Answers to Study for CS477 Final Examination May 6, 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) $\log_2 n = O\ (\log_4 n)$

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

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

   (ix) **T** Heapsort can be considered to be a sophisticated implementation of selection sort.

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

3. Fill in the blanks.

   (i) If a planar graph has 7 edges, it must have at least **5** vertices. (You must give the best possible answer, exactly. No partial credit.)

   (ii) The height of a binary tree with 17 nodes is at least **5**. (You must give the best possible answer, exactly. No partial credit.)

   (iii) 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]);
```

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

(v) 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.)

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

(vii) The postfix expression $zw + x \sim y - *$ is equivalent to the infix expression $(z + w) * (-x - y)$ and the prefix expression $* + zw - \sim y$

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

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

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

(xi) A **perfect** hash function fills the hash table exactly with no collisions.

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

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

(xiv) **quicksort** and **mergesort** are divide-and-conquer algorithms that we've studied this semester.

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

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

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

(xviii) If a directed acyclic graph has n vertices, it must have n strong components.

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

(xx) 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**.

(xxi) The following is pseudo-code for what algorithm? HERE **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]);
```

(xxii) 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 $ab \sim cd-e \sim *+*$

(xxiii) 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 = 1; i < n; i = i+i)
  cout << "Hello world" << endl;
```

$\Theta(\log n)$

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

$\Theta(\log\log n)$

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

$\Theta(n\log n)$

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

$\Theta(n)$

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

$\Theta(\sqrt{n})$

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

$\Theta(n)$

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

$\Theta(n\log n)$

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

$\Theta(\log \log n)$

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

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

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

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

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

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

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++.

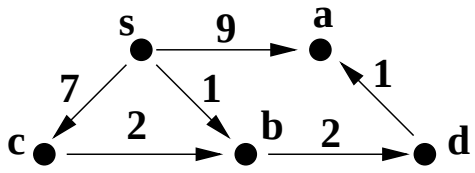
Using that recurrence, Describe a $\Theta(\log n)$-time memoization algorithm which reads a value of n and computes $F[n]$, but computes only $O(\log n)$ intermediate values.

Here is pseudocode for that algorithm. We assume a search structure storing pairs of the form (i,F(i)), where the first component is the key. We also assume division is truncated, as in C++.

```
store the pair (1,1)
store the pair (2,1)
int F(int i)
    search for a pair (i,x) in the structure.
    if that pair is found
        return x.
    else
        x = F((i-1)/2)*F(i/2) + F((i+1)/2)*F((i+2)/2)
        store the pair (i,x)
        return x.
```

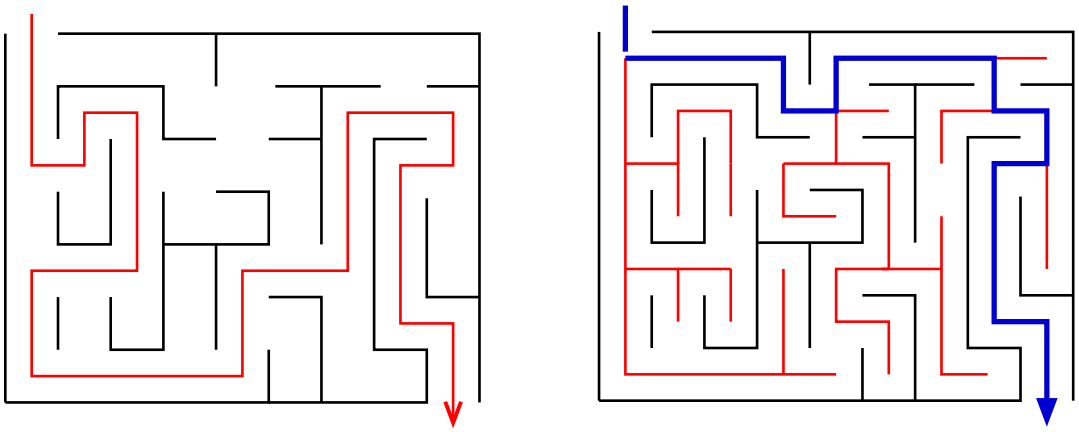
If n is positive, evaluation of $F(n)$ will requre $\Theta(\log n)$ pairs to be stored in the search structure. If it takes $O(\log \log n)$ to store or fetch, the time complexity of the algorithm to compute $F(n)$ for a given n would be $O(\log n \log \log n)$.

7. 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

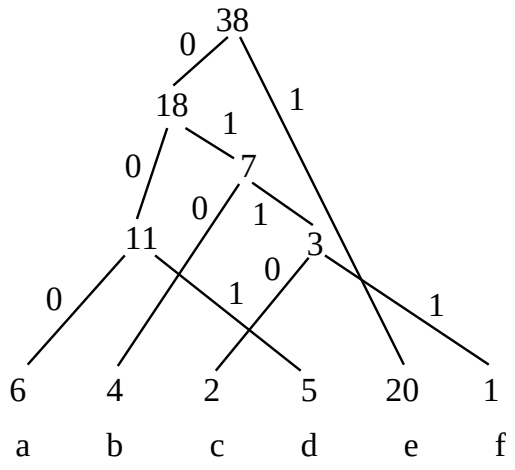
8. The figure below shows an example maze. The black lines are walls. You need to find the shortest path, avoiding the walls, from the entrance at the upper left and the exit at the lower right. The red path shows one such path, although it is not the shortest. Describe a program to find the shortest 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.



First, let G be the (undirected) graph show vertices are cells of the figure, and where two vertices are adjacent of there is no wall between the cells. The start vertex is the upper left cell and the target vertex is the lower write cell. We use depth first search, which gives us a tree rooted at the start vertex, and also gives the shortest path to the target.

9. 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	000
b	4	010
c	2	0110
d	5	001
e	20	1
f	1	0111



10. 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;
 }
```

$x * n = y * m + z$

11. (i) 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	3	2	3	4	5
d	4	3	3	3	3	3	**2**	3	4
a	5	4	4	4	4	4	3	3	4
f	6	5	5	5	5	5	4	3	4
g	7	6	5	6	6	6	5	4	4

(ii) The entry **2** in row 4, column 6 is the solution to one subproblem. State that subproblem.

12. 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;
 }
```

13. List properties of a good hash function.

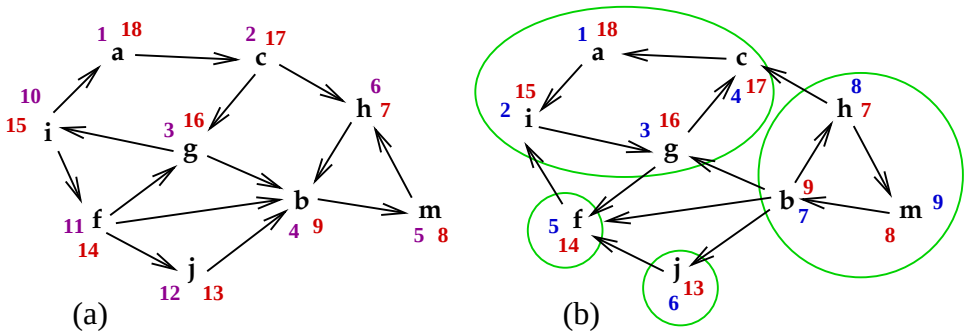
    (i) Deterministic.

    (ii) Easy to compute.

    (iii) Unrelated to any actual property of the data.

    (iv) Uniformly distributed over the data, or close to it.

14. Use the DFS method to find the strong components of the digraph shown below as (a). Use (a) and (b) for your work.



The algorithm executes depth first search twice: first on the original graph, and then on the inverse graph. During DFS, each vertex is pushed onto the stack once and is popped off the stack once Each of these $2n$ stack operations take one time step. Each vertex is assigned two numbers: its **pre** number, the time it is pushed onto the stack, and its **post** number, the time it is popped. We repeat the following steps until all vertices are visited and the stack is empty.

If the stack is empty, push the highest priority unvisited vertex.

If the stack is not empty, push an unvisited neighber of the top vetex if there is one.

If the stack is not empty and the top vertex has no unvisited neighbor, pop the stack.

Each vertex is now labeled with two numbers, its **pre** and **post** numbers, written in magenta and red, respectively in (a).

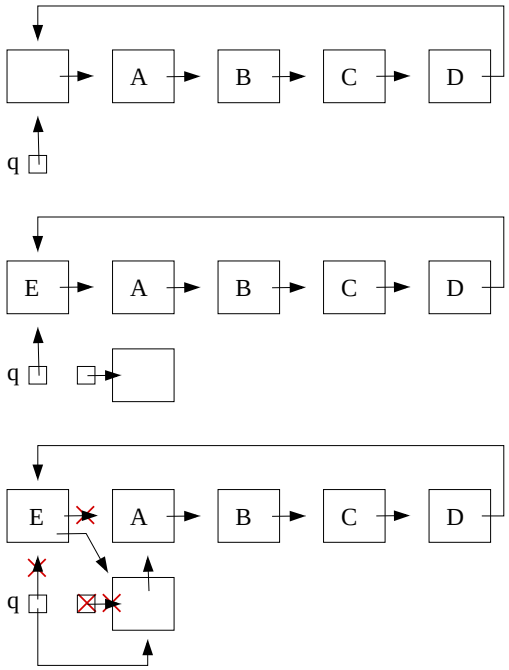
Now construct the inverse graph by reversing all arrows, as shown in (b). Each vertex is labeled with its **post** number from the previous DFS. Priority is the reverse of this label. We exectue DFS on the reverse graph. In our example, the vertex **a** has highest priority.

Each time we pop a vertex in this second DFS, we pop all the items currently on the stack, and these vertices constitute a strong component. In (b) each strong component is enclosed in a green oval.

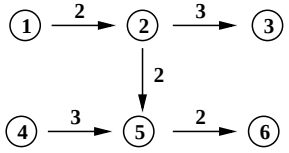
Here is a step by step description of our computation. During the first DFS, priority is arbitrary, but we use alphabetical order.
Push **a**: **pre** = 1
Push **c**: **pre** = 2
Push **g**: **pre** = 3
Push **b**: **pre** = 4
Push **m**: **pre** = 5
Push **h**: **pre** = 6
Top = **h** has no unvisited out-neighbor.
Pop **h, m, b**: **post** = 7, 8, 9
Push **i**, highest priority vertex with an unvisited out-neighbor. **pre** = 10
Push **f**: **pre** = 11
Push **j**: **pre** = 12
Top = **j** has no unvisited out-neighbor.
Pop **j, f, i, g, c, a**: **post** = 13, 14, 15, 16, 17, 18
Stack is empty and all vertices have been visited.

Push **a**, **pre** = 1
Push **i**, **pre** = 2
Push **g**, **pre** = 3
Push **c**, **pre** = 4
Top = **c** has no unvisited out-neighbor.
Pop **c, g, i, a** These make a strong component.
Push **f**, the highest priority unvisited vertex, **pre** = 5
Top = **f** has no unvisited out-neighbor.
Pop **f**, another strong component.
Push **j**, the highest priority unvisited vertex, **pre** = 6
Top = **j** has no unvisited out-neighbor.
Pop **j**, another strong component.
Push **b**, the highest priority unvisited vertex, **pre** = 7
Push **h**, **pre** = 8
Push **m**, **pre** = 9
Top = **m** has no unvisited out-neighbor, and there are no unvisited vertices.

15. 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.



16. 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. For example, in the digraph shown below, the maximum weight directed path is (1,2,5,6). Assume that $i < j$ for any arc (i, j), as in the figure. You need not write pseudo-code if you can explain the algorithm without it.



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 $\texttt{forw}_i$ for each vertex v_i Explain how those pointers are used to find the path.

Let $v_1, \ldots v_n$ be the vertices in topological order. For each i, define A_i to be the meximum length directed path in G starting at v_i. The following algorithm computes A.

For all i from n down to 1

If v_i has no outneighbors, let $A_i = 0$

Else choose v_j an outneighbor of v_i for which A_j is maximum, and define $A_i = w(v_i, v_j) + A_j$ and $\text{forw}_i = j$.

The following code writes the largest weight directed path in G

Choose i for which A_i is maximum.
Write i
While$(A_i > 0)$
$j = \text{forw}_i$
write j
$i = j$

17. Walk through mergesort with the array given below.

```
          VJATNLDQMEFSPWGL
     VJATNLDQ          MEFSPWGL
  VJAT     NLDQ     MEFS     PWGL
JV   AT   NL   DQ   ME   FS   PW   GL
JV   AT   LN   DQ   EM   FS   PW   GL
 AJTV     DLNQ     EFMS     GLPW
    ADJLNQTV          EFGLMPSW
         ADEFGJLLMNPQSTVW
```

18. Write pseudocode for the Floyd/Warshall algorithm.

Let the vertices be $1, \ldots n$. Let $W(i,j)$ be the weight of the arc from i to j, or ∞ if there is no such edge. We compute $V(i,j)$ to be the smallest weight of and directed path from i to j, and $B(i,j)$ be the backpointer at j of that path.

For all i and all j
    $V(i,j) = W(i,j)$
    $B(i,j) = i$
For all i from 1 to n
    $V(i,i) = 0$
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
                $B(i,k) = B(j,k)$

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

Let the vertices be $0, 1, \ldots n$ where 0 is the source. Let $e_1, \ldots e_m$ be the arcs, where $e_k = (s_k, t_k)$ and has weight (length) w_k. We compute V_i to be the length of the shortest path from 0 to i for all i, and the backpointer b_i for $i \neq 0$.

$V_0 = 0$ for all $i > 0$
    $V_i = \infty$
bool done = false
while(not done)
    done = true
    for all k from 1 to m
        $i = s_k$
        $j = t_k$
        $j = t_k$
        temp $= V_i + w_k$
        if(temp $< V_j$)
            $V_j =$ temp
            $b_j = b_i$
            done = false

20. 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)`?

$\Theta(n^2)$

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

$\Theta(n)$

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

$\Theta(n)$

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

$\Theta(\log n)$

21. 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.

Let x_i be the value of the i^{th} coin. We let A_i be the maximum value of any legal subsequence which contains the i^{th} coin.

$A_1 = x_1$
$A_2 = \max(A_1, A_2)$
$A_3 = A_1 + x_3$
For i from 4 to n
$\quad A_i = \max(A_{i-2}, A_{i-3}) + x_i$

The answer is $\max(A_n, A_{n-1}$.

We now compute backpointers. Let back(3) = 1. For $i \geq 4$ let back$(i) = \begin{cases} i-2 \text{ if } A_{i-2} > A_{i-3} \\ i-3 \text{ otherwise} \end{cases}$

Let $\ell = n$ if $A_n > A_{n-1}$, and $n-1$ otherwise.

The optimal subsequence ends with x_ℓ and can be computed by following backpointers starting from ℓ and ending at 1 or 2.

22. Write pseudocode for a function `float power(float x, int n)` that returns x^n. You may assume that $x \neq 0$, but your code must work for any integer n. It is not necessary to use the algorithm given in class; use any $O(\log n)$ time algorithm.

Here is a solution using recursion.

```
float power(float x, int n)
 {
  if(n == 0) return 1.0;
  else if(n < 0) return power(1/x,-n);
  else if(n%2) // that is, n is odd
   return x*power(x,n-1);
  else // n is even, but not 0
   return power(x*x,n/2);
 }
```

Here is a solution using the method we did in class.

```
float power(float x, int n)
 {
  if(n < 0)
   {
    x = 1/x;
    n = -n;
   }
  float z = 1.0;
  float y = x;
  int m = n;
  while(m > 0)
   {
    if(m%2) // that is, m is odd
     z = z*y;
    m = m/2; // truncated
    y = y*y;
   }
  return z;
 }
```

23. Fill in the blanks.

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

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

(iii) The recurrence $F(n) = F(n/5) + F(7n/10) + n$ is used to compute the time complexity of **BFPRT**, a selection algorithm which is sometimes called the "median of medians" algorithm.

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

(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 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.

Assume that A[n] is the last entry in the array.

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

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

(viii) 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.

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

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

ACBXFREYGMQSNDZ

ACFRGMQADZ

BXEYN

ABCFRXADNZ

EGMQY

ABCEFGMQRXY

ADNZ

ABCDEFGMNRXYZ

26. 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)
 {
  // input condition: s.size < N
  s.item[s.size] = i;
  s.size++;
 }
 bool empty{s&stack)
  {
   return size == 0;
  }
 int pop(s&stack)
  {
   // input condtion: s.size > 0
   int rslt = s.itm[s.size-1];
   s.size --;
  }
```

27. 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. The offset of A[i][j][k] is $i * 10 * 10 + j * 10 + k$ Thus the address of A[i][j][k] in main memory is $B + 100i + 10j + k$.

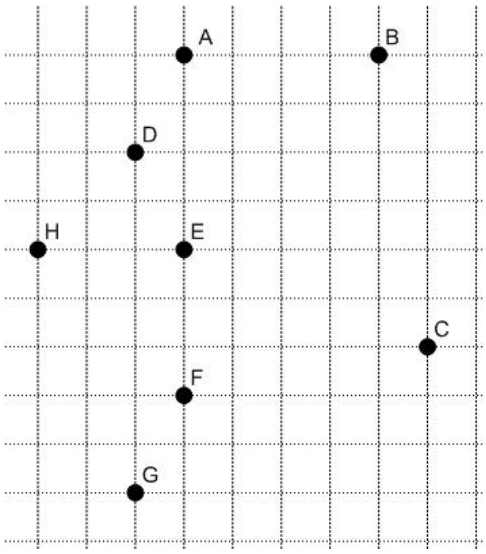
28. In class, we implemented a minheap as an almost complete binary tree implemented as an array. Suppose the minheap is initialized as shown in the first line of the array shown below. Show the evolution of the structure when deletemin is executed.

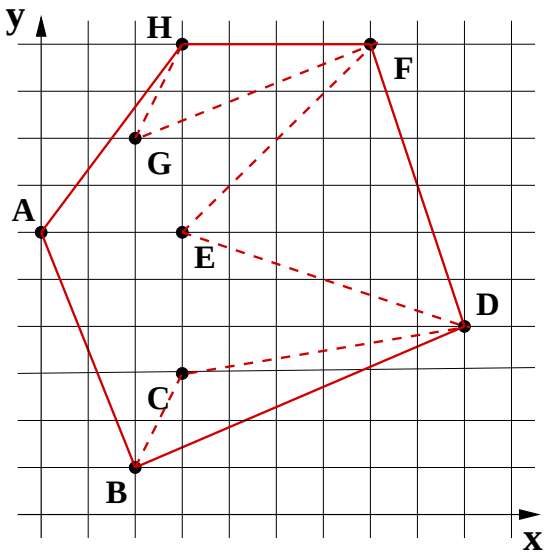
A	C	F	D	Q	H	L	R	Z
Z	C	F	D	Q	H	L	R	
C	Z	F	D	Q	H	L	R	
C	D	F	Z	Q	H	L	R	
C	D	F	R	Q	H	L	Z	

29. 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

30. 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

For this example, I decided to sort the points by slope of the line from the base point. I relabeled to points to be in slope order, then used the original Graham Scan algorithm.

31. 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.

    The answer to this problem is in hw7ans.

32. I have decided to delete the Longest Monotone Subsequence Problem from the study guide and it won't be on the final exam.