University of Nevada, Las Vegas Computer Science 477/677 Fall 2026

Answers to Examination March 11, 2026

1. True or False.

   (a) [5 points] **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.

   (b) [5 points] **F** Open hashing uses open addressing.

   (c) [5 points] **F** Open hashing uses probe sequences.

   (d) [5 points] **T** False overflow for a queue can be avoided by implementing the queue as a circular list.

   (e) [5 points] **T** If a stack is implemented as a linked list, the head of the linked list should hold the top item of the stack.

   (f) [5 points] **F** Heapsort is a dynamic programming algorithm.

   (g) [5 points] **T** A hash function should appear to be random, but cannot actually be random.

   (h) [10 points] **F** The worst case time complexity of quicksort is $O(n \log n)$

2. Fill in the blanks.

   (i) [5 points] Any comparison-based sorting algorithm on a file of size n executes $\Omega(n \log n)$ comparisons. (Use Ω notation.)

   (ii) [15 points] The asymptotic time complexity to find an item in an ordered array of length n is $O(\log n)$ using the divide and conquer algorithm **binary search**.

   (iii) [10 points] In a priority queue, each item in the structure represents an **unfulfilled obligation**.

   (iv) [5 points] The height of a binary tree with 20 leaves must be at least **five**. Exact answer please: no partial credit.

3. For each of the code fragments below, find the asymptotic time complexity in terms of n.

(a) [5 points]

```
for(float x = n; x > 1; x = x/2)
```

$\Theta(\log n)$

(b) [5 points]

```
for(int i = 0; i*i < n; i++)
```

$\Theta(\sqrt{n})$

(c) [10 points]

```
for(float x = n; x > 2.0; x = sqrt(x))
```

$\Theta(\log \log n)$

(d) [5 points]

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

$\Theta(\log n)$

(e) [5 points]

```
for(int i = n; i > 1; i=i/2)
```

$\Theta(\log n)$

(f) [10 points]

```
for(int i = n; i > 1; i--)
 for(int j = i; j > 1; j = j/2)
```

$\Theta(n \log n)$

(g) [10 points]

```
for(int i = n; i > 1; i--)
 for(int j = n; j > i; j = j/2)
```

$\Theta(n)$

(h) [10 points]

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

$\Theta(\log \log n)$

4. [20 points] Write pseudocode for the Floyd-Warshall algorithm. Let $W_{i,j}$ be the initial weight of the arc from i to j, ∞ if there is no arc. Let $V_{i,j}$ be the minimum weight of any path from i to j, and $back_{i,j}$ the backpointer.

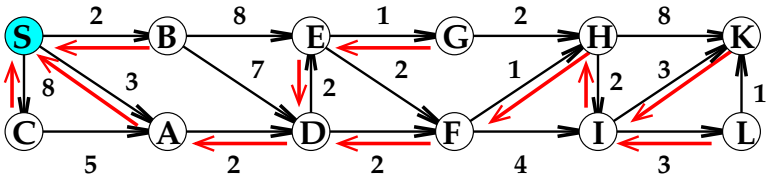
For all i
  For all j
    $V_{i,j} = W_{i,j}$
    $back_{i,j} = i$
For all j
  For all i
    For all k
      temp $= V_{i,j} + V_{j,k}$
      If(temp $< V_{i,k}$)
        $V_{i,k}$ = temp
        $back_{i,k} = back_{j,k}$

5. [20 points] Write Pseudocode for the Bellman-Ford algorithm. Be sure to use the shortcut.

   Arcs are ordered triples (s_j, t_j, w_j) for $1 \leq j \leq m$ Vertices are $0 \ldots n$ and the source is 0.

   For all i from 1 to n
     $V[i] = \infty$
   V[0] = 0
   bool finished = false
   while (not finished) loop
     finished = true;
     For all j from 1 to m
       temp $= V[s_j] + w_j$
       if (temp $< V[t_j]$)
         $V[t_j]$ = temp
         $back[t_j] = s_j$
         finished = false;
   end loop

6. [20 points] Walk through Dijkstra's algorithm for the following directed graph, where S is the start vertex.



	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

7. [20 points] Sort the following array using heapsort. Add extra rows if needed.

A	M	G	X	E	W
A	M	W	X	E	G
A	X	W	M	E	G
X	A	W	M	E	G
X	M	W	A	E	G
G	M	W	A	E	**X**
W	M	G	A	E	**X**
E	M	G	A	**W**	**X**
M	E	G	A	**W**	**X**
A	E	G	**M**	**W**	**X**
G	E	A	**M**	**W**	**X**
A	E	**G**	**M**	**W**	**X**
E	A	**G**	**M**	**W**	**X**
A	**E**	**G**	**M**	**W**	**X**
A	**E**	**G**	**M**	**W**	**X**

8. [10 points] Execute mergesort for the following array: QWIKRYZQHZDB

   Here are the steps.

   1. If there is only one item you are done. Otherwise:
   2. Divide the list into two approximately equal length sublists.
   3. Recursively sort each of those sublists.
   4. Merge the two sublists. The resulting list will be in order.

   Here is the computation for our example, obtained by unrolling the recursion.

   QWIKRYZQHZDB

   Q W I K R Y Z Q H Z D B

   Q W I K R Y Z Q H Z D B

   Q W I K R Y Q Z H Z D B

   Q W I K R Y Q Z H D Z B

   I Q W K R Y H Q Z B D Z

   I K,Q R W Y B D H Q Z Z

   B D H I K Q Q R W Y Z Z

9. [10 points] Execute polyphase mergesort for the following array.

   You must identify runs. Merging runs must be done in the correct order to maintain $O(n \log n)$ asymptotic time. By merging runs in the wrong order, the time complexity could go quadratic, that is, $O(n^2)$.

   QWIKRYZQHZDB

   Identify the runs.

Q W I K R Y Z Q H Z D B

Move the runs to two new files, each with half the runs.

Q W Q D

I K R Y Z H Z B

Execute *phases* until there is just one run. Each phase coonsists of merging the i^{th} phase of the first phase with the i^{th} run of the second phase, for all i. The merged runs are alternately placed in two new files.

Phase 1.

I K Q R W Y Z B D

H Q Z

The number of runs in the resulting two files is approximately halved. Separate the those two files into runs for the next phase.

I K Q R W Y Z B D

H Q Z

Phase 2. Same as phase 1, yielding two files.

H I K Q Q R W Y Z Z

B D

Last Phase. Eventually, after $O(\log n)$ phases, the two files will each have just one run. Merge these to form a single run, the sorted list.

B D H I K Q Q R W Y Z Z

10. [20 points]

There is a row of n coins, $c_1, c_2, \ldots c_n$. Each coin has a positive value, $v_1, \ldots v_n$. Write an algorithm which finds the maximum total value of any set of those coins which contains no three coins which are consecutive in the original row.

When you introduce variables while writing a program, it must be clear what these variables mean. Here is documentation I used.

n is the number of coins in the given set (row)
c_i is the i^{th} coin in the set
v_i is the value of c_i
A set of coins is *legal* if it contains no three consecutive coins of the original row.
$A[i]$ is the maximum value of any legal subset of the first i coins of the row.

My program starts by initializing the first three values of A. $A[0]$ defaults to zero, since there are no coins of index less than 1. The next step is to compute $A[i]$ for $i \geq 3$ dynamically. Finally, we write the answer, $A[n]$.

$A[0] = 0$
$A[1] = x_1$
$A[2] = x_1 + x_2$

For $i = 3$ to n
  $A[i] = max(A[i-1], x_i + A[i-2]), x_i + x_{i-1} + A[i-3])$
Return $A[n]$

11. [40 points] Write pseudocode for quicksort. Assume that no two items are equal.

Here is a high-level description of quicksort. Suppose $X[N]$ is the orginal array, and terms are unique.

1. If $N = 1$ we are done. Otherwise, go to step 2.
2. Choose some $p = X[i]$, the *pivot* value.
3. Partition the terms of X, other then $X[i]$, into two arrays A and B such that each term of A is less than p and each term of B is greater than p.
4. Recursivly sort both A and B.
5. The sorted list is the concatenation $A(p)B$.

Partition. A naive implementation of quicksort uses $O(n)$ workspace, that means, space beyond that needed for the input and output data. However partition can be done within the same space used for the input. The goal of a partition algorithm is to permute the initial array X to a concatenation of two subarrays, say A and B, and possibly other terms, such that sorting both A and B causes X to be sorted.

Hoare's Partition Algorithm. The original, and best-known, partition algorithm was designed by Tony Hoare, the inventor of quicksort. We describe that algorithm assuming no duplicate terms.

Here is a "rough" description of Hoare's partion algorithm.
1. Pick one term of X to be the pivot. Every other term of X will be either greater or less than the pivot.
2. Swap the pivot into $X[0]$.
3. Define running pointers lo on the left and hi on the right, which will move to the right and left, respectively, until $lo = hi$. See below for details.
4. Swap $X[0]$ with $X[lo]$. At this point, the array will be the concatenation $A(pivot)B$. The pivot is in its final position.
5. Recursively quicksort both A and B, and we're done.

Here is a detailed description of step 3.
  3.a Set $lo = 1$ and $hi = N$.
  3.b If $lo < hi$ and $A[lo + 1] < pivot$ increment lo.
  3.c If $lo < hi$ and $A[hi] > pivot$ decrement hi.
  3.d If $lo < hi$, $A[lo + 1] > pivot$, and $A[hi] < pivot$:
    3.d.i Swap $A[lo + 1]$ and $A[hi]$
    3.d.ii Increment lo
    3.d.iii Decrement hi

It is standard practice to use Hoare's algorithm even if terms are not unique. But, as I've mentioned in class, it is rather tricky to write that code. I do not recommend using Hoare's, or Lomuto's, algorithm in the case of duplicate terms. I will show you a new algorithm that is easier to understand and code.

Lomuto's Partition Algorithm A more recent partition algorithm is by Nico Lomuto. Programmers argue whether it's better than Hoare's. I think is equally good. Just as in Hoare's algorithm, Lomuto's is simple to undersand and code if terms are unique.

Choosing the Pivot. The partition of X into A and B should be balanced. That balance is etermined by the choice of pivot. A repeated lopsided partition could lead to quicksort taking quadratic time.

(i) The best choice of pivot is the median value of the terms of X, but that is usually not available.

(ii) We could dhoose a pivot at random. This gives, on the avarage, a balanced partition, but random numbers are computationally expensive.

(iii) I recommend picking the item in the middle of the array, $X[N/2]$, to be the pivot.

`https://www.geeksforgeeks.org/dsa/quick-sort-algorithm/`