CS 2C Final Exam

Due Jun 23 at 11:59pm **Points** 39.9 **Questions** 30
Available Jun 23 at 12am - Jun 23 at 11:59pm 23 hours and 59 minutes
Time Limit 120 Minutes

Instructions

Finish this test before the due date. Once you begin, you will have 120 minutes minutes to complete. If you do not submit before that time your incomplete exam will be automatically submitted as is.

You can look at lectures or texts and even use your compiler, but you may not consult any other individuals for help.

There are 30 questions, each question is worth about 1.33 points, for an exam total of 40 points.

Multiple choice questions with square check-boxes may have more than one correct answer. Multiple choice questions with round radio-buttons have only one correct answer.

Any code fragments you are asked to analyze are assumed to be contained in a program that has all the necessary variables defined and/or assigned.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	61 minutes	28.82 out of 39.9

Score for this quiz: **28.82** out of 39.9
Submitted Jun 23 at 5:46pm
This attempt took 61 minutes.

Question 1

0.44 / 1.33 pts

Match the item on the left with the best description on the right.

Correct!

is used to find the shortest paths in a graph. The dijkstra algorithm ou Answered

is used to find the minimum spanning tree in a graph. A Maximum Flow Alg

Correct Answer The kruskal algorithm

ou Answered

prefers finding greater costs rather than smaller costs for its final edges. The kruskal algorithm

Correct Answer A Maximum Flow Algorithm

Question 2

1.33 / 1.33 pts

You have learned a great deal about C++ and data structures in this class.

Correct!

[✓ True]

False

Question 3

1.33 / 1.33 pts

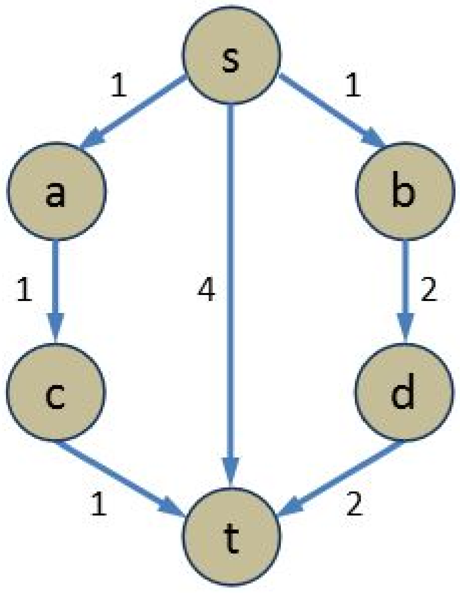
Shell sort is based on another sort in the sense that, in its final iterative step it actually ***the same as*** this mystery sort. What sort is Shell sort based on?

- ○ Bubble sort
- Correct! [✓ Insertion sort]
- ○ Merge sort
- ○ Quick sort

Question 4

1.33 / 1.33 pts

Select the ***shortest path*** (based on standard graph theory definitions) from **s** to **t** in the following directed graph, whose edge costs are clearly labeled:



- Correct! [✓ s → a → c → t]
- ○ s → c → t
- ○ s → t

○ s → b → d → t

Tie between

s → a → c

and

s → b → d → t

○

Tie between

s → a → c

and

○ d → t

Feedback

Shortest paths always count costs. There are very few paths to try, and this can be done without any algorithm, based solely on the definition of shortest path and the two vertices, **s** and **t**, selected for the ***start*** and ***end***.

Question 5

0.44 / 1.33 pts

Match the item on the left with the best description on the right.

ou Answered

typically use two containers to implement. Quadratic Probing Ha

Correct Answer Separate Chaining Hash Tables

Correct!

suffer from clustering. Linear Probing Hash T

ou Answered

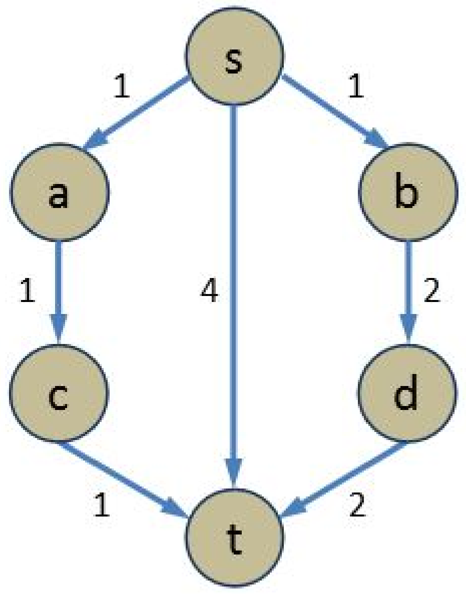
require a load factor of 0.5 Separate Chaining Ha

Correct Answer Quadratic Probing Hash Tables

Question 6

1.33 / 1.33 pts

Select the ***maximum flow*** (based on standard graph theory definitions) from **s** to **t** in the following directed graph, whose edge costs are clearly labeled:



◯ 2 units

◯ 3 units

○ 4 units

○ 5 units

Correct! [✓ 6 units]

○ 7 units

Feedback

Any flow will always use the middle path, providing 4 units. There are only two other paths, which do not interact, so each one can be considered separately. The left path clearly has flow 1 (as that's the only value each pipe on that side has). The path on the right has a pipe that can only hold, one, so the 2s in that side don't help anything. So 1 unit for each of the two sides and 4 for the middle = 6 units.

Question 7

1.33 / 1.33 pts

Which of the following ADTs are provided in stl as a template class. That is, which of the ADTs can be used right out-of-the-box, without adding much or any code.

*[Note: I am asking about **stl**, not some 3rd party or other library. It has to be in **stl** in some form that is essentially a general implementation of the requested ADT. It's okay if the **stl** version does more, as long as it can be used to handle the ADT without lots of extra code.]*

Example. **stl** has a **Deque interface**. Even though this does *more* than the ADT **queue**, it is good enough: We would check this box.

- [ ] General Tree
- [ ] Binary Tree
- [x] Linked List — Correct!
- [x] Priority Queue — Correct!
- [ ] Graph
- [x] Stack — Correct!

Feedback

No trees or graphs in stl that can be used by the client as template ADTs (even if there my be some underlying trees used to implement sets -- such trees are not useable by us as general trees.)

The rest have useful analogs.

Question 8

0 / 1.33 pts

The fastest general purpose sorting algorithm in widespread use has a big-O search time of:

orrect Answer: N^2

ou Answered: [✓ $N\log N$]

$(\log N)^2$

N

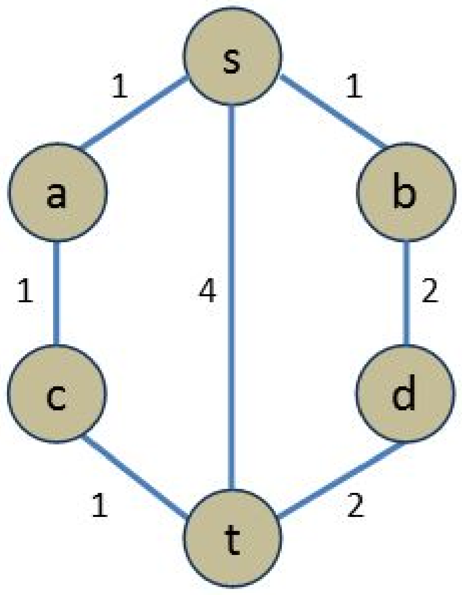
Feedback

The fastest general purpose sort is ***quicksort***, and this has big-O N^2.

Question 9

0 / 1.33 pts

Select a ***minimum spanning tree*** (based on standard graph theory definitions) of the following non-directed graph, from the choices listed:



[Note: While there may be more than one MST for this graph, only one MSTs appears in the following list.]

- Edges **[s, a]. [a, c], [c, t], [s, t], [s, b]** and **[b, d]**

ou Answered

- [✓ Edges **[s, a]. [a, c], [c, t], [s, b], [b, d]** and **[d, t]**]

- Edges **[s, t], [s, b], [b, d]** and **[d, t]** orrect Answer

○ Edges **[s, a]. [a, c], [c, t], [s, b]** and **[b, d]**

○ Edges **[s, a]. [a, c], [c, t], [b, d]** and **[d, t]**

Feedback

The minimum spanning tree needs to connect all nodes, yet do so using the minimum overall edge cost. One of the solutions is is in our list:

Edges **[s, a]. [a, c], [c, t], [s, b]** and **[b, d]**

Another, not listed, is just as good is:

Edges **[s, a]. [a, c], [c, t], [s, b]** and **[d, t]**

The cost of either of these is 6.

Question 10

1.33 / 1.33 pts

A class, **`AllowedPerson`**, has, among other fields, the following four which are defined here:

- **`name`** - a String which is the persons's name.
- **`age`** - an int which is the person's age in years
- **`ssn`** - a String which is the person's social security number (unique legal ID)
- **`weight`** - an int which is the person's weight in pounds

We will write some applications that use the stl **`Set`** data structure.

The **< operator** will be defined differently, depending on the application. Each of the above fields represents one possible choice, and when we consider an application, we will pick one of those choices. ***Assume we are using operator < to construct exact equality in each case -- not ranges***: Two people will be equal based on name if they have the ***same*** **name**). Two people will be equal based on **age** if they have the ***same***

age, (numeric **= =**), etc. Assume that we have defined < so that when < is used in a logical expression of some sort, it can determine an exact match.

Here are the applications that we will write:

1. A ***voting application*** which must record each person's vote along with some way to track who voted for whom (non-anonymous) and requires that ***no person be counted twice***.
2. A ***martial arts application*** which requires ***exactly two people*** in the each weight category. Weight categories are of the form
   1. < 100 pounds
   2. >= 100 but < 120.
   3. >= 120 but < 135
   4. etc.
3. ***Winners in a swimming competition*** which ***only allows one winner*** in each age group. Age groups are ***single years*** (14 years is one group. 15 years is another, etc.) *[**Note**, that this is the data structure to hold **only winners**, no all contestants.]*

Check the true statements. I am asking whether the **Set** selected in each option (using the exact member-equality) ***essentially solves***, without any additional machinery or logic, the problem of storing **`AllowedPerson`** objects into a **`HashSet`** providing the same restrictions as the application describes.

Example:

- A **Set** based on **`name`** equality ***is good*** for an application which ***stores candidates for naming a newborn baby***. (Even though we would be storing the **`ssn`** and **`age`**, along with the **`name`** -- which might seem excessive -- the question is about whether the **name** field works for baby name options and the answer is ***yes***. Exactly one name per **`AllowedPerson should`** be allowed into the **`HashSet`** candidate pool. (WE WOULD CHECK THIS BOX)
- A **Set** based on name equality ***would not be good*** for a voting application, since two people having the same name should both be allowed to vote; names are not unique. WE WOULD NOT CHECK THIS BOX.

☐ A `Set` based on `name` equality is good for the ***winners in a swimming competition***.

Correct!

☑ A `Set` based on `age` equality is good for ***winners in a swimming competition***.

Correct!

☑ A `Set` based on `ssn` is good for the ***voting application***.

☐ A `Set` based on `weight` is good for the ***martial arts application***.

Feedback

- **age** works for the swimming winner application. We want only one winner in each age group, and they are grouped by individual years, just like the ages.
- **ssn** works for voting. We want legal unique equality, and this is what **ssn** means.
- **weight** ***does not work*** for the martial arts application. Trying to use weight for the martial arts will not be enough for at least two different reasons, either one of which is enough to disqualify it:
  1. We can have two, not one person in each category. **Sets** are not designed to enforce two of each object for a given equality, and the problem asks us to assume that we are using exact equality (not ranges) and we also want the choice to essentially solve the problem without additional logic.
  2. Two people of different weights can be in the same category, so we need something that is not based on exact equality.
- **name** ***does not work*** for the voting (or any of the above) applications. names are not unique.

Question 11

1.33 / 1.33 pts

Check the true statements about any class you might design to emulate the **stl map** template.

Interpret ***"first position*** of a pair" to mean the **key**, used in as the index of a **myMap[key] = value** assignment, and "***second position*** of a pair" to mean the **value**, or RHS of that assignment.

Correct!

[✓] It would assume that a **< operator** is defined for for its **key** *type- parameter* such that each **key** is assumed to be unique relative to that operator.

We should allow that the **key** appear in either the ***first*** or in the ***second*** position of each ***pair*** stored in the **map**.

If our **map** contains a pair with the **key** 9999, we should assure that any attempt to overwrite this pair will fail.

It would assume that a **== operator** is defined for for its **key** type parameter such that each **key** is assumed to be unique relative to that operator.

It would assume that a **< operator** ***or*** a **== operator** is defined for for its **value** type parameter such that each **value** is assumed to be unique relative to that operator.

It should be designed so that the client instantiates/specializes our **map** using ***one type-parameter***.

Feedback

The **key** for each pair in a **map** is unique, and should appears, ***always*** as the ***first position*** of the pair. We can successfully overwrite pairs already in the **map**. Our map must require two type-parameter in order to emulate the stl map, and it should only require a **< operator**, but always that one operator.

Question 12

0 / 1.33 pts

Quicksort has better performance by using the ***smallest recursion limit*** (the size of the sub-array at which point we employ a different sort, like insertion sort).

ou Answered

[✓ True]

orrect Answer

False

Feedback

The best recursion limit for ***quicksort*** is always larger than 2, as stated in the modules and also as you all discovered in your assignments, so this statement cannot be true.

Question 13

1.33 / 1.33 pts

In DIJKSTRA (match the phrases):

Correct!

might end up taking more time because inserting into this data structure is not a constant-time operation.

Using a set for the pa

Correct!

might end up taking more time due to multiple copies of the same vertex being pushed and popped unnecessarily.

Using a queue for the

Question 14

1.33 / 1.33 pts

Quadratic probing does not completely solve the ***collision*** problem of ***linear probing***, but it does reduce ***clustering***.

Correct!

[✓ True]

False

Feedback

It's all true.

Question 15

1.33 / 1.33 pts

A pure Insertion sort is faster than a pure Quicksort for small size input arrays.

Correct!

[✓ True]

False

Feedback

A "pure Quicksort" implies no recursion limits, but even you were thinking about a Quicksort with a recursion limit, it would still be slower for small arrays than calling Insertion sort directly from the client.

Question 16

1.33 / 1.33 pts

Both bubble sort and insertion sort have quadratic time complexity.

Correct!

[✓ True]

False

Question 17

0 / 1.33 pts

A single stl map can have the same value appear in more than one of its key-value pairs.

orrect Answer

True

ou Answered

[✓ False]

Feedback

A key can appear in only one pair of a map, but a value can appear in many.

Question 18

1.33 / 1.33 pts

Merge Sort and ***Heap Sort*** are both big-O ***NlogN.***

Correct!

[✓ True]

False

Feedback

It's all true.

Question 19

1.33 / 1.33 pts

Match the item on the left with the best description on the right.

Correct!

do not require an ordering operation for the underlying data type. — Hash Tables

Correct!

do require an ordering operation for the — Binary Heaps

underlying data type.

Question 20

0 / 1.33 pts

Internally, a bin heap is typically implemented as a **BST** that has the ***additional*** ordering property that ***the minimum value is at the root*** (or the ***maximum value is at the root***, if we chose to make the bin heap to be a max heap).

ou Answered

[✓ True]

orrect Answer

False

Feedback

A ***bin heap*** does not have the strong ordering property that a **BST** has. The only kind of **BST** that would also be a potential ***bin heap*** at some point in time is one in which the **root** had no left sub-tree (for a ***min heap***, say), and this is not going to be true of most **BST**s. Insisting that some implementation of a bin heap be a **BST** would impose both unnecessary ordering on the rest of the sub-tree, and also be highly unbalanced, something that a bin heap cannot be (since it ***must be complete***).

Question 21

1.33 / 1.33 pts

AVL Trees ...

(Check all that apply.)

Correct! ✓ ... do not ***drastically alter*** their structure when the tree is searched.

☐ ... might result in left- and right-subtrees having heights that differ by > 2.

☐ ... will always leave the result of a successful (i.e., "found") search or insert at the master root node.

Correct! ✓ ... are special kinds of BSTs.

Correct! ✓ ... will, for any given single search, deliver **O(log*N*)** search performance.

Feedback

Answers self-explanatory, except possibly for the drastic alteration question. Searching for an element in an AVL tree does not alter the tree. Splay trees are the only tree that we studied which alter their structure during a search.

Question 22

1.33 / 1.33 pts

Match the item on the left with the best description on the right.

Correct!	**gives an upper bound on growth.**	A Big-Oh time estimat
Correct!	**gives simultaneous upper and lower bounds on**	A Theta estimate

growth.

Correct!

gives a lower bound on growth. An Omega time estim

Question 23

0.67 / 1.33 pts

Consider properties of a ***BST*** vs. a ***General Tree***. Match the properties described with appropriate tree type.

ou Answered

BST Only Can handle a pre-ord

Correct Answer Requires an associated ordering property which inherently compares (using a less than-type relationship) any two elements in the tree.

ou Answered

Both Requires an associate

Correct Answer Can handle a pre-order traversal.

Correct!

General Tree Only For a fixed height, say

Correct!

Neither Is an efficient way to s

Feedback

Correct answers are explanatory.

Question 24

1.33 / 1.33 pts

All **AVL** Trees have a **O(log*N*)** search time.

Correct!

[✓ True]

False

Feedback

AVLs are always balanced, so they have **O(log*N*)** search times.

Question 25

1.33 / 1.33 pts

Match the item on the left with the best description on the right.

Correct!	**has a tight Big-Oh search time of O(logN).**	An AVL tree
Correct!	**will find the most recently accessed element in constant time.**	A Splay tree

Question 26

0 / 1.33 pts

The solution to the subset-sum problem for the master list:

{15, 5, 3, 1, 5,}

and the target

14

is:

- 5
- 8
- ou Answered [✓ 12]
- 13
- orrect Answer 14
- 15

Feedback

Sublist **{5, 5, 3, 1}** has the sum = **14**, which is a perfect hit.

Question 27

0.67 / 1.33 pts

Splay Trees ...

(Check all that apply.)

Correct!

- [✓] ... are reasonably efficient if we access the same element of the tree several times in succession, without doing any intervening insertions or deletions.

orrect Answer

- [ ] ... might result in left- and right-subtrees having heights that differ by > 2.

Correct!

- [✓] ... will always leave the result of a successful (i.e., "found") search or insert at the master root node.

Correct!

- [✓] ... are special kinds of BSTs.

ou Answered

- [✓] ... for any given single search will deliver **O(log*N*)** search performance.

Feedback

Answers self-explanatory, except possibly for the one concerning successive searches.

Splay trees are perfect for accessing the same element multiple times in succession: that's the property they were designed to optimize.

Question 28

1.33 / 1.33 pts

You are given an array of N elements in sorted order. You can create a balanced Binary Search Tree from this array in

(Pick only the tightest bound)

Correct!

[✓ O(N) time]

O(N log N) time

O(N^2) time

O(1) time

To create a balanced BST from the array, we'd simply pick the elements in a particular order (e.g. mid, (0+mid)/2, etc.) to guarantee that we grow the BST systematically. Array element access is O(1) and constructing the BST in this way is O(n) if you know exactly which array index holds the element you want to insert into the current node.

Note that if the array is sorted, then it's already had O(n log n) work done on it (from the POV of the final goal).

Question 29

1.33 / 1.33 pts

Logarithmic running times grow more slowly (w.r.t. the data size, *N*) than quadratic running times.

Correct!

[✓ True]

False

Question 30

1.33 / 1.33 pts

Match the phrases.

Correct!	**a binary search tree.**	A data structure that r
Correct!	**a general tree.**	A heirarchical organiz

Quiz Score: **28.82** out of 39.9