Answers to Study for CS477 Final Examination December 11, 2024

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

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

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

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

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

   (vii) **F** Open hashing uses open addressing.

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

   (ix) **T** Binary tree sort (also called "treesort") can be considered to be an efficient implementation of insertion sort.

3. Fill in the blanks.

   (i) 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]);
```

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

   (iii) 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.)

(iv) The time complexity of every comparison-based sorting algorithm is $\Omega(n \log n)$. (Your answer should use Ω notation.)

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

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

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

(viii) **Huffman's** algorithm finds a binary code so that the code for one symbol is never a prefix of the code for another symbol.

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

(x) **Mergesort** and **Quicksort** are divide-and-conquer sorting algorithms that we've studied this semester.

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

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

(xiii) If a planar graph has 10 edges, it must have at least **6** vertices. (Exact answer. No partial credit.)

(xiv) 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.

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

```
int x[n];
obtain 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]);
```

(xvi) The prefix expression $*a+ \sim b * -c\,d \sim e$ is equivalent to the infix expression $a * (-b + (c - d) * e)$ and the postfix expression $ab \sim cd - e \sim * + *$.

(xvii) In **cuckoo** hashing, each item has more than one hash value, but only uses one of them.

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

(i)
```
for(i = 0; i < n; i = i+1);
cout << "Hello world!" << endl;
```
$\Theta(n)$

(ii)
```
for(int i = 2; i < n; i = i*i)
  cout << "Hello world" << endl;
```
$\Theta(\log \log n)$

(iii)
```
for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j = 2*j)
    cout << "Hello world" << endl;
```
$\Theta(n \log n)$

(iv)
```
for(int i = 1; i < n; i++)
  for(int j = i; j < n; j = 2*j)
    cout << "hello world" << endl;
```
$\Theta(n)$

(v)
```
for(int i = 1; i*i < n; i++)
  cout << "hello world" << endl;
```
$\Theta(\sqrt{n})$

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

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

(viii)
```
for(int i = n; i > 2; i=sqrt(i))
  cout << "Hello world!" << endl;
```
$\Theta(\log \log n)$

(ix)
```
for(int i = 1; i < n; i++)
  for(int j = 2; j < i; j=j*j)
    cout << "Hello world" << endl;
```
$\Theta(n \log \log n)$

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

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

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

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

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

(iii) $F(n) = 4F\left(\frac{n}{2}\right) + n$

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

(iv) $F(n) = F\left(\frac{n}{2}\right) + 2F\left(\frac{n}{4}\right) + n$

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

(v) $F(n) = 2F(n/2) + n^2$

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

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

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

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

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

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

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

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

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

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

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

(xi) $F(n) = 2F(n/4) + \sqrt{n}$

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

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

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

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

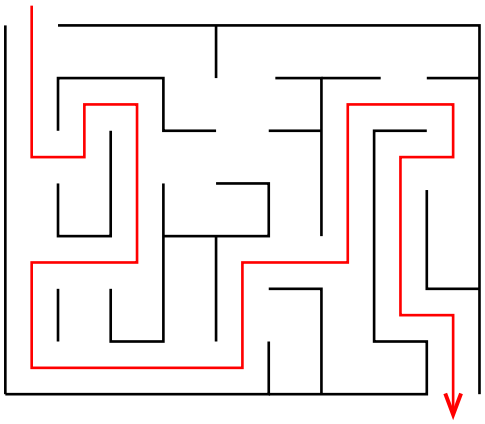
However, there is another recurrence:
$F[1] = F[2] = 1$
$F[n] = F\left[\frac{n-1}{2}\right] * F\left[\frac{n}{2}\right] + F\left[\frac{n+1}{2}\right] * F\left[\frac{n+2}{2}\right]$ for $n > 2$
where integer division is truncated as in C++.

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.

We keep a search structurer of memos of the form $(n, F[n])$. The following code computes $F[n]$.

```
int F(int n)
 {
  if n <= 2 return 1
  else if there is a memo (n,X) return X
  else
   X = F(n-1)+F(n-2)
   store the memo (n,X)
   return X
 }
```

7. 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.



Let G be the graph whose vertices are the squares of the maze. One vertex is the start, and one is the goal. There is an edge between adjecent squares if there is no wall between them. BFS, breadth first search, will find the shortest path. We use a queue. Initially the queue contains only the start vertex. At each step, the front vertex v is dequeued and all neighbors of v which have not been visited are enqueued. When the goal vertex is visited, the shortest path has been found, and can be reconstructed using backpointers.

8. 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

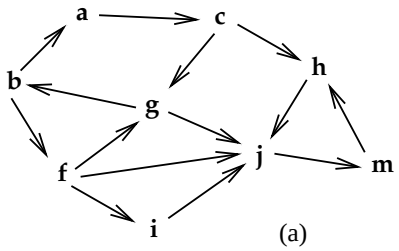
The Levenstein distance is 4.

9. 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, $I(3,2) = 8$.

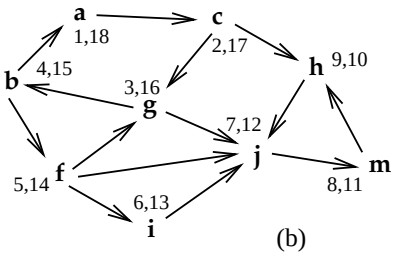
```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

```
int I(int n, int k)
 {
  // the position of n choose k in the linear array
  assert(k >= 0 and n >= k);
  int indx = k + n*(n+1)/2;
  return indx;
 }
```

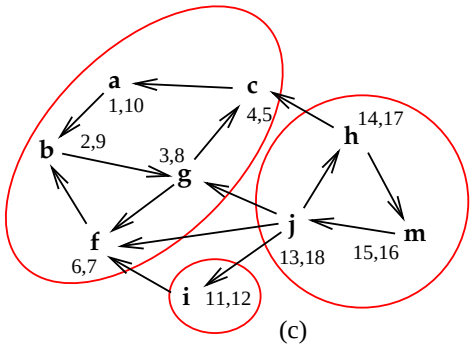
10. Use the DFS method to find the strong components of the digraph shown below. Show your steps.



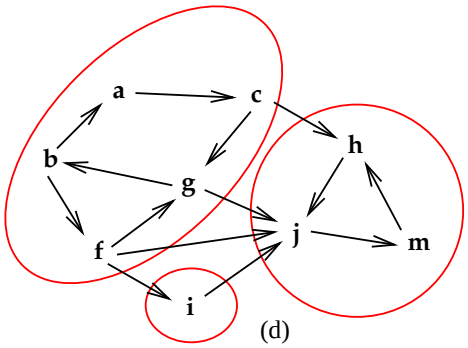
(a)



(b)

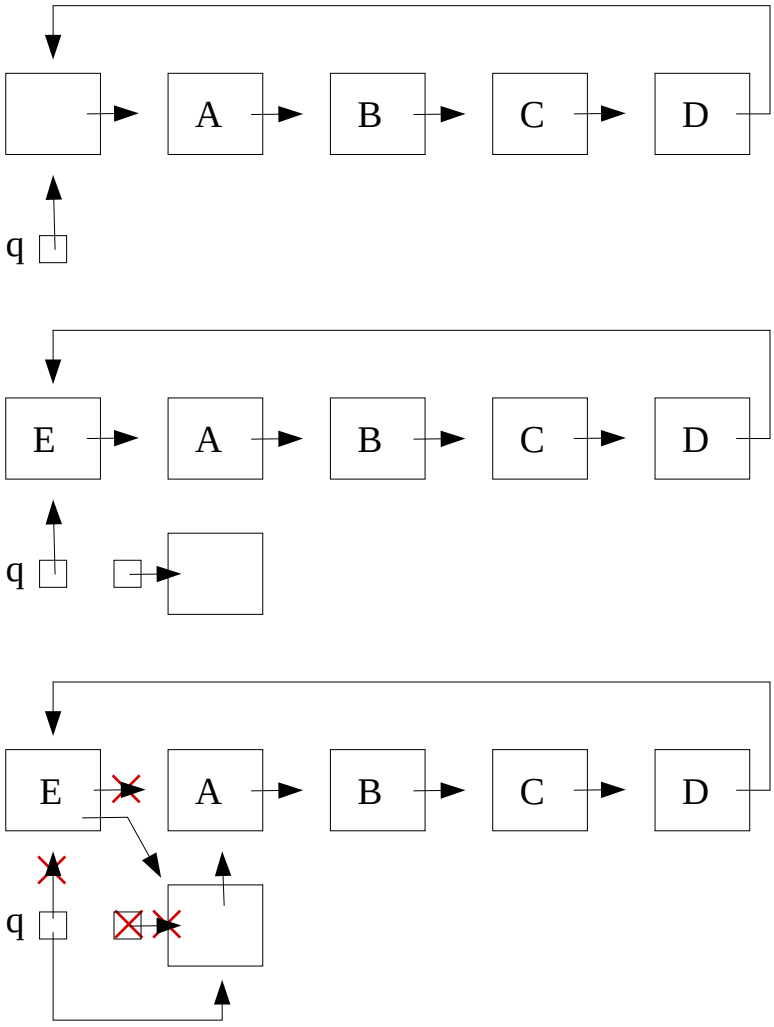


(c)



(d)

11. 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.



12. 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 of any directed path that ends at x?

Define $F(x)$ to be the maximum weight of any path ending at x. Let $x_1, x_2, \ldots x_n$ be the vertices of G in topolotical order. Let $In(i)$ be the set of in-neighbors of x_i.

For i from 1 to n
if In(i) is empty, $F(i) = 0$ and back(i) is undefined
else pick $j \in$ In(i) such that $F(j) + w(x_j, x_i)$ is maximum.
Let $F(i) = F(j) + w(x_j, x_i)$ and let back$(i) = j$.

Pick k such that $F(k)$ is maximum. The maximum path is found by following the back pointers from k.

13. Write pseudocode for the Bellman-Ford algorithm. Be sure to include the shortcut that ends the program when the final values have been found.

    We assume that the vertices are integers from 0 to n-1. There are m edges. The j^{th} edge is the ordered pair $(s[j], t[j])$, and has weight $w[j]$, where $0 \leq s_j, t_j < n$ for each j.

    We need to compute $V[i]$, the minimum weight of any directed path from 0 to i, for all i. For each $i \neq 0$, we also need to compute $B[i]$, the backpointer.

    Here is the algorithm. for all i $V[i] = infinity$;
    V[0] = 0
    bool finished = false;
    while(not finished)
    {
    finished = true;
    for(int $j = 0$; $j < m$; $j++$)
    {
    int temp = $V[s[j]] + w[j]$
    if(temp < $V[t[j]]$)
    {
    $V[t[j]]$ = temp;
    $B[t[j]] = s[j]$
    finished = false;
    }
    }
    }

14. List properties of a good hash function.

    - Deterministic.
    - Fast to compute.
    - More or less equally spread out, even if the data are clustered.
    - Closely related data do not have close hash values.

15. 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

ADEFGJLLMNPQSTVW

16. Consider the following C++ code.

```
int george(int n)
 {
  if(n == 0) return 1;
  else return george(n/2)+george(n/2-1)+n*n;
 }
```

(i) What is the asymptotic complexity of `george(n)`?

The recurrence is $G(n) = 2G(n/2) + n^2$, hence $G(n) = \Theta(n^2)$.

(ii) What is the asymptotic time complexity of the recursive code given above?

The recurrence is $T(n) = 2T(n/2) + 1$, hence $T(n) = \Theta(n)$.

(iii) What is the asymptotic time complexity of a dynamic programming algorithm to compute `george(n)`?

The values of george(i) are computed from 1 to n, and all values are stored in an array as they are computed. The time is $\Theta(n)$.

(iv) What is the space complexity of a computation of `george(n)` using memoization?

We need to compute how many values of $F(m)$ are computed. During the recursive call of depth i, $F(m)$ is computed for at most three values of m, all of which are approximatly $n/2^i$. Thus, i cannot exceed $\log_2 n$. Thus, $F(m)$ is computed for $\Theta(\log n)$ values of m, and each computation takes constant time. The time complexity of the memoization algorithm is thus $\Theta(\log n)$.

17. Write pseudcode for the simple coin-row problem we discussed in class. You are given a row of n coins of various 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 identify the coins which are selected.

$X[i]$ = value of the i^{th} coin. These are given.
$V[i]$ = maximum value of a legal subset whose last coin is i.
$B[i]$ =, the backpointer, next-to-the-last coin in the maximum value legal subset whose last coin is i.

Here is the algorithm.
$V[1] = X[1]$.
$V[2] = X[2]$.
$B[1]$ and $B[2]$ are undefined.
$V[3] = V[1] + V[3]$
$B[3] = 1$.
For all i from 4 to n:
  If $V[i-2] > V[i-3]$
    $V[i] = V[i-2] + X[i]$
    $B[i] = i - 2$
  Else

$V[i] = V[i-3] + X[i]$
$B[i] = i - 3$

If $V[n-1] < V[n]$ then $X[n]$ is the last coin in the optimal set, otherwise $X[n-1]$ is the last coin. In either case, use the backpointers starting at the last coin to find the optimal set.

18. 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. There are two versions. One is the one you've seen before, the other is the recursive version below.

```
float square(float x)
 {
  return x*x;
 }

float power(float x, int n)
 // input condition: x <> 0.0 and n < 0
 {
  if(n == 1) return x;
  else if(n%2) return x*power(x,n-1);
  else return square(power(x,n/2));
 }
```

19. Fill in the blanks.

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

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

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

    (iv) The following is pseudo-code for what algorithm? **selectionsort**

```
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]);
```

    (v) 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.

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

To make it clear, I've separated runs by commas. In an actual program, that would not be necessary. The time complexity of polyphase mergesort is thus $O(n \log n)$. Only 4 files are needed, since files are reused. The space complexity is thus only $O(n)$.

Input: AC,BX,FR,EY,GMQS,N,DZ

File 1: AC,FR,GMQS,DZ

File 2: BX,EY,N

File 3: ABCX,GMNQS

File 4: EFRY,DZ

File 1: ABCEFRXY

File 2: DGMNQSZ

File 3 = output: ABCDEFGMNQRSXYZ

21. 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;
```

The loop invariant is xn = ym + z

22. Write pseudo-code for the Floyd/Warshall algorithm. Let the vertices be $\{1, 2, \ldots n\}$. Let $W(i, j)$ be the given weight of the arc (i, j), if any, where $W(i, j) = \infty$ if there is no arc. Compute $V(i, j)$, the minimum weight of any path from i to j, and $B(i, j)$, the backpointer for that minimum path.

```
for all i from 1 to n
```

```
 V(i,i) = 0
for all i from 1 to n
 for all j from 1 to n
  V(i,j) = W(i,j)
  B(i,j) = i
for all j from 1 to n
 for all i from 1 to n
  for all k from 1 to n
   temp = V(i,j)+V(j,k)
   if(temp < V(i,k))
    V(i,k) = temp
    B(i,k) = B(j,k)
```

23. 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. Answer: $i * 10 * 18 + j * 18 + k$

24. 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)
 {
  assert(s.size < N);
  s.item[s.size] = i;
  s.size++;
 }
bool empty{s&stack)
 {
  return s.size == 0;
 }
 int pop(s&stack)
  {
   assert(not empty(s));
   s.size--;
   return s.item[s.size];
  }
```

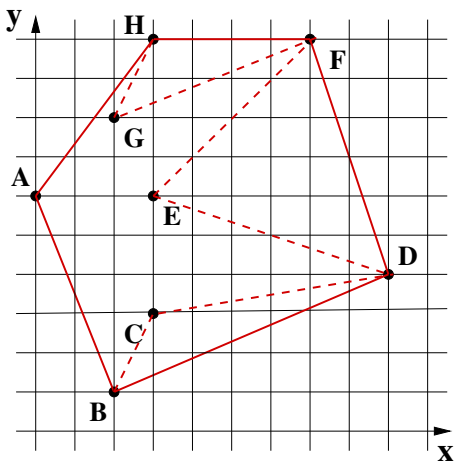
25. In class, we implemented a minheap as an almost complete binary tree implemented as an array. Suppose the minheap is initialized as shown in the first line of the array shown below. Show the evolution of the structure when deletemin is executed.

A	C	F	D	Q	H	L	R	Z
Z	C	F	D	Q	H	L	R	
C	Z	F	D	Q	H	L	R	
C	D	F	Z	Q	H	L	R	
C	D	F	R	Q	H	L	Z	

26. Starting from the configuration given, show the evolution of the structure when B is inserted.

C	D	F	R	Q	H	L	Z	
C	D	F	R	Q	H	L	Z	B
C	D	F	B	Q	H	L	Z	R
C	B	F	D	Q	H	L	Z	R
B	C	F	D	Q	H	L	Z	R

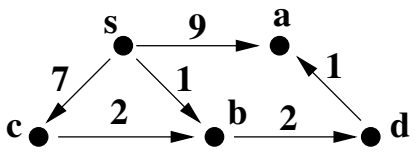
27. Using one of the algorithm we mentioned in class, find the convex hull of the set of points indicated in the figure below. Show your steps.



Slopes
AB − 5/3
AC − 1
AD −9/2
AE 0
AF 4/7
AG 1
AH 4/3

The direction of a turn can determend by sign of the cross product of vectors. The cross product of two-dimensional vectors is the determinant of a 2×2 matrix. A positive determinant indicates a left turn. For example, $\overrightarrow{AB} = (2, -5)$ $\overrightarrow{BC} = (1, 2)$, thus $\overrightarrow{AB} \times \overrightarrow{BC} = \begin{vmatrix} 2 & -5 \\ 1 & 2 \end{vmatrix} = 9$. Thus the turn at B is a left turn. On the other hand, $\overrightarrow{CD} = (6, 1)$ and $\overrightarrow{BC} \times \overrightarrow{CD} = \begin{vmatrix} 1 & 2 \\ 6 & 1 \end{vmatrix} = -11$. Thus the turn at C is a right turn.

28. 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

29. Find an optimal prefix code for the alphabet $\{a, b, c, d, e, f\}$ where the frequencies are given in the following array.

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