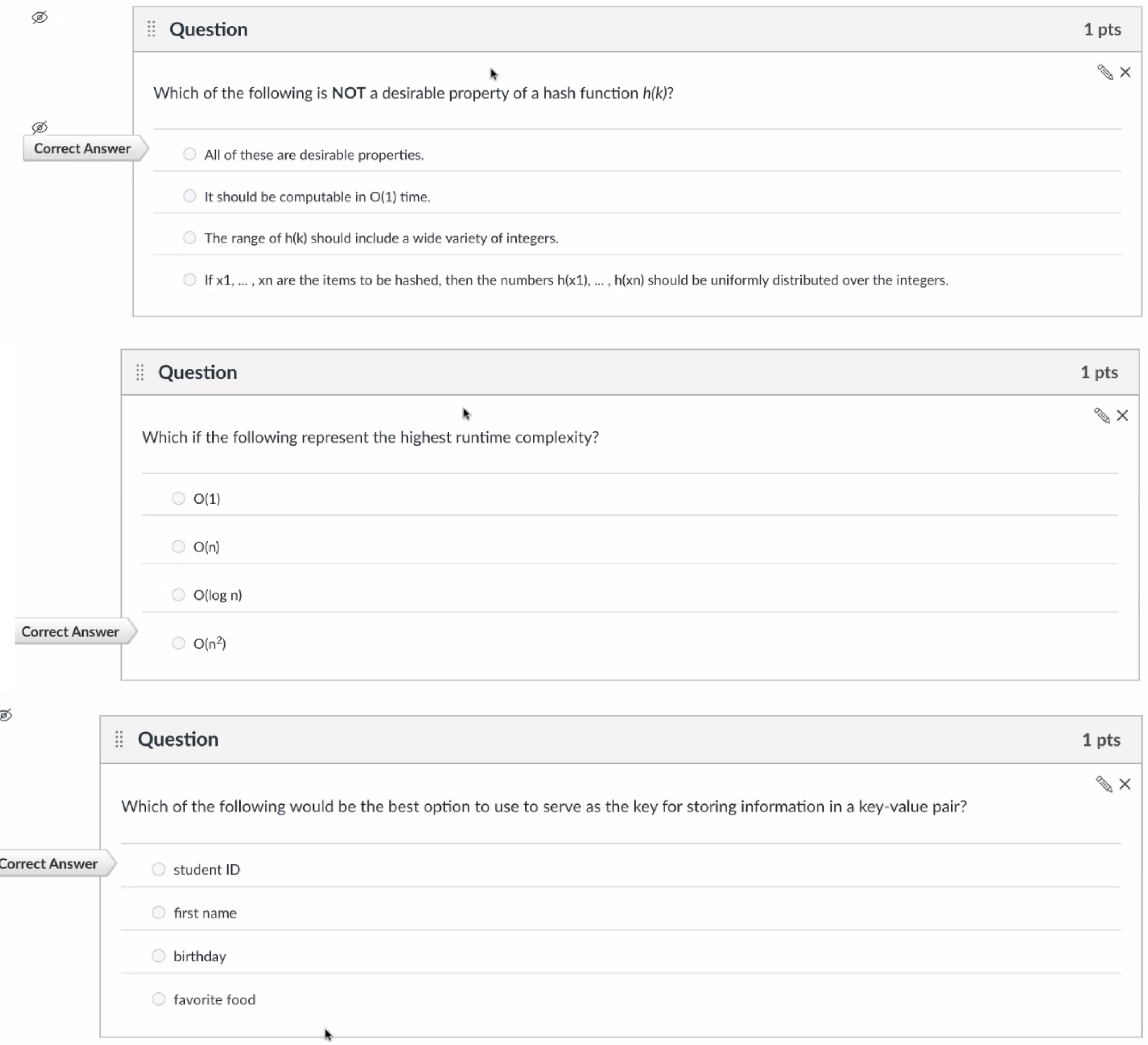


Question 1 pts

Which of the following is **NOT** a desirable property of a hash function *h(k)*?

Correct Answer
- [✓ All of these are desirable properties.]
- It should be computable in O(1) time.
- The range of h(k) should include a wide variety of integers.
- If x1, ... , xn are the items to be hashed, then the numbers h(x1), ... , h(xn) should be uniformly distributed over the integers.

Question 1 pts

Which if the following represent the highest runtime complexity?

- O(1)
- O(n)
- O(log n)

Correct Answer
- [✓ O(n^2)]

Question 1 pts

Which of the following would be the best option to use to serve as the key for storing information in a key-value pair?

Correct Answer
- [✓ student ID]
- first name
- birthday
- favorite food

Assuming all the supporting code that is necessary is available, what operation is the code below performing on a linked list?

```
void function (int val, int newVal) {
  if (head == nullptr) {
    return;
  }

  Node* current = head;
  Node* prev = nullptr;

  while (current != nullptr && current->data != val) {
    prev = current;
    current = current->next;
  }

  if (current == nullptr) {
    return;
  }

  Node* newNode = new Node(newVal);

  if (prev == nullptr) {
    newNode->next = head;
    head = newNode;
  } else {
    prev->next = newNode;
    newNode->next = current;
  }
}
```

General answer comments

This code is a function that takes 2 values. It traverses the list to find the first value and creates a new node with the second value and inserts it in the list before the node with the first value that was specified. It also does the usual checks to make sure the list is not empty and ensures that all the pointers are updated correctly.

Question — 1 pts

A hash tables can perform many operations like insert, delete, find and others very efficiently, this can only happen if the hash function and load factor have certain properties. What is the most important property?

Correct Answer → ○ The keys are distributed uniformly
- ○ It is fast
- ○ Each key is mapped to a unique location
- ○ All of the above

er

Question — 1 pts

n

In the linked list implementation of a stack, in the push operation, if new nodes are inserted at the front of the linked list, then when popping, nodes must be removed from the end of the linked list.

- ○ True

Correct Answer → ○ False

Question

1 pts

If the characters 'D', 'C', 'B', 'A' are placed in a queue (in that order), and then removed one at a time, in what order will they be removed?

er

- ABCD
- ABDC
- DCAB
- Correct Answer DCBA

Unanswered

Question

3 pts

Explain the concept of amortized analysis and how it is applied in algorithm analysis.

General answer comments

Amortized analysis is a method used to determine the average time complexity of a sequence of operations performed on a data structure. It involves analyzing the total cost of a series of operations over time, rather than individual operations.

ser

Unanswered

Question

3 pts

Describe a scenario where using a queue would be more appropriate than using a stack. Ensure that you show how data is manipulated in each data structure when you justify your choice for the scenario you specify.

General answer comments

There are many examples of scenarios where using a queue would be more appropriate than using a stack. One exmaple is in managing tasks or print jobs.

When you have a shared printer, multiple users may send print jobs to the printer at different times. A queue ensures that print jobs are processed in the order they were received, following the First-In-First-Out (FIFO) principle. This means that the first job to arrive is the first to be printed.

ser

Using a queue:

1. When a user sends a print job, it is added to the end of the queue.
2. The printer dequeues jobs from the front of the queue and processes them one by one.
3. New jobs are continuously added to the back of the queue as users send print requests.

n

This approach ensures fairness in print job processing, as all jobs are processed in the order they were received. It prevents any single user from monopolizing the printer's resources by submitting multiple jobs simultaneously.

In contrast, using a stack in this scenario would result in the Last-In-First-Out (LIFO) principle, where the most recent print job sent to the printer would be processed first. This would not be ideal in the example above, as it would prioritize newer jobs over older ones, potentially causing delays for users who submitted their print requests earlier.

Unanswered

Question

3 pts

Compare and contrast singly linked lists and doubly linked lists in terms of operations and memory overhead.

General answer comments

In a singly linked list each node knows which next node comes next in the list, while doubly linked list's nodes knows both which node comes next and which node is directly before it in the list. As such singly linked lists have lower memory overhead, storing half the number of pointers, but doubly linked lists allow traversal in both directions and easier deletion of nodes.

Unanswered

UnansweredUnansweredQuestion

8 pts

Part 1 *(5 points)*

Consider a hash table storing integer keys that handles collision using double hashing.

If N = 15

h(k) = k mod 15

d(k) = 11 -k mod 11

Insert the following keys in order into the hash table below 32, 64, 18, 33, 19

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		32	18	64			19							33

Part 2 *(3 points)*

Assuming your hash table is implemented as an array, write C++ code to implement the double hashing approach above. Assume you only have to implement the insert method and you have an array called table with each location initialized to -1. This method should take as input the int key. In the method you will need to do the following:

- compute the hash code
- detect a collision
- compute d(k)
- place the data correctly

Make sure your code treats the array as a circular array, and wraps around.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		32	18	64			19							33

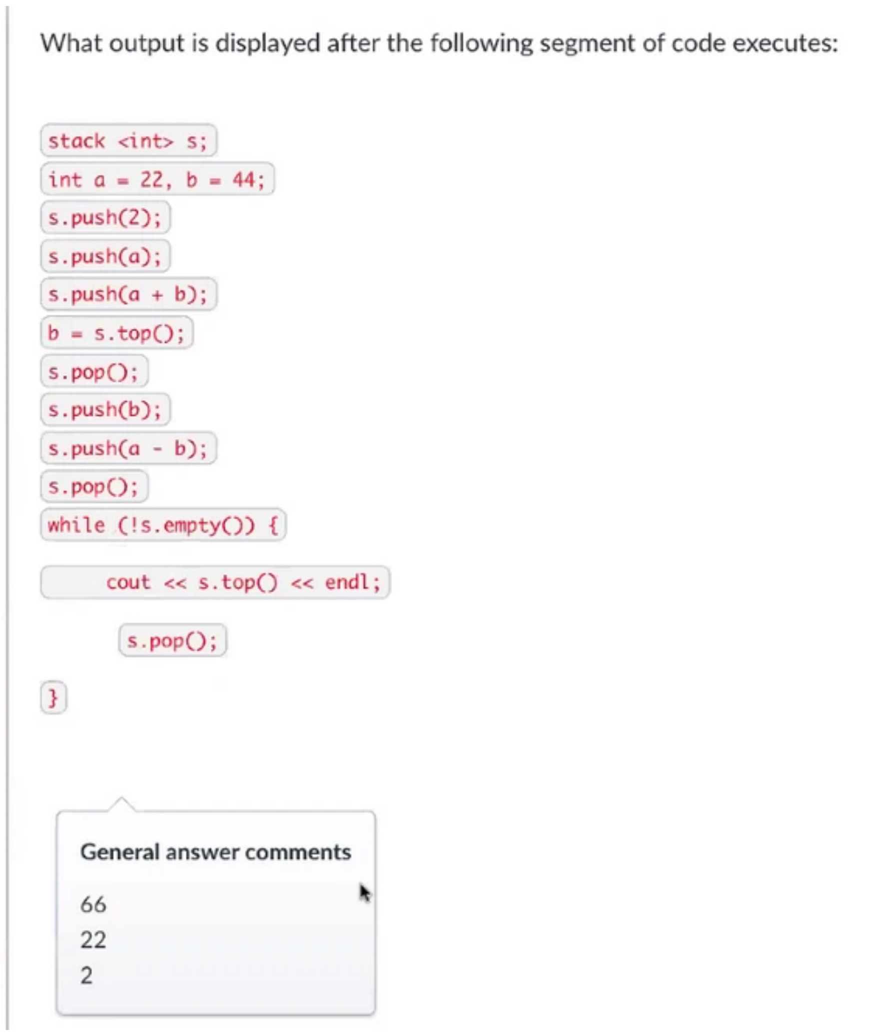
Part 2:

```
void insert(int key) {
    // Compute the hash code
    int index = key % 15;

    if (table[index] == -1) {
        // No collision, insert directly
        table[index] = key;
    } else {
        // Collision occurred, use double hashing with d(k) = 11 - k mod 11
        int step = 11 - (key % 11);
        int newIndex = (index + step) % 15;

        while (table[newIndex] != -1) {
            // Continue probing until finding an empty slot
            newIndex = (newIndex + step) % 15;
        }

        // Insert into the new index after collision resolution
        table[newIndex] = key;
    }
  }
```

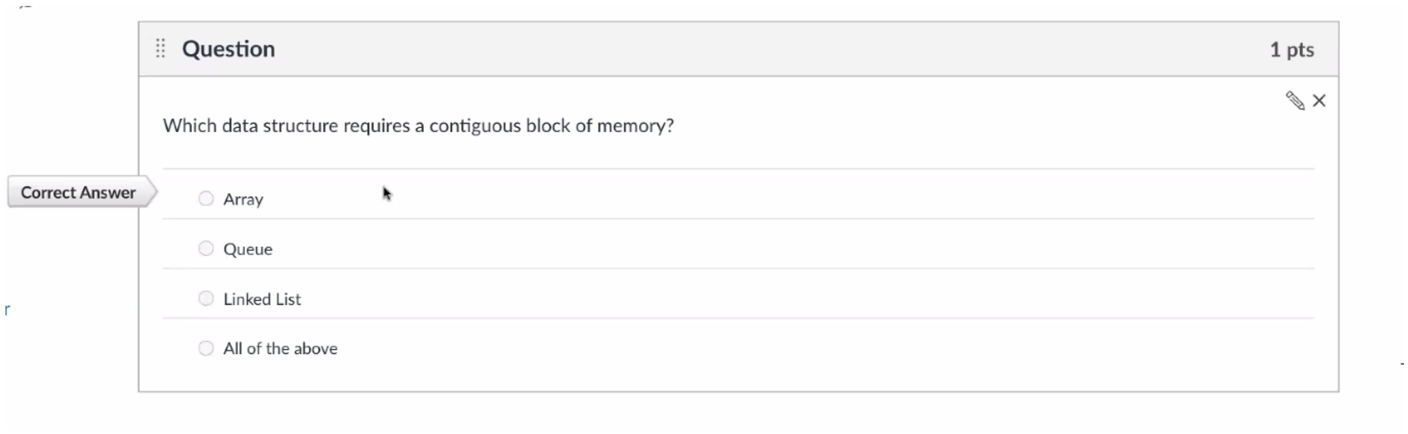


What output is displayed after the following segment of code executes:

```
stack <int> s;
int a = 22, b = 44;
s.push(2);
s.push(a);
s.push(a + b);
b = s.top();
s.pop();
s.push(b);
s.push(a - b);
s.pop();
while (!s.empty()) {
    cout << s.top() << endl;
    s.pop();
}
```

General answer comments

```
66
22
2
```

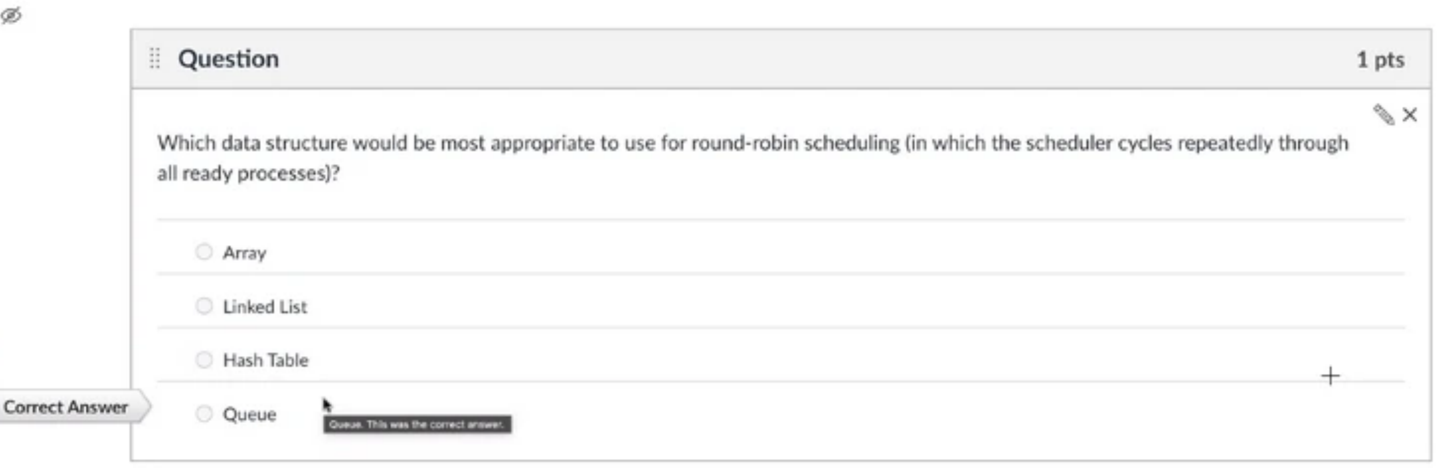


Question 1 pts

Which data structure requires a contiguous block of memory?

Correct Answer

- Array
- Queue
- Linked List
- All of the above



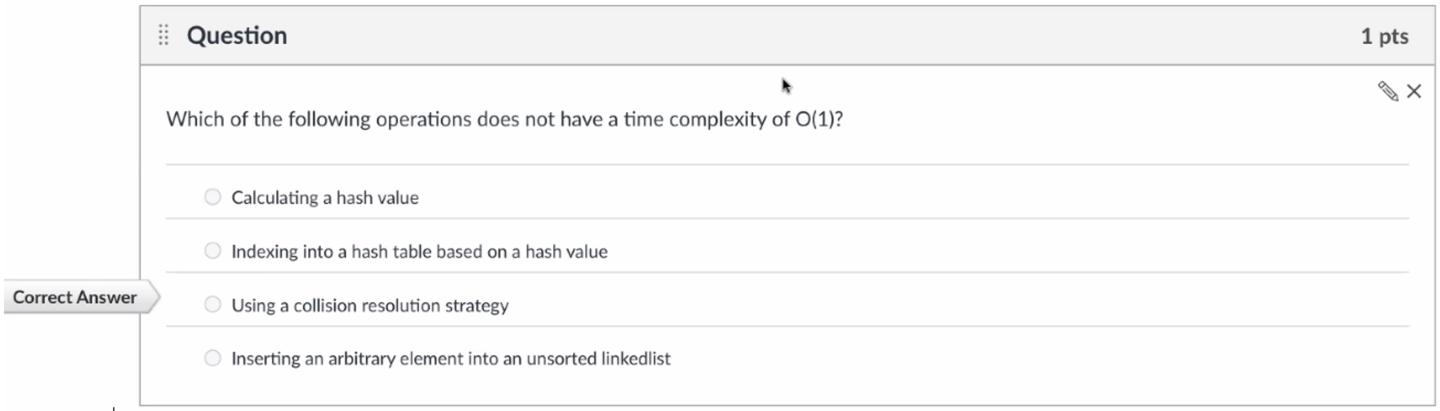
Question 1 pts

Which data structure would be most appropriate to use for round-robin scheduling (in which the scheduler cycles repeatedly through all ready processes)?

- Array
- Linked List
- Hash Table
- Queue

Correct Answer

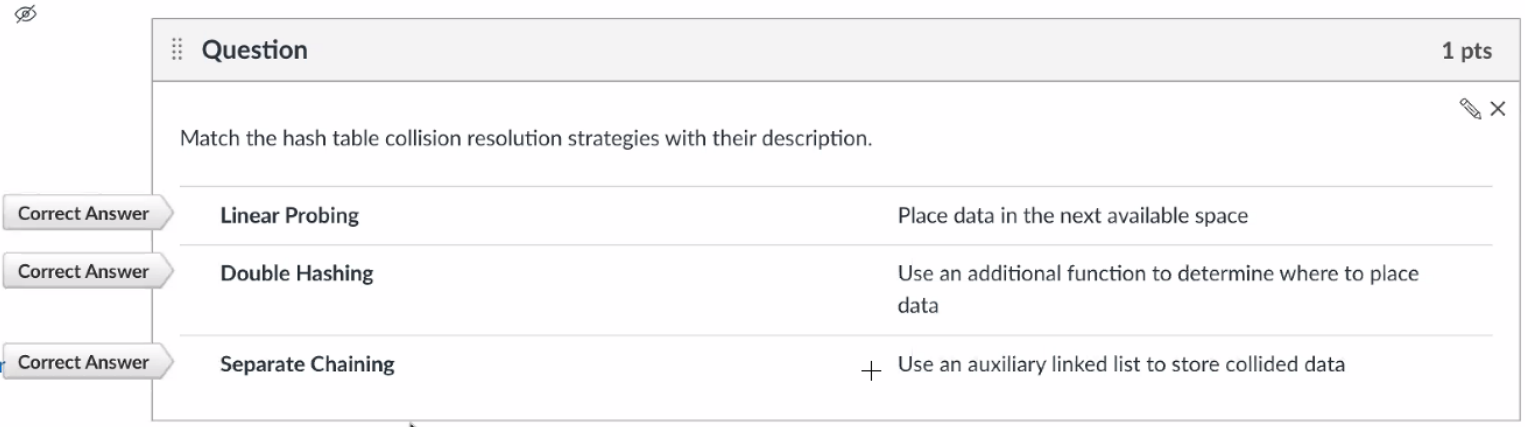
Queue. This was the correct answer.



Question **1 pts**

Which of the following operations does not have a time complexity of O(1)?

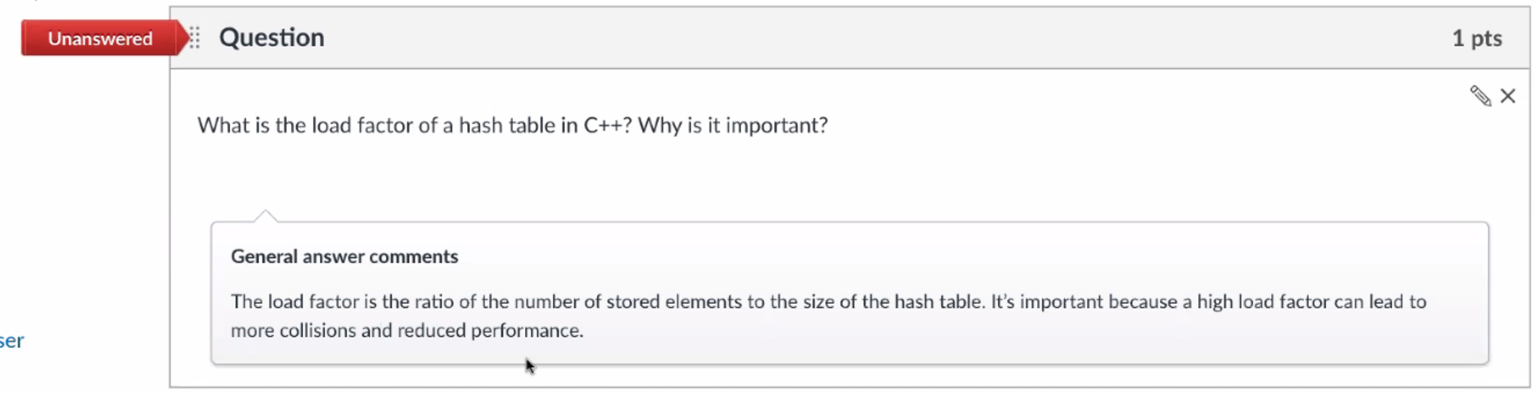
- ○ Calculating a hash value
- ○ Indexing into a hash table based on a hash value
- Correct Answer ○ Using a collision resolution strategy
- ○ Inserting an arbitrary element into an unsorted linkedlist



Question **1 pts**

Match the hash table collision resolution strategies with their description.

Correct Answer	**Linear Probing**	Place data in the next available space
Correct Answer	**Double Hashing**	Use an additional function to determine where to place data
Correct Answer	**Separate Chaining**	Use an auxiliary linked list to store collided data



Unanswered **Question** **1 pts**

What is the load factor of a hash table in C++? Why is it important?

General answer comments

The load factor is the ratio of the number of stored elements to the size of the hash table. It's important because a high load factor can lead to more collisions and reduced performance.

ser