CS 2C Midterm Exam

Due May 12 at 11:59pm **Points** 20 **Questions** 20
Available May 12 at 12am - Jun 23 at 11:59pm about 1 month **Time Limit** 60 Minutes
Allowed Attempts 2

Instructions

Finish this test before the due date and time. Once you begin, you will have 60 minutes to complete it. If you do not submit before that time, your incomplete exam will be automatically submitted as is. You are not authorized to take this test in multiple sessions, so do not start it unless you have protected time in which to take the test.

You can look at lectures or texts and even use your compiler, but you may not consult any other individuals or non-course help/ask sites for help. Reference sites are fine.

Each question is worth 1 point.

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.

None of these questions is a trick. They pose straightforward questions about programming and language concepts and rules taught in this course.

Best,

&

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	46 minutes	16.17 out of 20

Score for this attempt: **16.17** out of 20
Submitted May 12 at 1:19pm

This attempt took 46 minutes.

Question 1

0 / 1 pts

If an algorithm has an **O(log*N*)** search, and it takes 10 units of time to process ***x***-items, how long will it take to process 16***x*** items?

- 20 units of time

ou Answered

- 40 units of time
- 360 units of time

orrect Answer

- 14 units of time
- 60 units of time

Feedback

For 16x items, it ought to take log(16x) time.

Assume log base = 2. Then log(16x) = log(16) + log(x) by property of logarithms.

$\log_2 16 = 4$ and $\log_2 x = 10$ as given, so it will take 14 units of time.

Question 2

1 / 1 pts

Logarithmic running times grow more slowly (with respect to the data size, *N*) than quadratic running times.

Correct!

- (selected) True
- False

Question 3

1 / 1 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 growth. — A Theta estimate

Correct!

gives a lower bound on growth. — An Omega time estim

Question 4

0 / 1 pts

Consider the following correct code, that inserts **theTune** into an **theList**, in smallest-to-largest order (declarations omitted for clarity):

```
for (iter = theList.begin(); iter != theEnd; ++iter )
   if ( theTune < *iter)
      break;

theList.insert(iter, theTune);
```

Assume the other methods associated with this effort maintain the order that the above loop establishes.

Based on this code, check the true statements (may be more than one).

Correct!

- [x] Changing the **insert()** so that it maintained a largest-to-smallest order ***would NOT NECESSARILY*** require us to modify our **remove()** algorithm: the typical design for the one that we had before, will continue to work for the new, opposite, order.

orrect Answer

- [ ] This requires the underlying class that **theTune** and the **theList *elements*** all belong to have an overloaded operator<().

- [ ] If we were to change this code so that it maintained a ***greatest-to-smallest*** order, the underlying class would have to overload either or both of operator>() or operator>=().

ou Answered

- [x] The code does not allow an element to be added to the list more than once.

Feedback

Changing insert so that it creates an opposite ordering only requires that we change

```
      if ( theTune < *iter)
```

to

```
      if ( *iter < theTune )
```

No new operators need be defined. Also, the remove() does not have to be changed.

Question 5

1 / 1 pts

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

{17, 5, 3, 3, 7}

and the target

25

is:

- 24
- 27
- Correct! 25
- 23
- 26
- 22

Feedback

Sublist **{17, 5, 3}** has the sum = **17 + 5 + 3** = 25 which is our target.

Question 6

1 / 1 pts

vectors are:

(Check all that apply.)

Correct!

- [x] ... admit reasonably fast random access for the `k` th element.
- [ ] ... faster than arrays.

Correct!

- [x] ... uses standard array notation (like `myVec[k]`) to access the `k` th element.

Correct!

- [x] ... more convenient for automatically growing (from the client's perspective) the number of elements that the data structure can and does hold.
- [ ] ... better than arrays

Feedback

We've seen that **vectors** are not necessarily faster than arrays in all situations, so they cannot be unconditionally better, covering two of the choices.

We can use standard array notation with **vectors**. The implementation is, internally, a simple array access, so it is reasonably fast, covering two more choices.

The one thing that we first learn about **vectors** is that they grow as needed.

Question 7

0.5 / 1 pts

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

ou Answered

BST Only — Have both insertion a

Correct Answer — Can handle a well-defined and obvious in-order traversal

ou Answered

Both — Can handle a well-def

Correct Answer — Have both insertion and removal methods.

Correct!

General Tree Only — For a fixed height, say

Correct!

Neither — Have a worst-case tin

Feedback

Both types of tree could have a bad structure (like a list) requiring O(*N*) search. Other answers should be clear.

Question 8

1 / 1 pts

This question is based on our implementation of **vector (FHvector)**, which is typical. Recall that our implementation, internally, calls a method **reserve()** whenever extra storage space is needed. Some details of this operation are universal for theoretical reasons. We may not have covered the theoretical reasons yet, but we *have* studied the implementation of **reserve()**. Consider the following scenario.

The Current State of One Particular FHvector

The ***capacity*** (**mCapacity**) of a particular **FHvector** is **five** (**5**) at some point in time. At that same time, there happens to be **four** (**4**) elements of client data in the list.

The Next Operation Requested of this FHvector

From that state, a client requests to **push_back() 25** data items into the list.

The Internal Reaction of the FHvector

This will certainly require an increase in the vector's **capacity**. The question is about *what actually happens internally* in the time interval during which the totality of of these **25 push_back()**s are executed.

- There will be **two** calls to **reserve()**.
- There will be ***no*** calls to **reserve()**.

Correct!

- There will be **three** calls to **reserve()**.
- There will be ***one*** call to **reserve()**.

Feedback

reserve() (roughly) doubles the allocation each time, so after we reach the limit of 5, it will double to 10, but that won't be enough for the (soon to be total 4 + 25 =) 29 elements the list will have to hold. Well need to double the list three times 5 -> 10, 10 -> 20, 20 -> 40. (or, if we double and add one, 5 -> 11, 11 -> 23, 23 -> 47)

Question 9

1 / 1 pts

Match the phrases.

Correct!

a binary search tree. A data structure that r

Correct!

a general tree. A heirarchical organiz:

Question 10

1 / 1 pts

A ***4×4 sparse matrix*** of **ints** is defined with **default value = 2**. Through a series of mutators, the matrix has the logical (theoretical) value:

```
 0   2  -1  -1
-1   2  -1   2
 0   0   2   0
 0   0   2  -1
```

How many data-holding sparse matrix Nodes (**MatNode**s) will be created?

(We ***do not count*** internal bookkeeping *head* or *tail* nodes of linked lists. We only consider nodes that hold numeric **int** data.)

- 5
- 13
- Correct! 11
- 16

Feedback

The value that is *not* stored is 2, since that is the ***default***. There are **5 (five) 2**s in the logical matrix, and 11 values which are non-default (11 values are not 2). Therefore ***11 data nodes are required***.

Question 11

1 / 1 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 12

1 / 1 pts

All **BST**s have a **O(log*N*)** search time.

Correct!

- (selected) False
- True

Feedback

A BST could be heavily unbalanced, producing a worst case **O(*N*)** search time.

Question 13

0 / 1 pts

What is the Big-Oh time complexity for this main algorithm (assume sqrt() returns the closest int to the square root of its argument):

```
// in main()
for (k = 0; k < M; k++)
{
      for (j = 0; j < sqrt(M); j++)
         myMethod(M);
}

// at global scope
void myMethod(int M)
{
   int k, x;
   for (k = 0; k < sqrt(M); k++)
      x = 1;  // represents any constant-time statement.
}
```

orrect Answer

- M^2
- $\log^2 M$ (which means log*M* squared)

ou Answered

- *M*
- *M*log*M*

Feedback

There are three loops in the code, all nested, two have length sqrt(*M*) and one has length *M*, so total is M^2

Question 14

1 / 1 pts

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

- False

Correct!

- True

Feedback

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

Question 15

1 / 1 pts

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

Correct!

typically expect the client to supply a type parameter in angled brackets, .

Template classes

Correct!

are invoked without a type parameter in angled brackets, .

Template methods

Question 16

0.67 / 1 pts

Template classes:

(Check all that apply.)

Correct!

- [x] ... might require that the class which is used to specialize the template by the client have many operator overloads (like <, >=, etc.) defined.

Correct!

- [x] ... allow the client to decide some or all of the template class's internal member data types.

orrect Answer

- [ ] ... will issue a fatal compiler error if a *type parameter* is missing.

- [ ] ... allow ***only primitive*** *type parameters*.

Feedback

Type parameters must be supplied by the client, and if the template definition which client invokes make use of several operators, like < or >=, for the type parameters, all of those operators have to be defined or overloaded for that type.

Question 17

1 / 1 pts

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

Correct!

allows the client random access EFFICIENTLY to individual members. A vector

Correct!

gives the client the ability to insert an element EFFICIENTLY into the middle of the container. A (linked) list

Question 18

1 / 1 pts

Lazy Deletion is good for ...

(Check all that apply.)

Correct!

- [x] ... fast removal.
- [ ] ... fast insertion.

- [ ] ... conserving memory.

Correct!

- [x] ... suppressing frequent garbage collection for small numbers of removals.

Feedback

Lazy deletion allow for a fast and easy removal, but makes insertion slightly more complicated than it otherwise would be. It does not make the memory of the tree any more efficient. In fact it requires more memory since we are not releasing dead nodes unless and until we do a garbage collection. It does reduce the garbage collection, by its very definition.

Question 19

1 / 1 pts

AVL Trees ...

(Check all that apply.)

Correct!

- [x] ... for any given single search will deliver **O(log*N*)** search performance.
- [ ] ... might result in left- and right-subtrees having heights that differ by > 2.

Correct!

- [x] ... are reasonably efficient and predictable if we are frequently and randomly changing the data we are searching for from one access to the next.

- [ ] ... will leave the most recently inserted data/node (from a call to **insert()**) at its root position, just as it will always leave a successfully found data/node (from a call to **contains()** or **find()**) at the root of the tree.

Correct!

- [x] ... is a special kind of BST.

Feedback

Answers self-explanatory, except possibly for the one concerning efficiency under randomly changing searches.

Splay trees can radically alter the structure of a tree, creating unsatisfactory O(*N*) (compared with O(log*N*) search times, causing them to act like lists rather than trees. This is hard to control if the client is constantly searching for new data.

By comparison, AVL trees do not change their structure on searches, and area always balanced within two levels when measured from any node, including the root. Thus, they have a predictable and efficient log*N* search time.

Question 20

1 / 1 pts

An array contains N elements in sorted order. You can construct a Binary Search Tree from this array in:

(Pick only the tightest bound)

- ○ O(1) time

Correct!

- ◉ O(N) time
- ○ O(N^2) time

- O(N log N) time

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)

Quiz Score: **16.17** out of 20