Exam 2- Requires Respondus LockDown Browser

This is a preview of the draft version of the quiz

Started: Oct 28 at 7:08pm

Quiz Instructions

Notes Regarding this Examination

- **Canvas with Respondus LockDown Browser** You must use Respondus LockDown Browser to take this examination in Canvas. You have previously been given instructions on how to download, install, and test Respondus LockDown Browser with Canvas on your own computing device (laptop, iPad, etc.).
  - If you have failed to follow those instructions previously, it is too late to follow them now. You must instead ask your instructor for a paper copy of the examination and you will incur the standard 20% late penalty for having failed to complete an assignment by the due date. (If you have previously requested a paper copy of the examination, you will be provided with a paper copy of the examination at no penalty.)
  - If you will be taking a paper copy of this examination, you must close all electronic computing devices including the one on which you are reading these instructions and place them out of sight (for example, in your pocket or backpack) for the duration of the examination. This includes but is not limited to calculators, computers, and cellular phones.
- **Open Book(s)** You may consult any printed textbooks in your immediate possession during the course of this examination.
- **Open Notes** You may consult any printed notes in your immediate possession during the course of this examination.
- **Restricted Electronic Resources** You may consult your online electronic textbook (**https://learn.zybooks.com/zybook/OUCS2413HougenFall2018 (https://learn.zybooks.com/zybook/OUCS2413HougenFall2018)**), the course website (**http://www.cs.ou.edu/~hougen/classes/Fall-2018/DataStructures/ (http://www.cs.ou.edu/~hougen/classes/Fall-2018/DataStructures/)**), and the files section of Canvas for this course (**https://canvas.ou.edu/courses/88748/files/folder/Lecture%20Slides**) using the links provided here. *You may not use other electronic resources during this exam, including but not limited to (1) following links from the approved sites to other sites in these or other domains and (2) any files stored locally on the device on which you are taking this exam.*
- **No Additional Electronic Devices Permitted** Other than the computing device on which you are completing this exam, you may not use any electronic devices during the course of this examination, including but not limited to calculators, computers, and cellular phones. All additional electronic devices in the student's possession must be turned off and placed out of sight (for example, in the student's own pocket or backpack) for the duration of the examination.
- **Violations** Copying another's work, or possession of unauthorized electronic computing or communication devices in the testing area, is cheating and grounds for penalties in accordance with school policies.

Question 1 — 1.5 pts

An ordinary, singularly-linked list can be traversed forward or backward.

- True
- False

Question 2 — 1.5 pts

An ordinary, singularly-linked list allows individual links to be placed in any available place in the heap (free store).

- True
- False

Question 3 — 1.5 pts

An ordinary, singularly-linked list requires a contiguous block of memory to hold all the links.

- True
- False

Question 4 1.5 pts

An ordinary, singularly-linked list has an insertion time of O(1) for inserting an item into the list.

- True
- False

Question 5 1.5 pts

An ordinary, singularly-linked list has a deletion time of O(1) for finding and deleting an item based on keys.

- True
- False

Question 6 1.5 pts

An ordinary, singularly-linked list has a replacement time of O(1) for finding and replacing an item based on keys.

- True
- False

Question 7 1.5 pts

An ordinary, singularly-linked list has a retrieval time of O(1) for finding and returning at item based on keys.

- True
- False

Question 8 1.5 pts

An ordinary, singularly-linked list an easily and efficiently be used for a stack.

- True
- False

Question 9 1.5 pts

An ordinary, singularly-linked list can easily and efficiently be used for a queue.

True

False

Question 10

1.5 pts

An ordinary, singularly-linked list can be created (empty) in O(1) time.

True

False

Question 11

1.5 pts

An ordinary, *doubly*-linked list can be traversed forward or backward.

True

False

Question 12

1.5 pts

An ordinary, *doubly*-linked list allows individual links to be placed in any available place in the heap (free store).

True

False

Question 13

1.5 pts

An ordinary, *doubly*-linked list requires a contiguous block of memory to hold all the links.

True

False

Question 14

1.5 pts

An ordinary, *doubly*-linked list has an insertion time of O(1) for inserting an item into the list.

True

False

Question 15 1.5 pts

An ordinary, *doubly*-linked list has a deletion time of O(1) for finding and deleting an item based on keys.

- True
- False

Question 16 1.5 pts

An ordinary, *doubly*-linked list has a replacement time of O(1) for finding and replacing an item based on keys.

- True
- False

Question 17 1.5 pts

An ordinary, *doubly*-linked list has a retrieval time of O(1) for finding and returning at item based on keys.

- True
- False

Question 18 1.5 pts

An ordinary, *doubly*-linked list can easily and efficiently be used for a stack.

- True
- False

Question 19 1.5 pts

An ordinary, *doubly*-linked list can easily and efficiently be used for a queue.

- True
- False

Question 20 1.5 pts

An ordinary, *doubly*-linked list can be created (empty) in O(1) time.

- True

False

Question 21

1.5 pts

An *open-indexing hash table* uses separate chaining as an collision-resolution strategy.

- True
- False

Question 22

1.5 pts

An *open-indexing hash table* allows individual *buckets* to be placed in any available place in the heap (free store).

- True
- False

Question 23

1.5 pts

An *open-indexing hash table* requires a contiguous block of memory to hold all the *buckets*.

- True
- False

Question 24

1.5 pts

An *open-indexing hash table* has an insertion time of O(1) for inserting an item into the table.

- True
- False

Question 25

1.5 pts

An *open-indexing hash table* has a deletion time of O(1) for finding and deleting an item based on keys.

- True
- False

Question 26

1.5 pts

An *open-indexing hash table* has a replacement time of O(1) for finding and replacing an item based on keys.

- True
- False

Question 27

1.5 pts

An *open-indexing hash table* has a retrieval time of O(1) for finding and returning at item based on keys.

- True
- False

Question 28

1.5 pts

An *open-indexing hash table* can easily and efficiently be used for a stack.

- True
- False

Question 29

1.5 pts

An *open-indexing hash table* can easily and efficiently be used for a queue.

- True
- False

Question 30

1.5 pts

An *open-indexing hash table* can be created (empty) in O(1) time.

- True
- False

Question 31

1.5 pts

It usually takes fewer steps to insert an item in a hash table than in a linked list, as the number of items already contained in the data structure becomes large.

- True
- False

Question 32

1.5 pts

It usually takes fewer steps to *find* a *specific* item in a hash table than in a linked list, as the number of items already contained in the data structure becomes large.

- True
- False

Question 33

1.5 pts

It usually takes fewer steps to insert an item in a hash table than in an *array*, as the number of items already contained in the data structure becomes large.

- True
- False

Question 34

1.5 pts

It usually takes fewer steps to *find* a *specific* item in a hash table than in an *array*, as the number of items already contained in the data structure becomes large.

- True
- False

Question 35

10 pts

Hashing

Given the following items to insert into a hash table of size 10, fill in the blanks/buckets in the table to show the hash table after all items have been inserted. **If a blank/bucket should have no item in it after all items have been inserted into the table, put the word "none" in that location in the table.**

- The items are to be inserted starting from the top of the list and working down.
- The primary hash function is *key modulus table_size*.
- The collision resolution strategy is double hashing.
- The secondary hash function is *key div table_size*, where *div* is integer division (that is, division discarding the remainder).

Items to insert

Item	Hash Code
A	54
B	43
C	28
D	60
E	33
F	79
G	81
H	41

I	88
J	67

Hash table

Bucket Number	Item
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Question 36

20 pts

Linear Hashing

Given the following items to insert into a hash table that uses linear hashing, fill in the blanks/buckets in the table to show the hash table after all items have been inserted. **To show how large the hash table has grown, only put bucket numbers in the blanks for the rows used in the table. For the remaining blanks, put the word "none" in place of the bucket number. Similarly, if a bucket or separate chain link should have no item in it after all items have been inserted into the table, put the word "none" in that location in the table.**

- The items are to be inserted starting from the top of the list and working down.
- The collision resolution strategy is separate chaining.
- Be sure to treat and write the hash codes and bucket numbers as binary numbers as shown in the 2018 slides.

Items to insert

Item	Hash Code
A	0010
B	1001
C	0000
D	1110
E	1000
F	0011
G	1100
H	1100
I	1101

J	1111

Hash table (with additional columns to represent where links for separate chaining would connect)

Bucket Number	Item	Link1	Link2

Question 37

30 pts

Radix Sort

```
// Radixsort takes:
// A: the array to sort
// r: the radix (base) for the keys to be sorted
// d: the number of digits (of the given radix) in each key
Algorithm Radixsort (A, r, d)
  create Q[ r ] // Q is an array of r queues, all initially empty
  for k from 0 to d-1
    for i from 0 to A.size
      Q[(A[ i ].key/(r to the power k)) modulus r].enqueue(A[ i ])
    end for i
    i ← 0
    for j from 0 to r do
      while Q[ j ] is not empty
        A[ i ] ← Q[ j ].dequeue()
        i ← i + 1
      end while
    end for j
  end for k
```

Given *r* is 10 and *d* is 2, show the steps followed by the Radix Sort algorithm given above in pseudocode when sorting the following array. Fill in the values in the figures for Q and A for each value of k. **All empty locations should be marked "none" in these figures.**

A (initially):

index	0	1	2	3	4	5	6	7	8	9
value	28	52	22	75	90	60	84	55	61	4

Q (when k = 0):

index	0 0 1	2 1 3	4 2 5	6 3 7	8 4 9	5	6
value at head							
value at next							

A (when k = 0):

index	0 0 1	2 1 3	4 2 5	6 3 7	8 4 9	5	6
value							

Q (when k = 1):

index	0 0 1	2 1 3	4 2 5	6 3 7	8 4 9	5	6
value at head							
value at next							

A (when k = 1):

index	0 0 1	2 1 3	4 2 5	6 3 7	8 4 9	5	6
value							

Question 38

6 pts

Match each term to a statement that is true for it. (Note that there are more statements than terms, so some statements will go unmatched.)

Linear probing

Quadratic probing

Double hashing

A perfect hash function

A minimal perfect hash function

Separate chaining

resolves collisions using the quadratic formula.

performs in O(n log n) time.

resolves collisions by searching linearly through the hash table for an open bucket.

is another name for folding.

never leaves empty buckets in the hash table.

never results in collisions.

resolves collisions through the use of a lookup table.

resolves collisions through binary search.

resolves collisions through the use of external linked lists.

performs in quadratic time, that is O(n^2) time.

resolves collisions by using different offsets for different keys.

resolves collisions by using increasingly large offsets.

Question 39

1.5 pts

Radix sort is stable.

- True
- False

Question 40

1.5 pts

Radix sort is an inplace algorithm.

- True
- False

Question 41

1.5 pts

Radix sort manages to surpass O(*n* log *n*) performance on unique keys by not using key comparisons.

- True
- False

Question 42

3 pts

Radix sort uses which of the following.

- Hash tables
- Binary search
- Stacks
- Queues
- Doubly-linked lists

Question 43

1.5 pts

Stacks are first-in/first-out (FIFO).

- True
- False

Question 44 1.5 pts

Queues are first-in/last-out (FILO).

- True
- False

Question 45 1.5 pts

Stacks can be built using linked lists.

- True
- False

Question 46 1.5 pts

Queues can be built using linked lists.

- True
- False

Question 47 1.5 pts

Stacks can be built using arrays.

- True
- False

Question 48 1.5 pts

Queues can be built using arrays.

- True
- False

Question 49 1.5 pts

Stacks allow for retrieval based on index.

True

False

Question 50 **1.5 pts**

Queues allow for retrieval based on index.

True

False

Question 51 **1.5 pts**

Primary clustering is a result of using prime numbers as hash table sizes.

True

False

Question 52 **1.5 pts**

One advantage of using modulus arithmetic as a hash function is that it is fast to compute.

True

False

Question 53 **1.5 pts**

Bucket search is the method used to find items in hash tables.

True

False

Question 54 **1.5 pts**

Circular queues can be built using arrays.

True

False

Question 55

1.5 pts

Circular queues can be built using linked lists.

- True
- False

Question 56

3 pts

Adding *n* items to a linked list while keeping it sorted takes how much time?

- $\Theta\left(n^2\right)$
- $\Theta\left(2^n\right)$
- $\Theta\left(n\right)$
- $\Theta\left(1\right)$
- $\Theta\left(n\ \log\ n\right)$

Question 57

3 pts

Adding *n* items to an array while keeping it sorted takes how much time?

- $\Theta\left(n\right)$
- $\Theta\left(n\ \log\ n\right)$
- $\Theta\left(2^n\right)$
- $\Theta\left(n^2\right)$
- $\Theta\left(1\right)$

Quiz saved at 7:09pm Submit Quiz