

    Let v_i be the value of the i^{th} coin.

    There are several choices of variables. Here is one.

    Let $A[i]$ be the maximum value of any legal subsequence of the first i coins. Let $B[i]$ be the maximum value of any legal subsequence which includes the i^{th} coin.

    $A[0] = 0$
    $B[1] = x_1$
    $A[1] = B[1]$
    For i from 2 to n
        $B[i] = x_i + A[i-2]$
        $A[i] = \max(B[i], A[i-1])$
    Return $A[n]$

    $A[0] = 0$

23. Write pseudcode for the variation of the coin-row problem where you are given a row of n coins of various values, and you must select a set of coins of maximum total value, subject to the condition that no **three** adjacent coins are selected. ~~Your code should allow recovery of the selected set of coins.~~

    Let v_i be the value of the i^{th} coin. Let $A[i]$ be the maximum combined value of any legal subset of the first i coins, and let $B[i]$ be the maximum value of any legal subset which does not contain the $(i-1)^{\text{st}}$ coin.

    There are many ways to answer this question with dynamic programming. Here is one.

```
A[0] = 0
A[1] = v_i
B[2] = v_2
A[2] = v_1+v_2
for i = 3 to n
  B[i] = v_i + A[i-2]
  A[i] = max(B[i],A[i-1],v_i+B[i-1])
Return A[n]
```

24. 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 the algorithm given in class; use any $O(\log n)$ time algorithm.

    Here is a simple recursive algorithm.

```
float power(float x, int n)
 if(n = 0) return 1;
 else if (n = 1) return $x$;
 else if (n%2)
```

```
 {
  float y = power(x,n/2);
  return y*y;
 }
else return x*power(x,n-1);
```

25. Walk through polyphase mergesort with the array given below.

AC BX FR EY GMQS N DZ

ACFR GMQS DZ

BX EY N

ABCFRX DNZ

EGMQSY

ABCEFGMQRSXY

DNZ

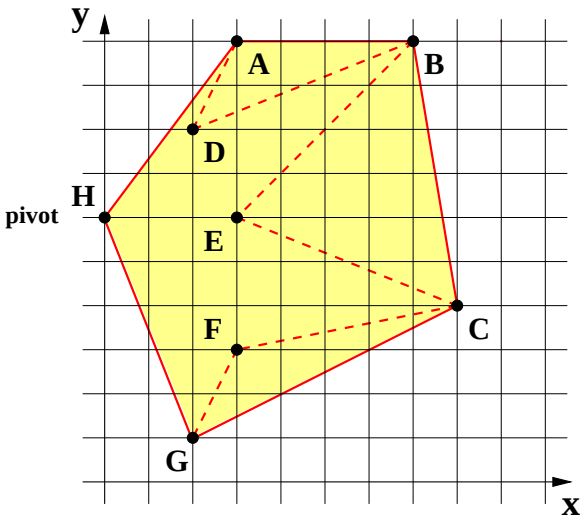
ABCDEFGMNQRSXYZ

26. What is the loop invariant of the loop in the following function?

```
float product(float x, int n)
 {
  // assert(n >= 0);
  float z = 0.0;
  float y = x;
  int m = n;
  while(m > 0)
   {
    if(m%2) z = z+y;
    m = m/2;
    y = y+y;
   }
  return z;
 }
```

$x^n = z + y * m$

27. Using the algorithm Graham Scan, find the convex hull of the set of points indicated in the figure below. Show your steps.



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

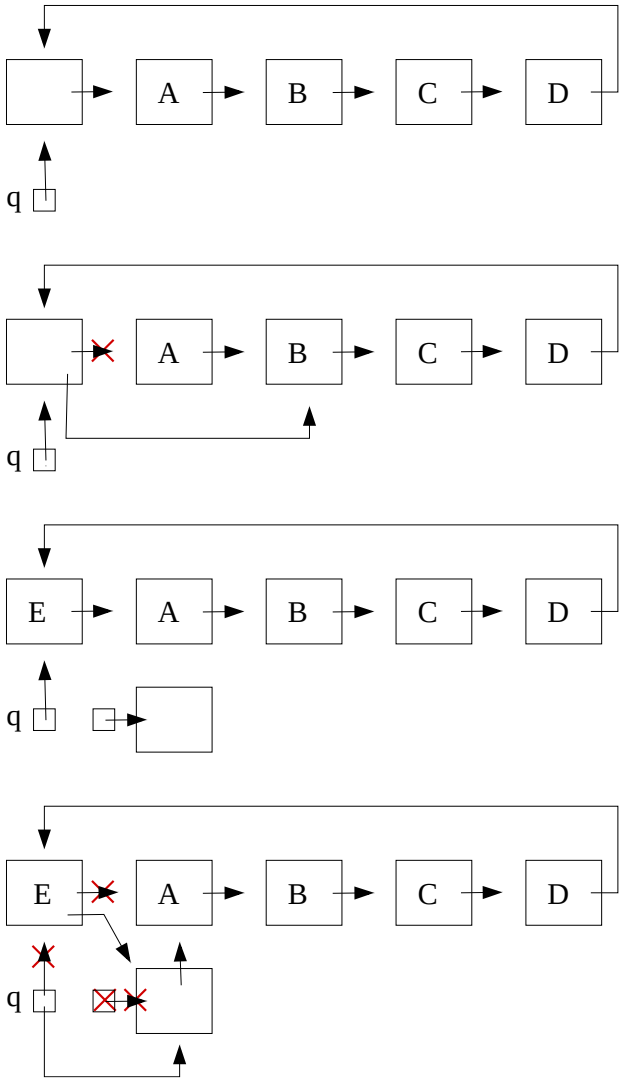
```
const int N = // whatever
struct stack
 {
  int item[N];
  int size; // number of items in the stack
  // bottom of the stack is at item[0];
 };
void initialize(s&stack)
 {
  s.size = 0;
 }
void push(s&stack,int i)
 {
  s.item[s.size++] = i;
 }
 bool empty{s&stack)
  {
   return s.size == 0;
  }
 int pop(s&stack)
  {
   assert(s.size > 0)
   return(s.item[s.--size];
  }
```

29. Compute the Levenstein distance between abcdafg and agbccdfc. Show the matrix.

		a	g	b	c	c	d	f	c
	0	1	2	3	4	5	6	7	8
a	1	0	1	2	3	4	5	6	7
b	2	1	1	1	2	3	4	5	6
c	3	2	2	2	1	2	3	4	5
d	4	3	3	3	2	2	2	3	4
a	5	4	4	4	3	3	3	3	4
f	6	5	5	5	4	4	4	3	4
g	7	6	5	6	5	5	5	4	4

The Levenstein distance is 4.

30. Sketch a circular linked list with dummy node which implements a queue. The queue has four items. From front to rear, these are A, B, C, D, and show the insertion of E into the queue. Show the steps. Don't erase deleted objects; instead, simply cross them out.



31. Write pseudocode for the Floyd/Warshall algorithm.

Variables:
Vertices are $\{1, 2, \ldots n\}$
$W(x, y)$ = weight of arc (x,y), ∞ if no arc.
$V(x, y)$ = min length of path from x to y.

For all x, y
    $V(x, y) = W(x, y)$
    back$(y) = x$
For all j
    For all i
        For all k
            temp $= V(i, j) + V(j, k)$
            if (temp $< V(i, k)$)
                $V(i, k) =$ temp
                back$(k) =$ back(j)

32. The following code is used as a subroutine for both quicksort and select. Assume A[n] is an array of integers. For simplicity, we assume that no two entries of A are equal. Write a loop invariant for the while loop.

```
int pivot = A[0];
int lo = 0;
int hi = n-1;
while(lo < hi)
 {
  if(A[lo+1] < pivot) lo++;
  else if(A[hi] > pivot) hi--;
  else swap(A[lo+1],A[hi]);
 }
```

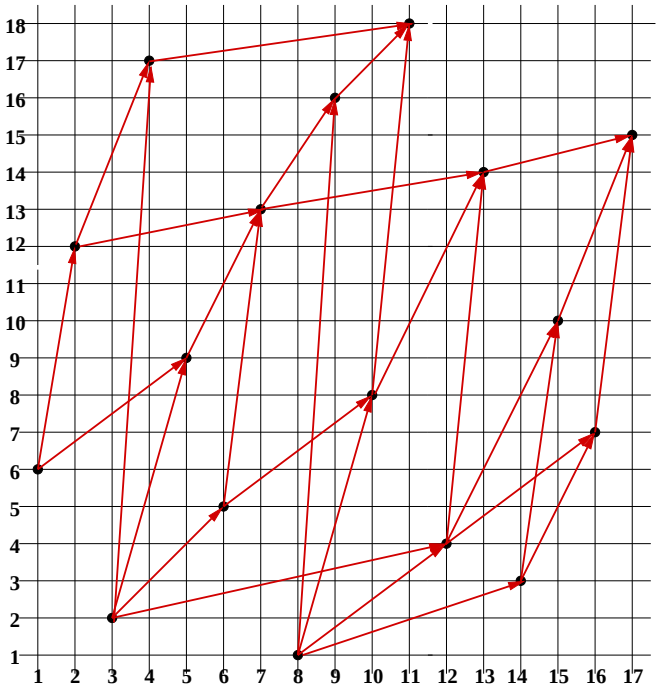
The loop invariant:

A[i] < pivot for all $0 < i \leq lo$ and A[j] > pivot for all hi $\leq$ j < n.

33. ~~Find the longest monotone subsequence of the sequence:~~

~~6,12,1,17,9,5,13,1,6,8,18,4,3,10,7,15~~

There is a simple reduction of the lms problem to the largest weight directed path problem given in Problem 18. This sequence then reduces to the following graph.



34. Write pseudo-code for the Bellman-Ford algorithm. Assume that the the vertices are $\{0, 1, 2, \ldots n\}$ the source is 0, and the arcs are $\{(x_1, y_1), (x_2, y_2), \ldots (x_m, y_m)\}$, and each arc $x_k, y_k)$ has weight W_k. Be sure to include the shortcut that ends the program when the final values have been found.

```
V[0] = 0
for i = 1 to n
  V[i] = infinity
bool finished = false
t = 1
while (t < m) and not finished
  finished = true;
  for k = 1 to m
    i = x_k
    j = y_k
    temp = V[i] + W_k
    if(temp < V[j])
      V[j] = temp
      back[j] = i
      finished = false
      t = t+1
end while
if (t > m) "There is no solution"
```

1. What does the following code compute? What is the loop invariant?

```
float mystery(float x, int n) // input condition: n > 0
 {
  assert(n > 0);
  float z = 1.0;
  float y = x*x; int m = 2*n; while(m > 0) {
    if(m%2) z = y*z;
    y = y*y;
    m = m/2;
   }
  return z;
 }
{\tt mystery(x,n)} computes $x^\maly{4n}$.  The loop invariant is
$x^\maly{4n} = y^\maly m*z$.
```

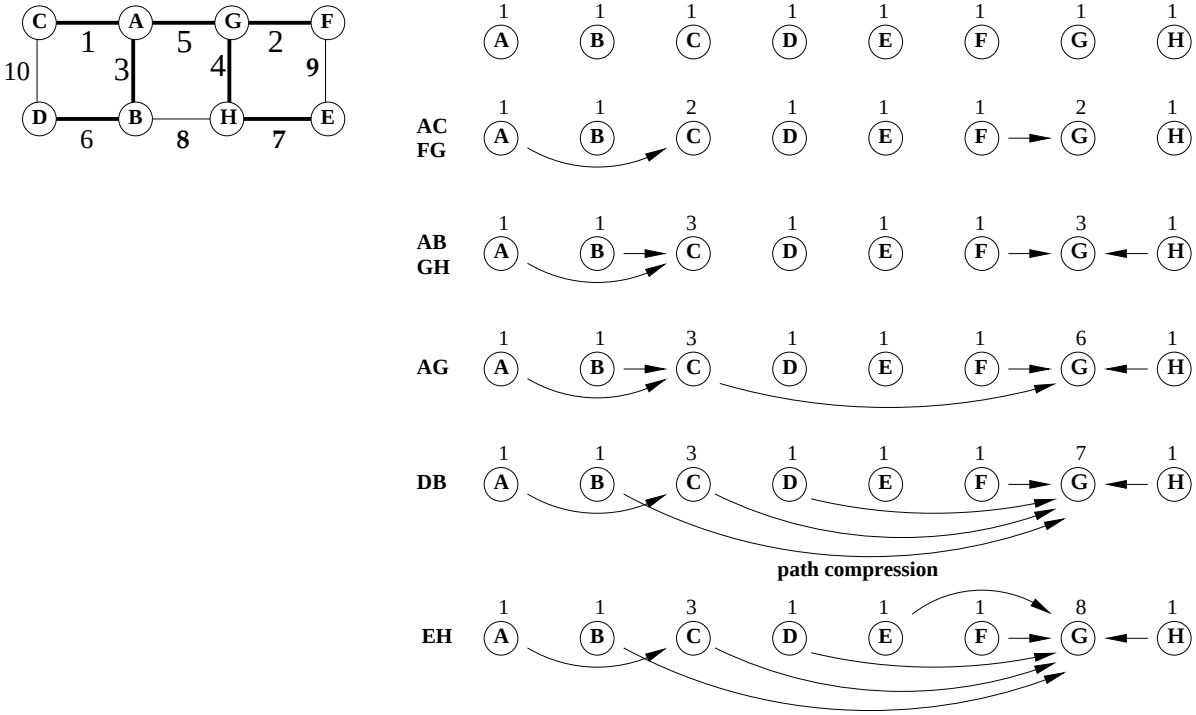
2. Sort the following array using heapsort. Add extra rows if needed.

1	2	3	4	5	6	7	8
A	N	H	Z	D	V	L	Q
A	N	V	Z	D	H	L	Q
A	Z	V	N	D	H	L	Q
A	Z	V	Q	D	H	L	N
Z	A	V	Q	D	H	L	N
Z	Q	V	A	D	H	L	N
Z	Q	V	N	D	H	L	A
A	Q	V	N	D	H	L	**Z**
V	Q	A	N	D	H	L	**Z**
V	Q	L	N	D	H	A	**Z**
A	Q	L	N	D	H	**V**	**Z**
Q	A	L	N	D	H	**V**	**Z**
Q	N	L	A	D	H	**V**	**Z**
H	N	L	A	D	**Q**	**V**	**Z**
N	H	L	A	D	**Q**	**V**	**Z**
D	H	L	A	**N**	**Q**	**V**	**Z**
L	H	D	A	**N**	**Q**	**V**	**Z**
A	H	D	**L**	**N**	**Q**	**V**	**Z**
H	A	D	**L**	**N**	**Q**	**V**	**Z**
D	A	**H**	**L**	**N**	**Q**	**V**	**Z**
A	**D**	**H**	**L**	**N**	**Q**	**V**	**Z**
A	**D**	**H**	**L**	**N**	**Q**	**V**	**Z**

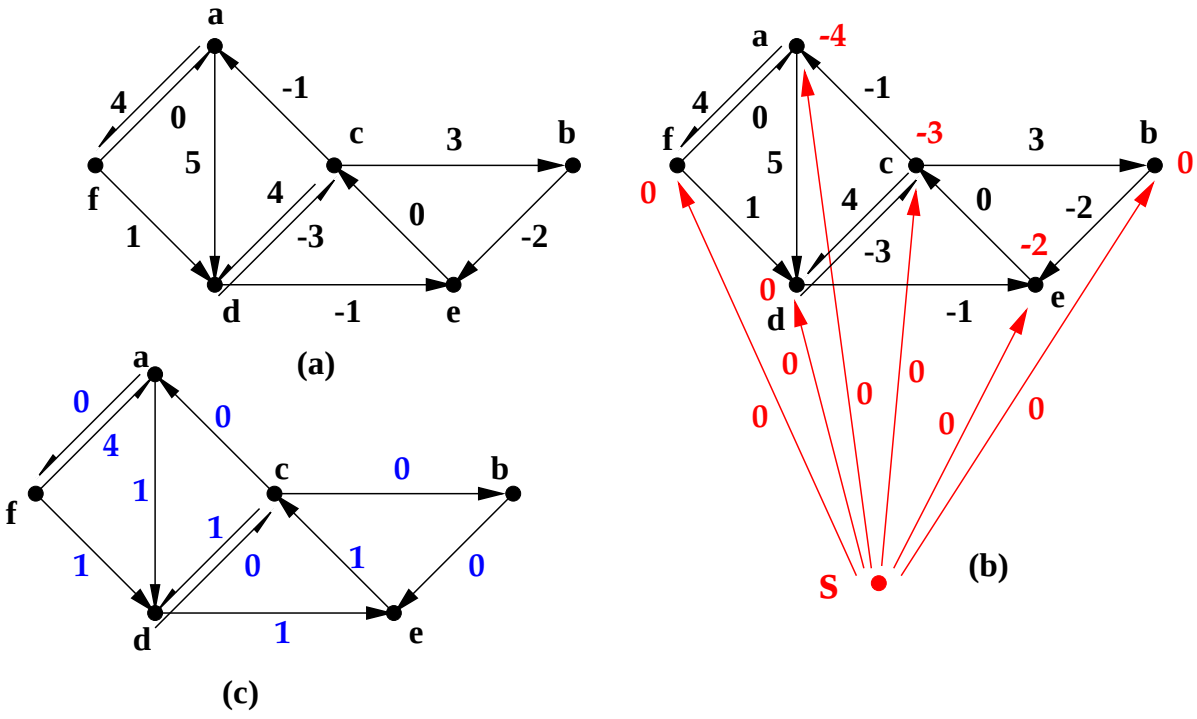
3. Explain how to implement a sparse array using memos and a search structure.

   Let A be the sparse virtual array. Let S be a search structure which contains ordered pairs of the form (i, x), where $A[i] = x$. To fetch the value of $A[i]$, search S for a pair (i, x). If that pair is found, return x, otherwise return a default value, such as 0. To store a value x for $A[i]$, search S for a pair (i, y). If that pair is found, replace y by x. That pair is not found, insert the pair (i, x) into S.

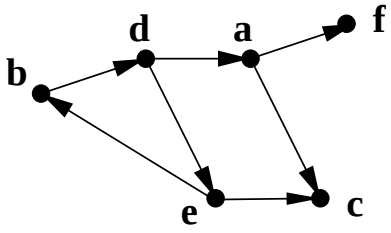
4. Walk through Kruskal's algorithm to find the minimum spanning tree of the weighted graph shown below. Show the evolution of the union/find structure. Whenever there is choice between two edges of equal weight, choose the edge which has the alphabetically largest vertex. (For making grading easier.) Whenever there is a union of two trees of equal weight, choose the alphabetically larger root to be the root of the combined tree. Indicate path compression when it occurs.



5. In the figure below, (a) shows a weighted directed graph. In (c), replace each edge weight using the techniques of Johnson's algorithm. Use (a) and (b) for your work. Do not complete Johnson's algorithm.



6. Write the in-neighbor list and out-neighbor list representations of the directed graph shown below.



a	e,f
b	d
c	
d	a,e
e	b,c
f	

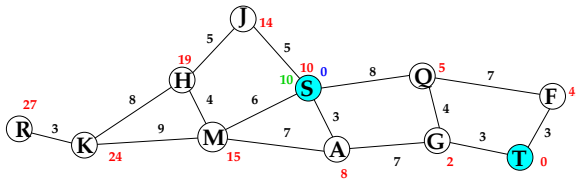
a	c
b	e
c	a,e
d	b
e	d
f	a

7. You are trying to construct a cuckoo hash table of size 10, where each of the 10 names listed below has the two possible hash values indicated in the array. Put the items into the table, if possible; otherwise, convince me it's impossible. Instead of erasing ejected items, simply cross them out, so that I can tell that you worked it properly.

	h1	h2
Ann	5	9
Ben	0	8
Cal	4	6
Deb	1	8
Eve	1	0
Fay	9	4
Gus	7	2
Hal	3	7
Ike	3	2
Jan	1	7

0	~~Cal~~ ~~Ann~~ Eve
1	~~Ann~~ ~~Dan~~ ~~Eve~~ ~~Dan~~ Ann
2	Fay
3	Hal
4	Bob
5	Gus
6	~~Dan~~ ~~Fay~~ Dan
7	Cal

8. Walk through the A^* algorithm for the weighted directed graph shown below, where the pair is (S, T). The heuristic is shown as red numerals.



.

Show the arrays and the contents of the heap at each step. h is the heuristic, f is the current distance from the source, g is the sum of h and f, while back is the backpointer.

Heap: S

	S	A	B	C	D	E	F	T
h	12	7	8	9	3	18	17	0
f	0							
g	12							
back								

Heap: BDE

	S	A	B	C	D	E	F	T
h	12	7	8	9	3	18	17	0
f	0		4		14	4		
g	12		12		17	22		
back			S		S	S		

Heap: ADE

	S	A	B	C	D	E	F	T
h	12	7	8	9	3	18	17	0
f	0	6	4		14	4		
g	12	13	12		17	22		
back		B	S		S	S		

Heap: DE

	S	A	B	C	D	E	F	T
h	12	7	8	9	3	18	17	0
f	0	6	4		11	4		
g	12	13	12		14	22		
back		B	S		A	S		

Heap: TE

	S	A	B	C	D	E	F	T
h	12	7	8	9	3	18	17	0
f	0	6	4		11	4		15
g	12	13	12		14	22		15
back		B	S		A	S		D

Heap: E

	S	A	B	C	D	E	F	T
h	12	7	8	9	3	18	17	0
f	0	6	4		11	4		15
g	12	13	12		14	22		15
back		B	S		A	S		D

T is fully processed, and we are done. The shortest path from S to T is (S,B,A,D,T) obtained by following the back pointers.

9. Here is another coin-row problem. You have a row of coins of various values, where the value of the i^{th} coin is $V[i] > 0$. Write pseudocode which finds the maximum value of a subset of coins, where the set may not contain coins which are either adjacent or just one apart in the row. That is, if the set contains the i^{th} coin, it may not contain either the $(i+1)^{st}$ coin or the $(i+2)^{nd}$ coin. For example, if the coins are ⓐ ⓑ ⓒ ⓓ ⓔ ⓕ ⓖ ⓗ in that order, the subset may be {ⓐ, ⓓ, ⓗ}, but not {ⓑ, ⓓ, ⓖ}. We have two dynamic programs for this problem.

   - Let $A[i]$ be the maximum sum of any legal sequence ending at i. Then the answer is $max(A[n-2], A[n-1], A[n])$ where the values of A are computed as follows.

     ```
     A[1] = V[1]
     A[2] = V[2]
     A[3] = V[3]
     A[4] = V[4] + A[1]
     A[5] = V[5] + max(A[1],A[2])
     for i from 6 to n
       A[i] = V[i] + max(A[i-5],A[i-4],A[i-3])
     ```

   - Let A[i] be the maximum sum of any legal subsequence of the first i coins. The answer is then $A[n]$ where the values of A are computed as follows.

     ```
     A[1] = V[1]
     A[2] = max(V[2],A[1])
     A[3] = max(V[3],A[2])
     for i from 4 to n
       A[i] = max(A[i-1],V[i] + A[i-3])
     ```

10. The following recursive code computes a function $F(n)$ for $n \geq 0$.

```
int F(int n)
 {
  if(n < 2) return 1;
  else return F(n/2)+F((n-1)/4)+F((n-2)/4) + n;
 }
```

(i) What is the asymptotic complexity of the function $F(n)$?

$\Theta(n \log n)$

(ii) What is the asymptotic time complexity of the computation of $F(n)$ using that recursive code? Assume that arithmetic operations take constant time.

$\Theta(n)$

Now, we wish to compute $F(n)$ for just one value of n, using memoization. We use the following recursive code, which stores memos into a sparse array, also named F. We modify the recursive code to store intermediate values in a search structure. `fetchF(n)` returns the stored value of $F(n)$, if it exists, and otherwise returns `default`, while `storeF(n,x)` stores a memo indicating that $F(n) = x$. Assume that `store` and `fetch` each take constant time.

```
int F(int n)
 {
  int result;
  if(n < 2) result = 1;
  else
   {
    int result = fetchF(n);
    if(result == default)
     {
      result = F[n/2] + F[n/4] + F[(n-1)/4] + n; // recursion here
      storeF(n,result);
     }
   }
  return result;
 }
```

(iii) What is the asymptotic time complexity of the computation of $F(n)$ using memoization?

$\Theta(\log n)$

(iv) What is the asymptotic space complexity of the computation of $F(n)$ using memoization?

$\Theta(\log n)$