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

Answers to Study for Examination April 9, 2025

1. True or False.

   (a) **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.

   (b) **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.

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

   (d) **T** Open hashing uses probe sequences.

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

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

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

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

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

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

2. Solve each of these recurrences, giving an asymptotic answer.

   (a) $F(n) = F(5n/13) + F(12n/13) + n$

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

   (b) $F(n) = F(n-1) + 2F(n-2)$

   Assume $F(n) = C^n$ Then $C^2 = C + 2$, solve the quadratic equation: $C = 2$.
   $F(n) = \Theta(2^n)$

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

   $$5\left(\frac{2}{3}\right)^4 + \left(\frac{1}{3}\right)^4 = 1$$

   $F(n) = \Theta(n^4 \log n)$.

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

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

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

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

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

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

(g) $H(n) = 4H(2n/5) + H(3n/5) + 2n^2$

$4(2/5)^2 + (3/5)^2 = 1$. Therefore $H(n) = \Theta(n^2 \log n)$. The coefficient on the last term of the recurrence is asymptotically irrelevant, since it is a positive constant.

(h) $G(n) = 4(G(n/2) + 5n^2$

$4(1/2)^2 = 1$, therefore $G(n) = \Theta(n^2 \log n)$. The coefficient on the last term of the recurrence is asymptotically irrelevant, since it is a positive constant.

(i) $F(n) = F(n - \log n) + \log^2 n$

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

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

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

(k) $G(n) = G(4n/5) + G(3n/5) + n^2$

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

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

$\left(\frac{3}{10}\right)^\gamma + \left(\frac{1}{5}\right)^\gamma < 1$ Therefore
$T(n) = \Theta(n^\gamma) = \Theta(n)$

3. Fill in the blanks.

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

(b) Name three kinds of search structurees.

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

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

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

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

(f) **(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.

(g) In an open hash table, there is a **search structure** at each table index.
Pick one of these answers:
heap
stack
search structure

(h) 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.

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

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

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

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

4. Write a recurrence for the time complexity of the code below, and solve that recurrence, giving an asymptotic function of n.

```
void george(int n)
 {
  if(n > 1)
   {
    for(int i = 0; i < n; i++) cout << "hello" << endl;
    george(n/2); george((n+1)/2);
   }
 }
```

$T(n) = T(n/2) + T((n+1)/2) + n$

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

5. 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].

The number of predecessors of A[5][13][7] is $7*20*10+13*10+5$ The offset is 2 times that. The address of that item in main memory is $1024 + 2 * (7 * 20 * 10 + 13 * 10 + 5) = 4094$

6. 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

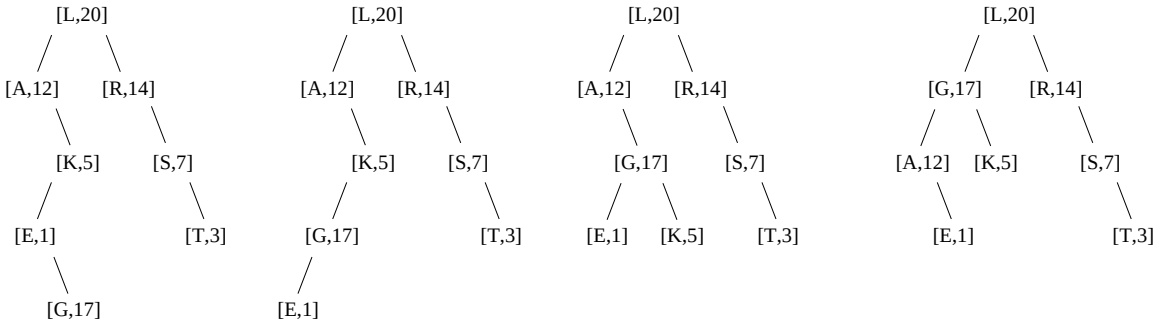
7. Let $\sigma = x_1, x_2, \ldots x_n$ be a sequence of numbers with both positive and negative terms. Write an $O(n)$ time dynamic program which finds the maximum sum of any contiguous subsequence of σ. For example, if the sequence is $-1, 4, -3, 2, 7, -5, 3, 4, -8, +6$ then the answer is $4 - 3 + 2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

   $X[i]$ = maximum sum of any legal subsequence of the first i terms of the sequence. $Y[i]$ = maximum sum of any legal subsequence which ends at x_i.

   $X[0] = 0$
   $Y[1] = x_1$
   For i from 1 to n
   $X[i] = \max\ (X[i-1], Y[i])$
   $Y[i] = x_i + \max\ (0, Y[i-1])$

   The maximum sum of any legal subsequence is $X[n]$.

8. 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 *key*, a letter, and the second component is a the *priority*, a random integer. Insertion of the letter G, where the priority is chosen (at random) to be 17. Show the steps.



9. 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.

10. Consider the following C++ function.

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

    (a) The function f is written as a recursive C++ function below. What is the asymptotic value of f(n)?

    $\Theta(n^2)$

    (b) What is the asymptotic time complexity of the computation of f(n) using the recursive code? Assume that each addition and each multiplication takes one time step.

    $\Theta(n)$

    (c) What is the asymtotic time complexity of a computation of f(n) using memoization? (Hint: it's a power of log n.) $\Theta(\log^2 n)$

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

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

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

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

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

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

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

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

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

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

13. Let F be computed by the code:

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

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

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

    What is the time of the code?

    Use the recurrence $T(n) = T(n/3) + 2T(n+1)/3) + 1$.

    Asymptotically, that's equivalent to $T(n) = 3T(n/3) + 1$.

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

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

    4*5 + 3 = 23

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

    The offset is 24, hence the address is 1048.

16. 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.")

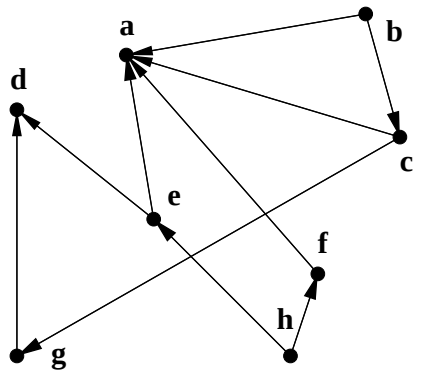
Let the vertices be V[1] ... V[n] in topological order. Let W[i][j] be the weight of the arc from V[i] to V[j], if that arc exists. We compute A[i] to be the maximum weight of any arc which ends at V[i].

Let InNb[i] be the set of all indices of in-neighbors of V[i].

The following code calculates the maximum weight of any path through G, but does not recover the actual path:

```
for all i from 1 to n
  if (InNb[i] = empty set)
    A[i] = 0
  else
    A[i] = max{j in InNb[i]}{A[j] + W[j][i]}
result = max_{1\le i\le n}{A[i]}
return result
```

17. The directed graph shown below can be stored as an array of out-neighbor lists as shown. Write the array of in-neighbor lists for the same graph.



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	

18. 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;
 }
```

(a) What is the asymptotic complexity of $f(n)$ 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, $f(n) = \Theta(n \log n)$.

(b) What is the asymptotic time complexity of the execution of this code as a as an asymptotic 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 $\Theta(n)$

19. Find the Levenshtein edit distance from the word "mennoover" to the word "maneuver." 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.

20. 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])
```