Answers to Study Guide for CS477 Examination April 10, 2024

In any problem involving grasphs or directed graphs, we use n to denote the number of vertices and m the number of edges, or arcs.

1. Fill in the blanks.

   (i) Name two greedy algorithms introduced in class this semester. **Huffman's Kruskal's** .

   (ii) In closed hashing, collisions are resolved by the use of **probe** sequences.

   (iii) The items in a priority queue represent **unfulfilled obligations**.

   (iv) Name three kinds of search structures.

   Here are three that are commonly used. **unordered list, hash table, binary search tree.**

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

   (vi) In closed hashing, if a collision occurs, a **probe sequence** can be used to locate an unused position in the hash table.

   (vii) In a **cuckoo** hash table, each item has two or more possible locations, and must be stored in one of those.

   (viii) (3) Which of the following three statements is closest to the truth?
   (1) In SHA256 hashing, collisions are impossible.
   (2) In SHA256 hashing, collisions occur no more than once a year in practice.
   (3) In SHA256 hashing, collisions are so unlikely that industry experts claim they never occur.

   (ix) The worst case time complexity of quicksort on a list of length n.

   $O(n^2)$

   (x) The average case time complexity of quicksort on a list of length n, if pivots are chosen at random.

   $\Theta(n \log n)$

   (xi) The worst case time complexity of building a treap with n items.

   $O(n^2)$

   (xii) The average case time complexity of building a treap with n items.

   $\Theta(n \log n)$

   (xiii) In an open hash table of size m holding n data items, the items at each index of the table are typically shown as linked list. However, that structure is only efficient if m/n is fairly small. In general, we should use a **search structure** at each table index.

   (xiv) 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.

2. For each of these recursive subprograms, write a recurrence for the time complexity, then solve that recurrence.

(i)
```
void george(int n)
{
 if(n > 0)
  {
   for(int i = 0; i < n; i++) cout << "hello" << endl;
   george(n/2); george(n/3);
  }
}
```

$T(n) = T(n/2) + T(n/3) + n$

$T(n) = \Theta(n)$

(ii)
```
void martha(int n)
{
 if(n > 0)
  for(int i = 1; i < n; i++)
   for(int j = 1; j < i; j++)
    cout << "hello world";
  martha(n/2);
}
```

$T(n) = T(n/2) + n^2$

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

3. A 3-dimensional $10 \times 20 \times 12$ rectangular array A is stored in main memory in column major order, and its base address is 1024. Each item of A takes two words of main memory, that is, two addressed location. Find the address, in main memory, of A[5][13][7].

   $1024 + 2 * (7 * 20 * 10 + 13 * 10 + 5) = 4094$

4. You are trying to construct a cuckoo hash table of size 8, where each of the 8 names listed below has the two possible hash values indicated in the array. Put the items into the table, if possible. Instead of erasing ejected items, simply cross them out, so that I can tell that you worked it properly.

	h1	h2
Ann	1	0
Bob	4	2
Cal	0	7
Dan	1	6
Eve	1	0
Fay	6	2
Gus	5	3
Hal	3	7

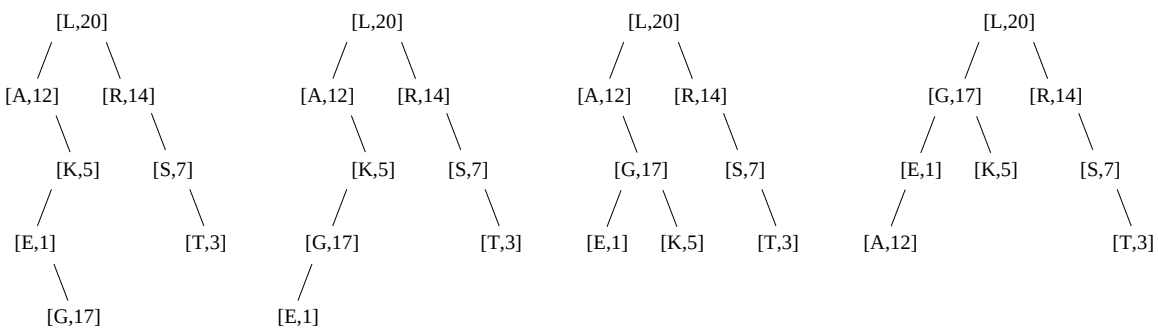
0	~~Cal~~ ~~Ann~~ Eve
1	~~Ann~~ ~~Dan~~ ~~Eve~~ ~~Dan~~ Ann
2	Fay
3	Hal
4	Bob
5	Gus
6	~~Dan~~ ~~Fay~~ Dan
7	Cal

5. Let $\sigma = x_1, x_2, \ldots x_n$ be a sequence of numbers with both positive and negative terms. Write an $O(n)$ time dynamic program which finds the maximum sum of any contiguous subsequence of σ. For example, if the sequence is $-1, 4, -3, 2, 7, -5, 3, 4, -8, +6$ then the answer is $4 - 3 + 2 + 7 - 5 + 3 + 4 = 12$.

   We define: $A[i]$ = maximum sum of any legal subsequence which ends at x_i.
   $B[i]$ = maximum sum of any legal subsequence of $x_1, \ldots x_i$

   Here is the dynamic program:
   $B[0] = 0$
   $A[1] = x_1$
   $B[1] = max(0, A[1])$
   for($i = 2$ to n)
   {
   $A[i] = max(0, A[i-1]) + x_i$
   $B[i] = max(B[i-1], A[i])$
   }

6. The figure below shows a treap, where the data are letters and the nodes of the tree are memos, where the first component is the <u>key</u>, a letter, and the second component is a the <u>priority</u>, a random integer. Insert the letter G, where the priority is chosen (at random) to be 17. Show the steps.



7. Explain how to implement a sparse array using 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.

8. Consider the following two C++ subprograms.

```
int f(int n)
  {
   if(n > 0)
    return f(n/2)+f(n/3)+f(n/6)+n*n;
   else
    return 0;
  }

void computef(int n)
 {
  f[0] = 0;
  for(int i = 0; i <= n; i++)
   f[i] = f[i/2]+f[i/3]+f[1/6]+n*n;
 }
```

(i) The first of those subprograms is a recursive function. What is the asymptotic value of f(n)?

$\Theta(n^2)$

(ii) What is the asymptotic time complexity of the computation of f(n) using the recursive function?

$\Theta(n)$

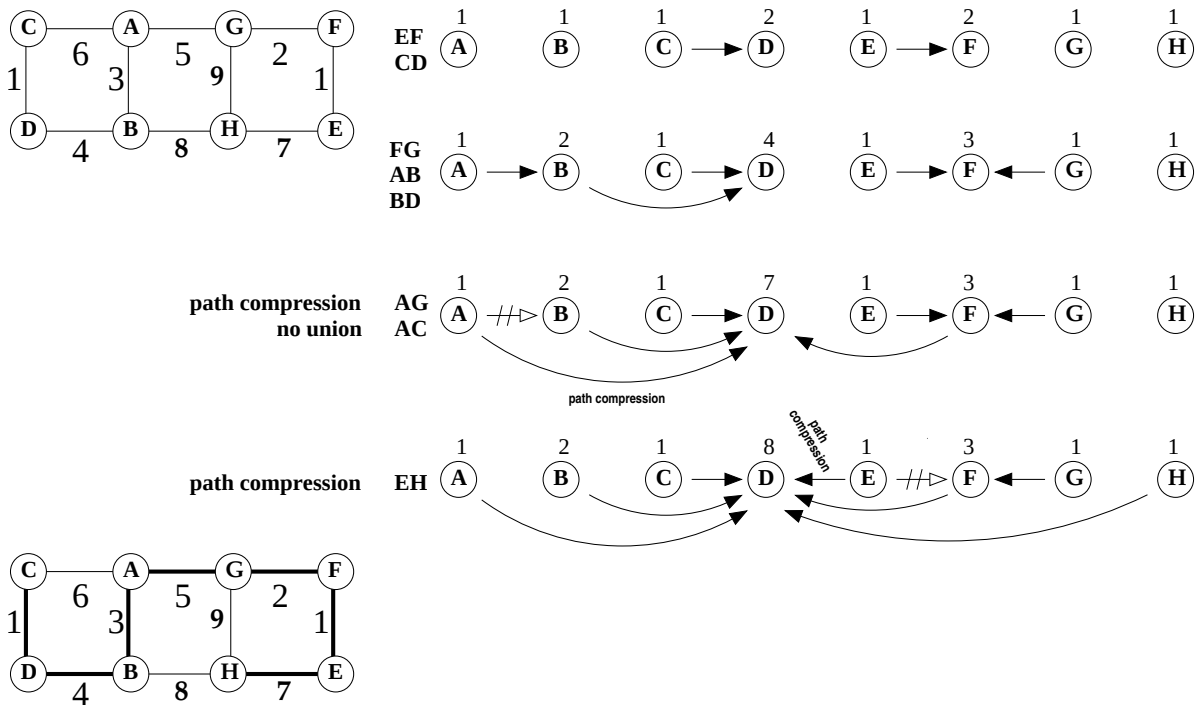
(iii) The second subprogram uses dynamic programming, and stores values in an array. What is the asymptotic time complexity of that computation?

$\Theta(n)$

(iv) What is the asymtotic time complexity of a computation of f(n) using memoization? (Hint: it's a power of log n.)

$\Theta(\log^2 n)$

9. 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. 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.



10. Write the prefix expression equivalent to the infix epression $-a * b - (-c - d) \wedge e$
(Don't forget that $\wedge$ means exponentiation.)

$-* \sim ab \wedge - \sim cde$

Some people wrote postfix instead. I gave partial credit. That answer is:

$a \sim b * c \sim d - e \wedge -$

11. Walk through the stack algorithm to change the infix expression $-a + b \wedge c \wedge -f$ to postfix. Show the stack at each step.

stack	infile	outfile
	$-a + b \wedge c \wedge -f$	
$\sim$	$a + b \wedge c \wedge -f$	
$\sim$	$+b \wedge c \wedge -f$	a
	$+b \wedge c \wedge -f$	$a \sim$
$+$	$b \wedge c \wedge -f$	$a \sim$
$+$	$\wedge c \wedge -f$	$a \sim b$
$+\wedge$	$c \wedge -f$	$a \sim b$
$+\wedge$	$\wedge - f$	$a \sim bc$
$+\wedge\wedge$	$-f$	$a \sim bc$
$+\wedge\wedge\sim$	f	$a \sim bc$
$+\wedge\wedge\sim$		$a \sim bcf$
$+\wedge\wedge$		$a \sim bcf \sim$
$+\wedge$		$a \sim bcf \sim \wedge$
$+$		$a \sim bcf \sim \wedge\wedge$
		$a \sim bcf \sim \wedge\wedge +$

12. Up to now, no one has written a polynomial time algorithm for the subset sum problem, given below. However, there is a pseudopolynomial time algorithm. The problem is to find a subsequence of a sequence of positive integers $x[1], \ldots x[n]$ whose sum is K. The time complexity of the algorithm is is $O(nK)$, so you might think the algorithm is polynomial time, but it isn't. Write code or pseudocode for the pseudopolynomial time algorithm for deciding whether there is a solution to a given instance of the subset sum problem.

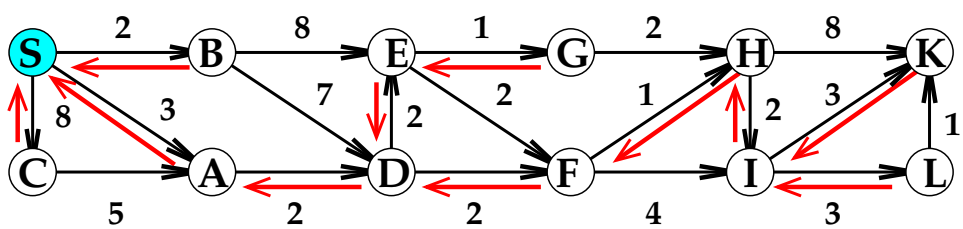
    The algorithm computes an $n \times K$ array of Boolean. bool $S[n+1][K+1]$; // $S[i][k]$ means there is a subsequence of $x[1] \ldots x[i]$ whose total is k

```
S[0][0] = true // the empty set is a solution if K = 0
for all k from 1 to K
   S[0][k] = false;
for all i from 1 to n // begin main outer loop
   for all k from 0 to K // begin main inner loop
     {
       S[i][k] = S[i − 1][k]
       if(S[i − 1][k] and k + x_i ≤ K)
         S[i][k + x_i] = true
     }
if(S[n][K]) write "There is a solution"
else write "There is no solution"
```

13. For the algorithm to be polynomial time, its time complexity must be a polynomially bounded funtion of the input size of the instance. For the instance given in the previous problem, the input size is not actually $O(nK)$. What is it?

    Each of the numbers x_i could be up to size K, hence be written with $O(\log K)$ bits. Thus, the input size is $O(n \log K)$. The algorithm is not $\mathcal{P}$, since nK is not a polynomially bounded function of $n \log K$.

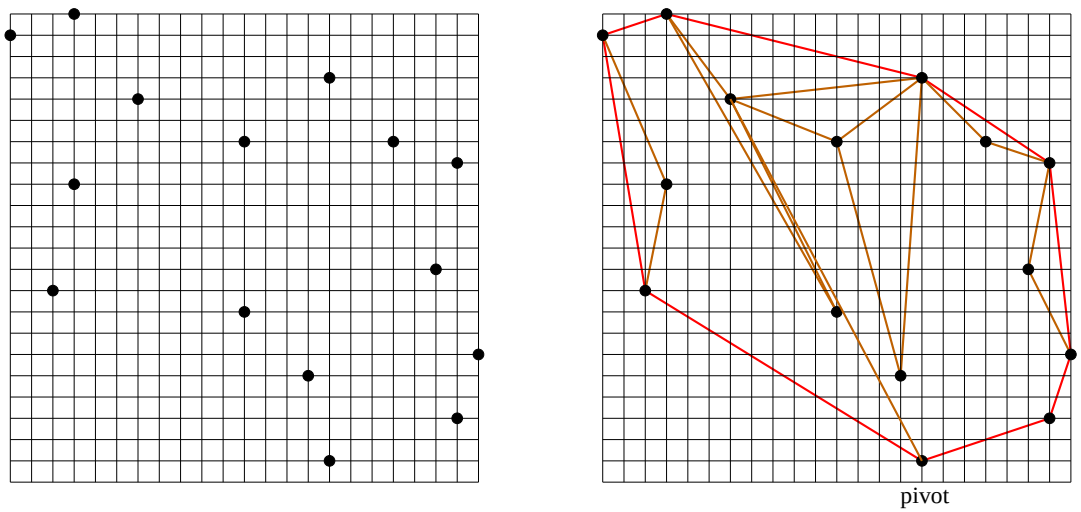
14. Walk through Dijkstra's algorithm for the following graph.



	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8	~~9~~ 5	~~10~~ 7	7	8	8	~~11~~ 10	~~16~~ 13	13
back		S	S	S	~~B~~ A	~~B~~ D	D	E	F	~~F~~ H	~~H~~ I	I

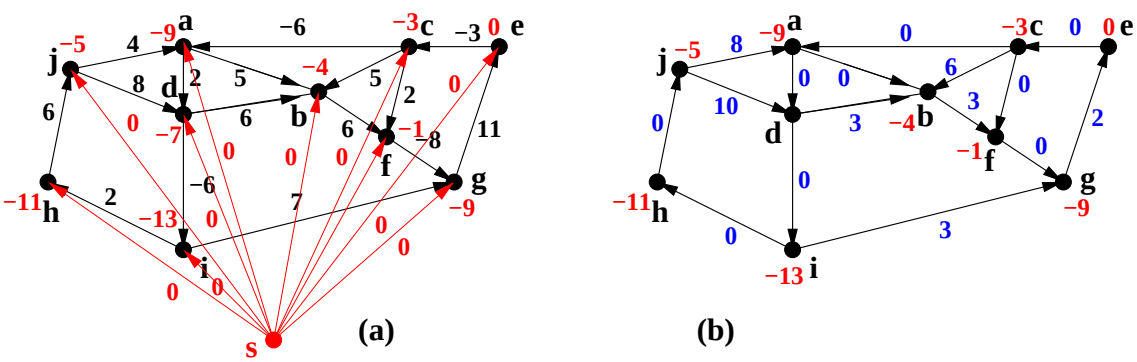
15. The convex hull of a set of a finite set of points in a plane is the smallest convex polygon which encloses the points, together with its interior. Walk through Graham Scan to find the convex hull of the points in the plane given in this figure. (I have not gone over Graham Scan yet.)

    I will not give a convex hull question on the April 10 exam, since we haven't covered it thoroughly.



16. Figure (a) below shows an instance of the all-pairs minpath problem. Work the first part of Johnson's algorithm on that graph, showing the adjusted weights in Figure (b).

    Do not complete the computation of Johnson's algorithm.



17. Walk through heapsort for the list BGHKRET.

1	2	3	4	5	6	7
B	G	H	K	R	E	T
B	G	T	K	R	E	H
B	R	T	K	G	E	H
T	R	B	K	G	E	H
T	R	H	K	G	E	B
B	R	H	K	G	E	T
R	B	H	K	G	E	T
R	K	H	B	G	E	T
E	K	H	B	G	R	T
K	E	H	B	G	R	T
K	G	H	B	E	R	T
E	G	H	B	K	R	T
H	G	E	B	K	R	T
B	G	E	H	K	R	T
G	B	E	H	K	R	T
E	B	G	H	K	R	T
B	E	G	H	K	R	T

18. True or False.

(i) **F** If there are 100 data items and 200 possible hash values, a collision is so unlikely that you can, in practice, assume that it won't happen.

(ii) **F** If there are 100 data items and 10000 possible hash values, collisions are so unlikely that you can, in practice, assume that they won't happen.

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

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

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

(vi) **T** False overflow for a queue can be avoided by implementing the queue as a circular list.

(vii) **T** If a stack is implemented as a linked list, the head of the linked list should hold the top item of the stack.

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

(ii) **F** There will be no collisions if the size of a hash table is at least ten times the number of data items.

(iii) **T** A hash function should appear to be random, but cannot actually be random.

19. Solve each recurrence, expressing each answer in terms of O, Ω, or Θ, whichever is most appropriate.

(i) $F(n) = F(n/3) + F(2n/3) + 1$

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

(ii) $G(n) = 2G(n/4) + \sqrt{n}$

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

(iii) $H(n) = \log n + 1$

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

(iv) Solve the recurrence: $H(n) < 4H(2n/5) + H(3n/5) + 2n^2$

$4(2/5)^2 + (3/5)^2 = 1$. Therefore $H(n) = \Theta(n \log n)$.

(v) Solve the recurrence: $G(n) = 4(G(n/2) + 5n^2$

$4(1/2)^2 = 1$, therefore $G(n) = \Theta(n \log n)$.

(vi) Solve the recurrence: $F(n) = F(n - \log n) + \log^2 n$

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

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

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

20. Find the time complexity of each of these code fragments in terms of n, using Θ notation.

(i)
```
for(int i = n; i > 1; i = i/2)
 cout < "Hello world!";
```
$\Theta(\log n)$

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

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

21. The asymptotic complexity of the Floyd/Warshall algorithm is $\Theta(n^3)$

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

23. Write pseudocode for the Floyd Warshall algorithm.

```
For i from 1 to n and for j from 1 to n
 {
  V[i,j] = W[i,j] // might be infinity
  back[i,j] = i
 }
for i = 1 to n
  V[i,i] = 0;
for j = 1 to n
  for i = 1 to n
    for k = 1 to n
      {
       temp = V[i,j]+V[j,k]
       if(temp < V[i,k])
         {
          V[i,k] = temp
```

```
        back[i,k] = back[j,k]
       }
    }
```

24. Write pseudocode for the Bellman Ford algorithm. Be sure to include the shortcut that stops execution when further computation is unnecessary.

```
For all i from 1 to n V[i] = infinity
V[0] = 0
bool finished = false
while not finished
  {
   finished = true
   For all j from 1 to m
     {
      temp = V[S[k]] + W[k]
      if(temp < V[T[k]])
        {
         V[T[k]] = temp
         back[T[k]] = S[k]
         finished = false
        }
     }
```

25. Here is another coin-row problem. You have a row of coins of various values, where the value of the ith 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 ith coin, it may not contain either the $(i+1)^{\text{st}}$ coin or the $(i+2)^{\text{nd}}$ 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])
```

26. Execute heapsort for the list BXQVRST. Show the array at each step, and identify the step at which the array is a heap for the first time.

B	X	Q	V	R	S	T
B	X	T	V	R	S	Q
X	B	T	V	R	S	Q
X	V	T	B	R	S	Q
Q	V	T	B	R	S	**X**
V	Q	T	B	R	S	**X**
V	R	T	B	Q	S	**X**
S	R	T	B	Q	**V**	**X**
T	R	S	B	Q	**V**	**X**
Q	R	S	B	**T**	**V**	**X**
S	R	Q	B	**T**	**V**	**X**
B	R	Q	**S**	**T**	**V**	**X**
B	R	Q	**S**	**T**	**V**	**X**
R	B	Q	**S**	**T**	**V**	**X**
Q	B	**R**	**S**	**T**	**V**	**X**
B	**Q**	**R**	**S**	**T**	**V**	**X**
B	**Q**	**R**	**S**	**T**	**V**	**X**

27. A **perfect** hash function fills the hash table exactly with no collisions.

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

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

30. In **open hashing** or **separate chaining** there can be any number of items at a given index of the hash table.

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

32. Give the asymptotic complexity, in terms of n, for each of these code fragments.

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

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

33. Solve each recurrence, giving asymptotic answers, using O, Ω, or Θ, whichever is most appropriate.

(i) $F(n) \leq 4F(n/2) + n^2$

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

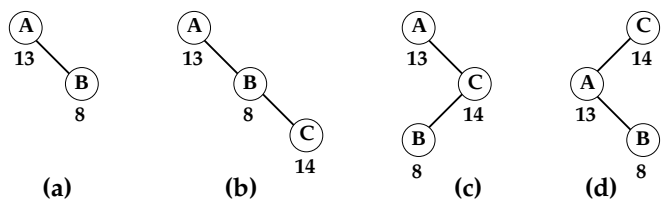
(ii) $G(n) \geq G(4n/5) + G(3n/5) + n^2$

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

(iii) $T(n) = T(3n/10) + T(n/5) + n$

$T(n) = \Theta(n)$

34. You need to store the items A, B, and C, in that order, in a treap. The priority for A is 13, for B is 8, and for C is 14. Use maxheap order. Draw the resulting treap after each insertion, and show each rotation.



We first insert A, then B. Heap order is preserved, so no rotation is necessary. We next insert C. To restore heap order, we do a left roatation at B. Heap order is still not restored. We do a left rotation at A. Heap order is then restored.

35. The BFPRT selection algorithm has asymptotic time complexity $\Theta(n)$, which is proved using a recurrence. Give that recurrence.

We have not covered the BFPRT algorithm, and it will not be on the April 10 exam.

$T(n) = T(n/5) + T(7n/10) + n$

36. Consider the function F computed by the recursive code given below.

(i) What is the asymptotic complexity of $F(n)$? Recurrence: $F(n) = 3F(n/3) + n^2$ Solution: $F(n) = \Theta(n^2)$

(ii) What is the asymptotic time complexity of the recursive code when it computes $F(n)$? Recurrence: $F(n) = 3F(n/3) + 1$ Solution: $F(n) = \Theta(n)$

(iii) What is the asymptotic time complexity of a memoization algorithm which computes $F(n)$? $F(n) = \Theta(\log n)$

```
int F(int n)
 {
  if(n < 3) return 1;
  else return F(n/3)+2*F((n+1)/3)+n*n;
 }
```

37. If the array $A[5][7]$ is stored in column-major order, how many predecessors does $A[3][4]$ have?

    4*5 + 3 = 23

38. Explain how to implement a sparse array using a search structure.

    Each item in my search structure is an ordered pair (i,x) where i is the sparse array index and x is the datum. The search key is i. To execute fetch(i) search for a pair (i,x) and return x. If such a pair is not found, return the default value. To execute store(i,x), search for a pair (i,y) and overwrite y with x. If a pair with index i is not found, store the pair (i,x).

39. You are implementing a 3D triangular array A where $A[i][j][k]$ is defined for $i \geq j \geq k \geq 0$, as a one-dimensional subarray of main memory, and you wish to store A in row-major order, with base address 1024. where $A[i][j][k]$ is defined for $i \geq j \geq k \geq 0$, Each term of A takes one place in main memory. What would be the address, in main memory, of $A[7][4][3]$?

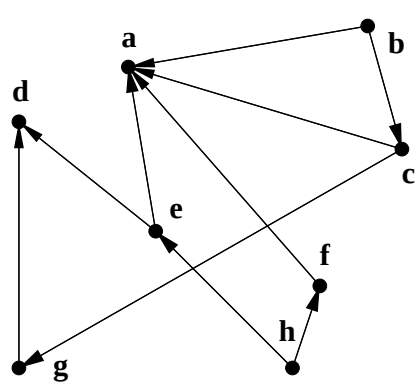
    There are choose(7+2,3)+choose(4+1,2)+choose(3,1) = 84+10+3 = 97 predecessors of A[7][4][3], hence A[7][4][3] is located at address 97+1024 = 1121

    This problem is far too lengthy to be on any exam, but I might put it on a homework assignment later.

40. You are given an acyclic directed graph $G = (V, E)$ where each arc has a positive weight. 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. (The weight of a path is defined to be the sum of the weights of its constituent arcs.) (Hint: your algorithm should use words such as "in-neighbor" or "out-neighbor.") Let the vertices be V[1] ... V[n] in topological order. Let W[i][j] be the weight of the arc from V[i] to V[j], if that arc exists. We compute A[i] to be the maximum weight of any arc which ends at V[i]. Let InNb[i] be the set of all indices of in-neighbors of V[i].

```
for all i from 1 to n
  A[i] = max{j in InNb[i]}{A[j] + W[j][i]}
result = max_{1\le i\le n}{A[i]}
return result
```

41. Write the array of in-neighbor lists and the array of out-neighbor lists for the directed graph shown below.



a	
b	a,c
c	a,g
d	
e	a,d
f	a
g	d
h	e,f

a	b,c,e,f
b	
c	b
d	e,g
e	h
f	h
g	c
h	

42. Consider the following recursive C++ function.

```
int f(int n)
 {
  if(n > 0) return f(n/2)+f(n/4)+f(n/4 + 1)+n;
  else return 0;
 }
```

(i) What is the asymptotic complexity of f as a function of n, using Θ notation?

The recurrence is $f(n) = f(n/2) + 2f(n/4) + n$ By the generalized master theorem, $f(n) = \Theta(n \log n)$.

(ii) What is the asymptotic time complexity of this code as a function of n, using Θ notation?

The recurrence is $T(n) = T(n/2) + 2T(n/4) + 1$ By the generalized master theorem, $T(n) = \Theta(n)$. Ans $\Theta(n)$

(iii) Write pseudo-code for a dynamic programming algorithm to compute $f(n)$ for a given n. What is the asymptotic time complexity of your code as a function of n, using Θ notation?

```
f[0] = 0;
for(int i = 1; i <= n; i++)
  f[i] = f[i/2] + f[i/4] + f[i/4 + 1] + i;
cout << f[n] << endl;
```

The value of $f(i)$ is computed for each i up to n. Ans: $\Theta(n)$

(iv) for a given n. What is the asymptotic complexity of a memoization algorithm to compute $f(n)$, in terms of n, using Θ notation?

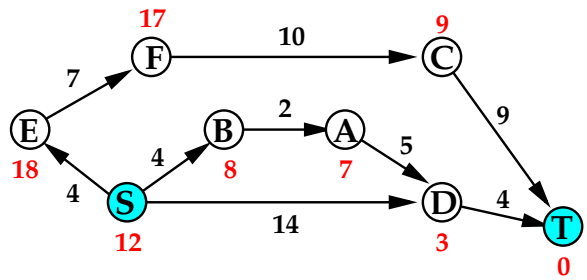
Represent the subproblem f[i] by the integer i. There is one subproblem for each integer from 0 to n. The subproblems are the vertices of a directed graph. There is an arc from i to j if the computation of f[j] requires the value of f[i]. We need to find the number of predecessors of n in this directed graph. It helps to work out an example. Let n = 1785. We need to compute f for the following integers: 1785, 892, 446, 447, 223, 224, 111, 112, 55, 56, 27, 28, 29, 13, 14, 15, 6, 7, 8, 3, 4, 1, 2, 0.

Except for the smallest few, the predecessors are in blocks where each block starts with n divided by a power of 2 and has at most three members. Thus the number of predecessors is approximately $3 \log_2$n. Thus the number of memos stored is $\Theta(\log n$.

But what is the time complexity if fetching from the search structure takes logarithmic time?

In that case, The search time needed is $O(\log n \log \log n)$.

43. 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.

44. Find the Levenshtein edit distance from the word "mennoover" to the word "maneuver." Show the matrix.

Since we did not go over the Levenshtein algorithm, it will not be on the April 10 exam.

		m	a	n	e	u	v	e	r
	0	1	2	3	4	5	6	7	8
m	1	0	1	2	3	4	5	6	7
e	2	1	1	2	2	3	4	5	6
n	3	2	2	1	2	3	4	5	6
n	4	3	3	2	2	3	4	5	6
o	5	4	4	3	3	3	4	5	6
o	6	5	5	4	4	4	4	5	6
v	7	6	6	5	5	5	4	5	6
e	8	7	7	6	5	6	5	4	5
r	9	8	8	7	6	7	6	5	4

The edit distance is 4.

45. (i) Fine the longest strictly monotone increasing subsequence of the sequence 1,5,2,2,4,8,7. The answer might not be unique. If there are choices, give just one answer. **I have not gone over this problem at all, although it is explained in the handout lms.pdf. It will not be on the April 19 exam.**

1,2,4,7 or 1,2,4,8

(ii) Write pseudocode for finding the length of the longest strictly monotone increasing subsequence of any given sequence of integers. (Hint: Use dynamic programming.)

The algorithm is found in the handout titled, "Longest Monotone Subsequence," which you can find as Handouts/lms.pdf.