University of Nevada, Las Vegas Computer Science 477/677 Spring 2025

Answers to Study for Final Examination May 12, 2025

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) **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) 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 is that
   1. For any $0 < i \leq lo$, $A[i] < A[0]$.
   and
   2. For any $hi < i < n$, $A[i] > A[0]$.

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

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

(xi) **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)
```

This will not halt. I should have started with "i = 2," in which case the answer would be $\Theta(n)$"

(iii)
```
for(int i = 1; i < n; i++)
  for(int j = i; j < n; j = 2*j)
```

$\Theta(n)$

(iv)
```
for(int i = n; i > 1; i--)
  for(int j = 1; j < i; j = 2*j)
```

$\Theta(n \log n)$

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

$\Theta(n \log n)$

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

$\Theta(n)$

(vii)
```
for(int i = 1; i*i < n; i++)
```

$\Theta(\sqrt{n})$

(viii)
```
for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j = 2*j)
```

$\Theta(n \log n)$

(x)
```
for(int i = 1; i < n; i++)
  for(int j = 2; j < i; j=j*j)
```

$\Theta(n \log \log n)$

(xii)
```
for(int i = 1; i < n; i = i+i)
```

$\Theta(\log n)$

(xiii) `for(int i = 2; i < n; i = i*i)`

$\Theta(\log \log n)$

(xv) `for(int i = n; i > 2; i = sqrt(i))`

$\Theta(\log \log n)$

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

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

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

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

(v) $F(n) = 2F(n/4) + \sqrt{n}$
$F(n) = \Theta(\sqrt{n} \log n)$

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

(vii) $F(n) = 3F(n/2) + n^2$
$F(n) = \Theta(n^2)$

(viii) $F(n) = 3F(n/3) + 3F(2n/3) + n^2$
$F(n) = \Theta(n^3)$

(ix) $F(n) = 3F(n/9) + 1$
$F * n) = \Theta(\sqrt{n})$

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

(xi) $F(n) = 4F(n/2) + n^2$
$F(n) = \Theta(n^2 \log n)$

(xiii) $F(n) = F(3n/5) + 4F(2n/5) + n^2$
$F(n) = \Theta(n^2 \log n)$

(xiv) $F(n) = F(\log n) + 1$
$F(n) = \Theta(log^* n)$

(xv) $F(n) = F(n-1) + n^2$
$F(n) = \Theta(n^3)$

(xvi) $F(n) = F(n/2) + 2F(n/4) + n$
$F(n) = \Theta(n \log n)$

(xvii) $F(n) = F(n/2) + F((n-1)/2) + 3n$
$F(n) = \Theta(n \log n)$

(xviii) $F(n) = F(n/3) + F(n/2) + n$
$F(n) = \Theta(n)$

(xix) $F(n) = F(n/5) + F(7n/10) + n$
$F(n) = \Theta(n)$

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

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

5. Fill in the blanks.

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

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

(iii) **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.

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

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

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

(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 $O(nm \log n)$.

(x) 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 $ab \sim c\,d - e \sim * + *$

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

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

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

(xiv) 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]);
```

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

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

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

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

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

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

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

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

(c) **Binary search** is a divide-and-conquer searching algorithms.

(d) **Mergsort** and **Quicksort** are divide-and-conquer sorting algorithms.

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

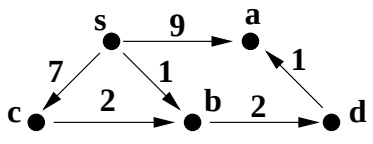
(f) In an open hash table, there is a **search** structure at each table index.

(g) 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.

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

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

8. 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

Figure 1

9. 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

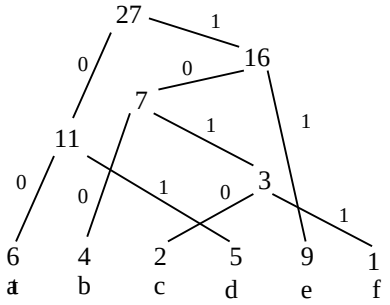


Figure 2

10. 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);
 }
```

The recurrence is $G(n) = G(n-1) + G(n-1)$ for $n >= 1$, G(0) = 1;

$\Theta(2^n)$ is the answer. We can verify this by induction. If $n = 0$, we have george(0) $= 1 = 2^0$. For $n > 0$, the inductive step, for $n > 0$, is george(n) = george(n-1)+george(n-1) $= 2 * 2^{n-1} = 2^n$.

11. 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);
 }
```

$\Theta(n)$, by the master theorem.

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

The recurrence is $T(n) = T(n/2) + 1$, since we allocate one time unit to fecth the compute of n, and thus the time complexity of the computation of martha(n) is $\Theta(\log n)$, by the master theorem.

13. Write a C++ function which solves the simplest coin-row 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 n be the length of the sequence, $x[i]$ the value of the i^{th} coin, and let $A[i]$ the maximum value of any legal subset of the first i coins.

```
A[0] = 0;
A[1] = x[1]
For i from 2 to n
  A[i] = max(A[i-1),A[i-2]+x[i])
Write A[n]
%\end{enumerate}

 \item
 The usual recurrence for Fibonacci numbers is:\\
        $F[0] = 0$\\
        $F[1] = 1$\\
 $F[n] = F[n-1]+F[n-2]$ for $n \ge 2$\\

 However, there is another recurrence:\\
 $F[0] = 0$;
 $F[1] = F[2] = 1$\\
 $F[n] = F\brackd{\frac{n-1}{2}}*F\brackd{\frac{n}{2}}+
 F\brackd{\frac{n+1}{2}}*F\brackd{\frac{n+2}{2}}$ 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 assume procedures store(n,x), which stores $F[n] = x$ into the
search structure and fetch(n)
which fetches the value of $F[n]$ if it's in the structure, and otherwise
returns a default value (that could be -1, for example).

\begin{verbatim}
int F(int n)
 {
  assert(n >= 0);
  int temp = fetch(n);
  else if(temp == default)
   {
    if(n < 2) temp = n;
    else temp = F((n-1)/2) + F(n/2) + F((n+1)/2) + F((n+2)/2);
    store(n,temp);
   }
  return temp;
 }
```

14. 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.

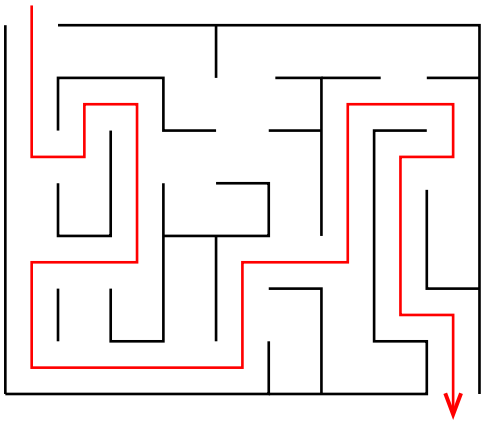


Figure 3

Use a queue to execute BFS. The maze is a graph, where the center of each square is a vertex, and two vertices are neighbors if there is no wall between their squares. You could also use Dijkstra's algorithm or A^*, but I claim that the simple version of BFS is fastest.

15. 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 and n ¡ N); int indx = n*(n+1)/2+k; return indx;

16. 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.

A[i][j][k] will be at address $180 * i + 18 * j + k$

Read This!

Here is a more general formulation. Suppose that a 3-D array A[M][N][P] is stored in row-major order. How many predecessors does A[i][j][k] have? Answer: $N * P * i + P * j + k$.

On the other hand, if A[M][N][P] is stored in column-major order, A[i][j][k] has $M * N * k + M * j + i$ predecessors.

17. 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].

$4096 + 2 * (3 + 7 * 8 + 4 * 8 * 9) = 4790$

18. 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 $A[4][2][1]$?

    In general, the number of predecessors of $A[i][j][k]$ is $\binom{i+2}{3} + \binom{j+2}{2} + k$. We have $i = 4$, $j = 2$, and $k = 1$, and thus the number of predecessors is $\binom{4}{3} + \binom{2}{2} + 1 = 20 + 3 + 1 = 24$ The address in memory is 1024 + 24 = 1048.

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.

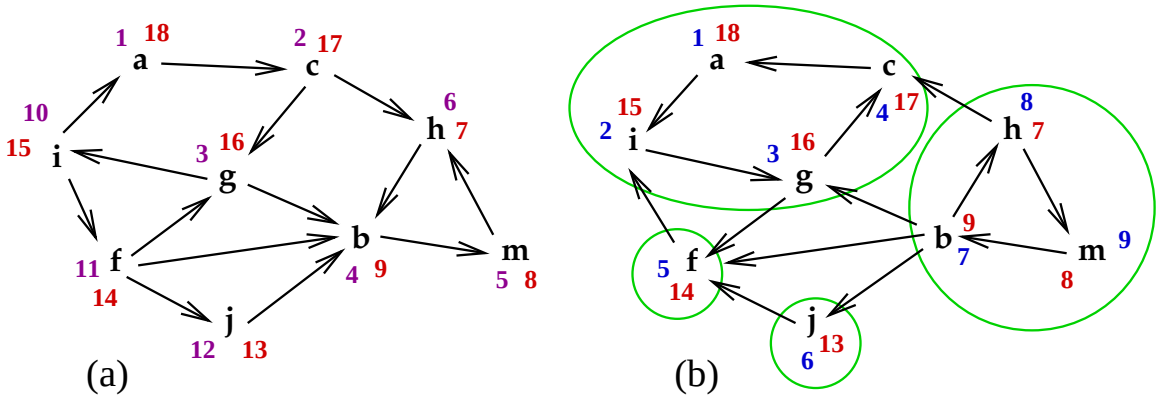


Figure 4

21. List properties of a good hash function.

    Must be deterministic.
    Must be fast to compute.
    Must be spread out .
    Must not have locality.

22. 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
```

24. 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 two adjacent coins are selected. Your code should identify the coins which are selected.

A[i] = maximum weight of legal sequence ending at x[i]
back[i] = coin just before x[i] in sequence ending at x[i]

```
A[1] = x[1]\\
back[1] = default\\
A[2] = x[2]\\
back[2] = default\\
A[3] = A[1]+x[3]\\
back[3] = 1\\
For all i from 4 to n
  if(A[i-3] > A[i-2])
    A[i] = A[i-3] + x[i]
    back[i] = i-2
  Else
    A[i] = A[i-2] + x[i]
    back[i] = i-2
If(A[n-1] < A[n]) last = n-1
Else last = n
```

The optimal set of coins can be recovered by using the backpointers back[i].

writebest(int i) // writes the best subsequence ending at x[i] if(back[i] not default) writebest(back[i]) write x[i]

The best set of coins is written by executing writebest(last)

Here is an alternative algorthm, which is equally easy to understand, I claim.

$A[i]$ is the maximum value of a legal subset of the first i coins. $B[i]$ is the maximum value of any legal subset that ends at the i^{th} coin.

```
A[0] = 0
B[1] = x[1]
A[1] = B[1]
For all i from 2 to n
   B[i] = x[i] + A[i-2]
   A[i] = max(A[i-1],B[i])
```

It's more complicated to find the backpointers. Can you do it?

25. 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 identify the coins which are selected.

Let A[i] be the maximum weight of any legal subsequence of the first i coins. Let B[i] be the maximum weight of any legal subsequence which contains the i^{th} coin, but not the $(i-1)^{st}$ coin.

```
A[0] = 0
B[1] = x[1]
A[1] = x[1]
for i from 2 to n
  B[i] = A[i-2] + x[i]
  A[i] = max(A[i-1],B[i-1]+x[i],B[i])
```

The answer is A[n].

26. Write an algorithm to compute x^n where n is a non-negative integer and x is a real (floating point) number.

    This recursive algorithm is different from the one I usually write on tests, but I claim it's easier to understand.

```
input conditions: n >= 0 and x not equal to 0.
power(float x, int n) // computes x to the power of n
  if(n == 0) return 1.0
  else if(n is odd) return x*power(x,n-1)
  else return the square of power(x,n/2)
```

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

AC BX FR EY GMQS N DZ

ACFR GMQS DZ
BX EV N

ABCFRX DNZ
EGMQSV

ABCEFGMQRSVX
DNZ

ABCDEFGMNQRSVXZ

28. 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 is odd
    m = m/2;
    y = y+y;
   }
  return z;
 }
```

The loop invariant is n∗x = m∗y + z

29. 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 V[i,i] = 0
For all i
  For all j
    V[i,j] = W[i,j]
    B[i,j] = i
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
        B[i,k] = B[j,k]
```

30. 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(s.size > 0);
   s.size--;
   return s.item[s.size];
  }
```

31. 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 on the left. Show the evolution of the structure when deletemin is executed.

32. Starting from the array given in the first line of the figure on the right, show the evolution of the structure when B is inserted.

33. 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.

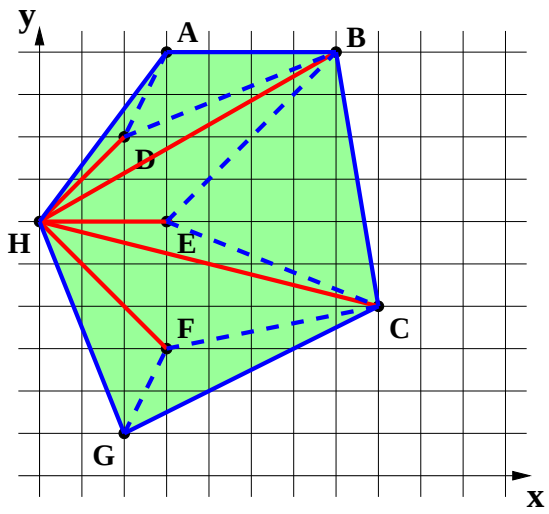


Figure 5

35. 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	5	4	4	3	4
g	7	6	5	6	6	5	5	5	5

The edit distance is 5.

37. 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.

In the figure below, a deletion occurs after the insertion.

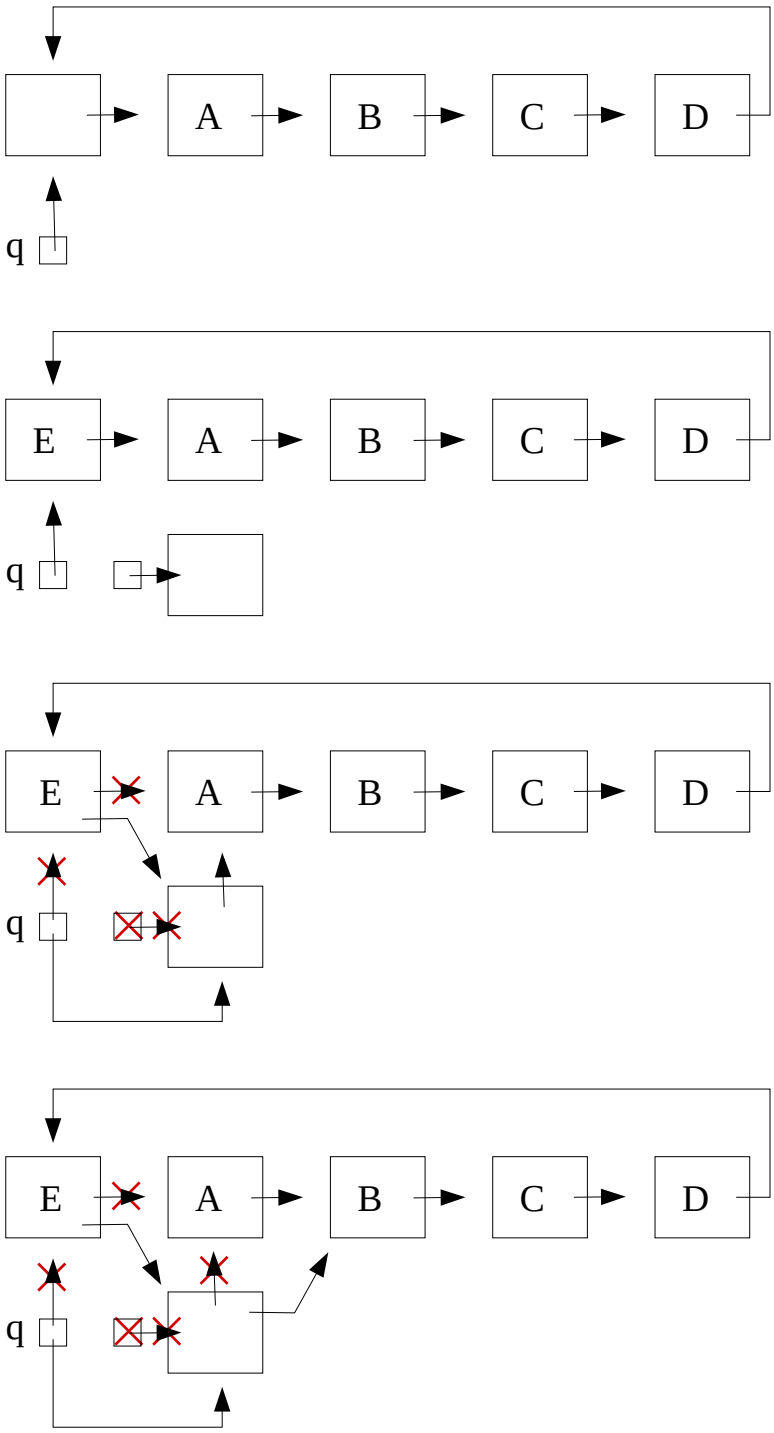


Figure 6

38. 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.

    There are two ways to set this problem up. I want you to use the right-to-left method, not the left-to-right. There is one subproblem for each vertex v, namely to compute $M[v]$, the maximum weight of any

directed path starting at v. Compute the forward pointer `forw`$[v]$ for each vertex. Explain how those pointers are used to find the path.

Give each vertex a name, $1 \ldots n$ in topological order. Assume all weights are non-negative.

Let Out(v) be the set of out-neighbors of a vertex v.

For all i from n to 1 in decrasing order
    If Out$(i) = \emptyset$
        $M[i] = 0$
        `forw`$[i]$ = default
    Else
        For all $y \in$Out(i) temp $= w(i, y)+$ M(y)
            If temp $<$ M(i)
                M(i) = temp
`forw`$(i) = y$

To find the maximum weight path, start with the vertex for which M is maximum, then follow the forward pointers until you reach a vertex with no out-neighbors.

40. 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]);
 }
```

41. Find the longest **monotone** (not strictly monotone) subsequence of the sequence

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

    For any i, let length(i) be the length of the longest path ending at the i^{th} term of the sequence. The terms will be placed in columns, one for each possible length.

    Here is the algorithm. Place the first term in column 1. Visit the remaining terms in order. Let m be the next term. Find the leftmost column, if any, whose last (bottom) entry is greater than m, then add m to that column. If the bottom of each column is less than or equal to m, start a new column, where m is the top entry. If m is in the leftmost column, no pointer is needed. Otherwise, draw a points (the backpointer) from m to the bottom entry of the previous column.

    Starting from an entry in the rightmost column, follow the backpointers to recover a maximal length monotone increasing subsequence.

Figure 7 shows the columns and the pointers. The longest monotone increasing subsequence is 1,5,6,8,10,15.

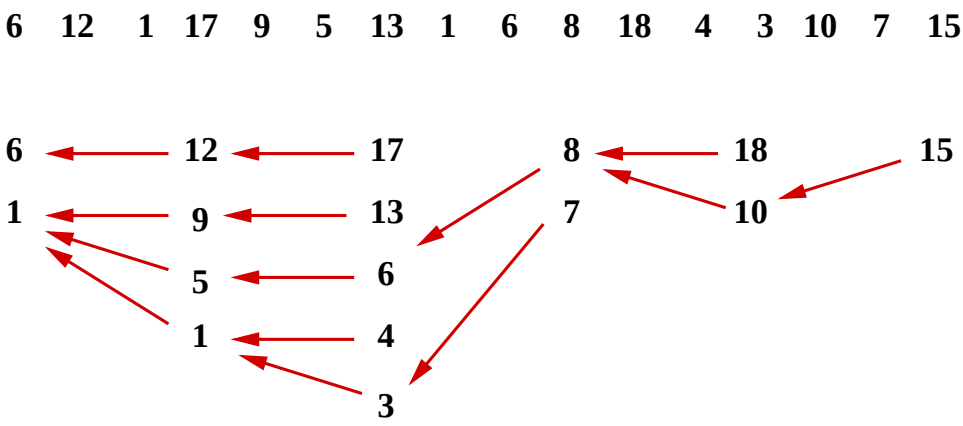


Figure 7

Figure 8 also illustrates the algorithm. First the terms are plotten in a graph. A dot at point (x, y) inicates that the x^{th} term of the sequence is y. The red arrows are the backpointters, just as in Figure 7. The blue arrows connect the entries in the same columnh. The thick blue curves separate the dolumns.

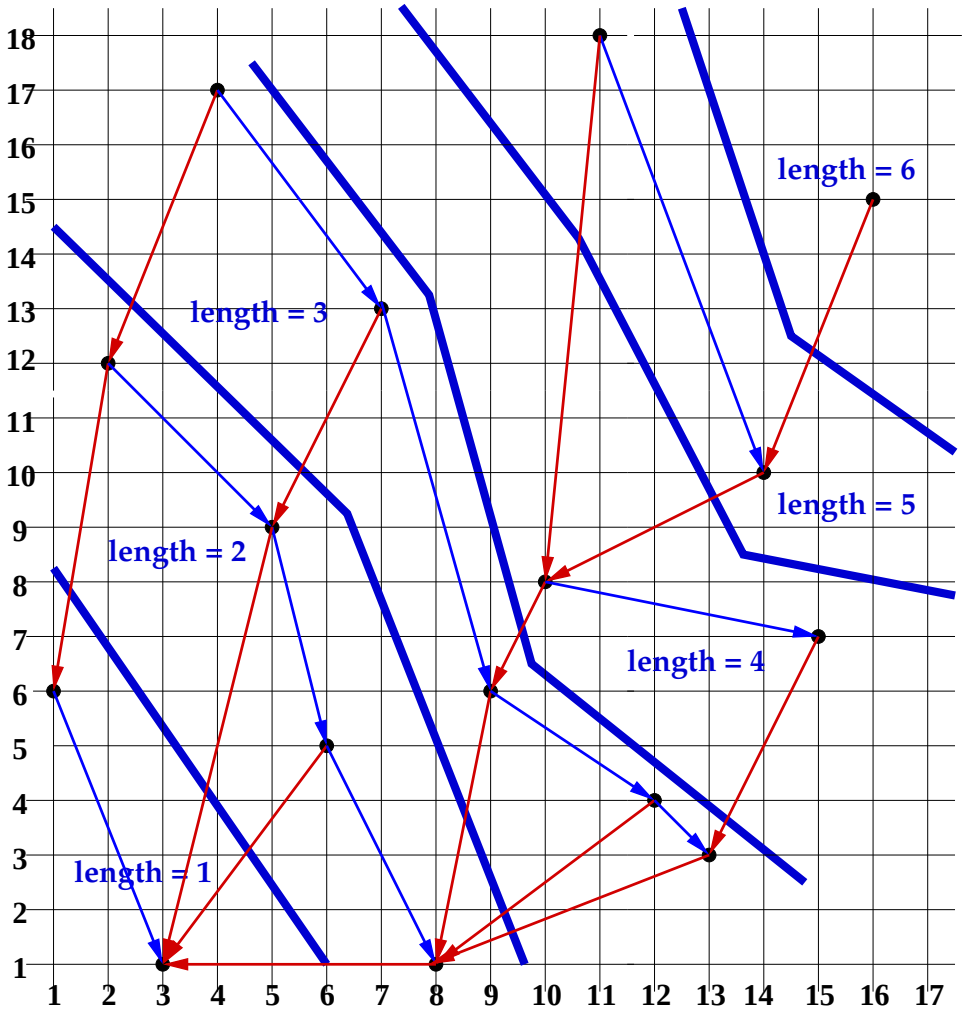


Figure 8

44. 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 $x_j, y_j)$ has weight W[j]. Be sure to include the shortcut that ends the program when the final values have been found.

    The output consist of two arrays: V and B, where V[i] is the minimum cost of any directed path from 0 to i and B[i] is the backpointer, the next-to-the-last vertex on some minimum weight path from 0 to i. We let B[0] = default.

    We need to consider the number of edges in a path, not just the total weight. The loop invariant is that, after t iterations of the outer loop, $V[i]$ is less than or equal to the minimum weight of any path from 0 to i which has no more than t edges.

    For i from 0 to n
        B[i] = default
        V[i] = ∞
    V[0] = 0
    changed = true
    For (t = 0; t < n and changed; t++)
        changed = false
        For j from 1 to m
            temp = V[x_j]+ W[j]
            If temp < V[y_j]
                V[y_j] = temp
                B[y_j] = x_j
                changed = true
    If(changed) "The graph has a negative weight cycle."

45. 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;
 }
```

`mystery(x,n)` computes x^{4n}. The loop invariant is $x^{4n} = y^m * z$.

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

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
H	L	D	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

47. 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.

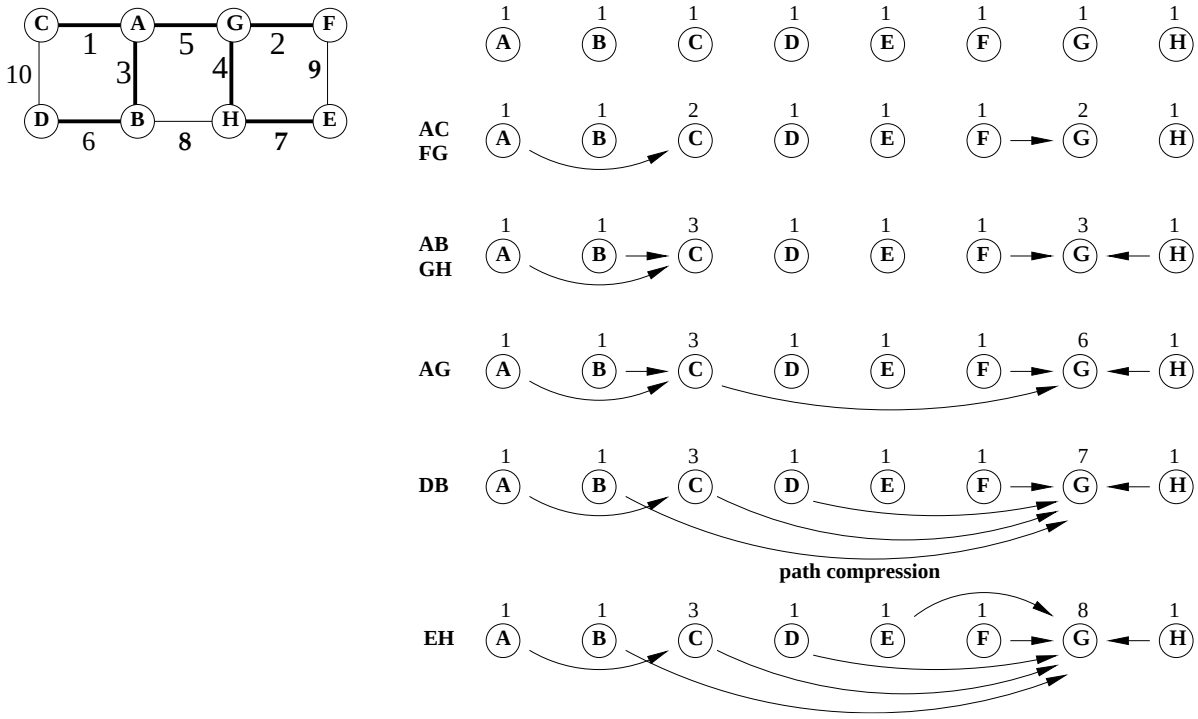


Figure 9

48. 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.

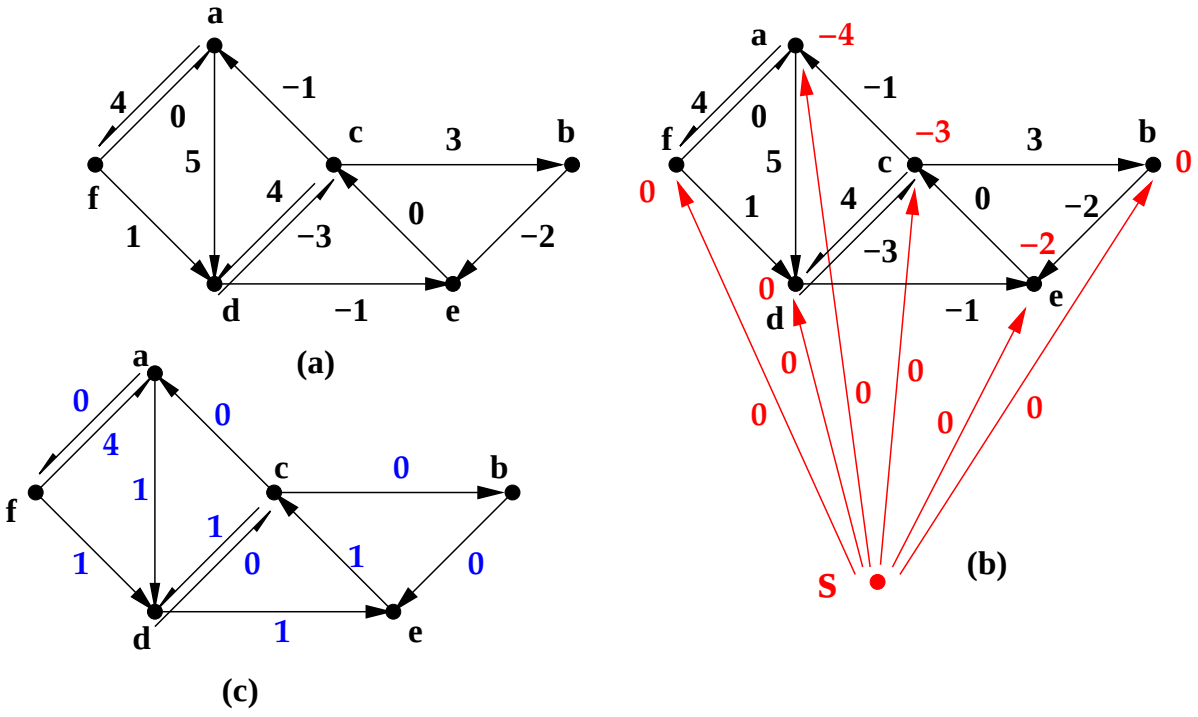


Figure 10

49. Find the asymptotic time complexity of each code fragment, in terms of n. Use Θ if possible.

(i)
```
for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j++)
```

$\Theta(n^2)$

(ii)
```
for(int i = 1; i < n; i++)
  for(int j = i; j < n; j++)
```

$\Theta(n^2)$

(iii)
```
for(int i = 1; i < n; i = 2*i)
  for(int j = 2; j < i; j = j*j)
```

$\Theta(n \log \log n)$

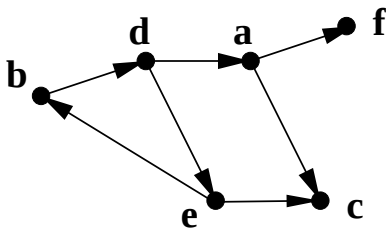
50. You are trying to construct a cuckoo hash table of size 10, where each of the 10 names listed below has the two possible hash values indicated in the array. Put the items into the table, if possible; otherwise, convince me it's impossible. Instead of erasing ejected items, simply cross them out, 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	
1	
2	
3	
4	
5	
6	
7	
8	
9	

The set of 7 data: {Ben,Deb,Eve,Gus,Hal,Ike,Jan} has only 6 hash values altogether: {0,1,2,3,7,8}. Thus, there is no solution.

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



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

Figure 11

52. 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 $4 - 3 + 2 + 7 - 5 + 3 + 4 = 12$.

    I have not gone over this algorithm in class. But you can do it anyway.

    $X[1] = max\{0, x_1\}$
    For i from 2 to n
    {
    $Y[i] = x_i + max\{0, Y_{i-1}\}$
    $X[i] = max\{Y[i], X[i-1]\}$
    }
    Write $X[n]$

53. 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.

54. Walk through the A^* algorithm to find the least cost path from S to T. The values of the heuristic function are indicated in red.

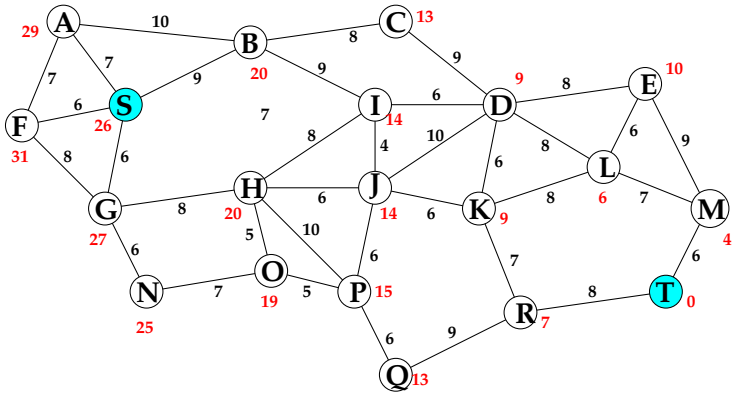
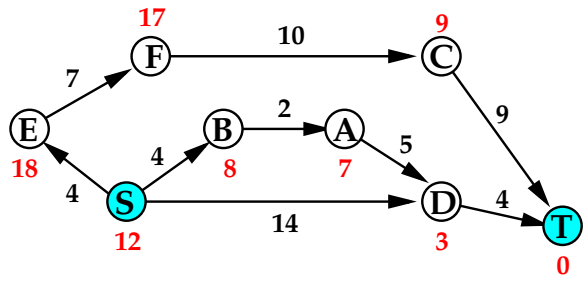


Figure 12

55. 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.



.

Figure 13

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.

56. Compute the Levenshtein edit distance from the word "proven" to the word "shore." Show the matrix.

		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.

57. Find 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.

The algorithm for the strictly monotone case is almost identical to the algorithm for the monotone case. The only difference is that, in the strictly monotone case, a term may be placed in a column under another term of the same value, as shown here.

```
1 5 4 8
2 7
2
```

There are two maximum length strictly monotone increasing subsequences: 1,2,4,7 and 1,2,4,8

58. 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])
```

59. THIS IS NOT ACTUALLY A PROBLEM HERE HERE This is a memoization problem. Values are stored in a sparse array A, whose indices and values are integers. Values can be retrived by executing the function `fetchA(int n)` which returns the value of $A[n]$ if it exists, otherwise returns `default`, and the procedure `store(n,x)` stores the memo (n, x), meaning that $A[n] = x$, into the search structure.

60. 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 $F(n)$?

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

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;
}
```

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

    Ans: $\Theta(\log n)$