1. What is the primary purpose of a hash function?
2. To sort elements in an array.
3. **To convert data into a fixed-size value.**
4. To allocate memory dynamically.
5. To encrypt data for secure storage.
6. #include <iostream>

#include <unordered\_map>

using namespace std;

int main() {

unordered\_map<string, int> hashTable;

hashTable["apple"] = 5;

hashTable["banana"] = 3;

hashTable["cherry"] = 8;

cout << hashTable["banana"] << endl;

return 0;

}

What will be the output of the program?

* 1. 5
  2. **3**
  3. 8
  4. It will result in a compilation error.

1. What is a collision in the context of hash tables?
   1. The process of resizing a hash table.
   2. **A situation where two different keys map to the same hash value.**
   3. The act of inserting an element into a hash table.
   4. The process of deleting an element from a hash table.
2. #include <iostream>

#include <unordered\_map>

using namespace std;

int main() {

unordered\_map<string, int> hashTable;

hashTable["apple"] = 5;

hashTable["banana"] = 3;

hashTable["cherry"] = 8;

cout << hashTable["orange"] << endl;

return 0;

}

What will be the output of the program?

* 1. **0**
  2. 3
  3. 8
  4. It will output a default value (0) and insert "orange" into the hash table.

1. Which property of a good hash function helps distribute keys evenly across the hash table?
   1. Collisions
   2. Clustering
   3. **Uniformity**
   4. Indexing
2. #include <iostream>

#include <unordered\_map>

using namespace std;

int main() {

unordered\_map<string, int> hashTable;

hashTable["apple"] = 5;

hashTable["banana"] = 3;

hashTable["cherry"] = 8;

hashTable.erase("banana");

cout << hashTable["banana"] << endl;

return 0;

}

What will be the output of the program?

* 1. 5
  2. **0**
  3. 8
  4. It will result in a compilation error.

1. What is the load factor of a hash table?
   1. The number of elements stored in the table.
   2. The size of the table.
   3. **The ratio of occupied slots to total slots in the table.**
   4. The number of collisions in the table.
2. Which of the following hash functions is most likely to cause clustering in a hash table?
   1. **h(k) = k % m**
   2. h(k) = floor(m \* (kA mod 1))
   3. h(k) = k
   4. h(k) = ((k / m) + k \* m) + k % m
3. Which collision resolution technique involves creating linked lists at each slot?
4. Linear Probing
5. Quadratic Probing
6. **Separate Chaining**
7. Double Hashing

#include <iostream>

#include <unordered\_map>

using namespace std;

int main() {

unordered\_map<int, string> hashTable;

hashTable[5] = "three";

hashTable[3] = "five";

hashTable[8] = "eight";

cout << hashTable[5] << endl;

return 0;

}

What will be the output of the program?

* 1. "five"
  2. **"three"**
  3. "eight"
  4. It will result in a compilation error.

1. In a hash table, what is the time complexity of inserting an element with separate chaining?
   1. **O(1)**
   2. O(log n)
   3. O(n)
   4. It depends on the load factor.
2. What will be the output if we pass key as 5 to the below function?

int total\_elements=13;

int getHash(int key){

return key % total\_elements;

}

* 1. 2
  2. **5**
  3. 3
  4. 1

1. Which hashing technique requires a secondary hash function to resolve collisions?
   1. Linear Probing
   2. Quadratic Probing
   3. Separate Chaining
   4. **Double Hashing**
2. What is the operation of following function?

char\* func(HashTable\* table, char\* key)

{

int index = hash\_function(key);

Ht\_item\* item = table->items[index];

// Provide only non-NULL values.

if (item != NULL)

{

if (strcmp(item->key, key) == 0)

return item->value;

}

return NULL;

}

* 1. Generate hash key
  2. **Search the key in hash table**
  3. Insert key in hash table
  4. Delete element from hash table

1. What is the worst-case time complexity of searching for an element in a hash table?
   1. O(1)
   2. O(log n)
   3. **O(n)**
   4. It depends on the load factor.
2. What is the operation of following function?

void func(int k, int v) {

int h = HashFunc(k);

while (t[h] != NULL && t[h]->k != k) {

h = HashFunc(h + 1);

}

if (t[h] != NULL)

delete t[h];

t[h] = new HashTableEntry(k, v);

}

* 1. Generate hash key
  2. Search the key in hash table
  3. **Insert key in hash table**
  4. Delete element from hash table

1. What is the primary disadvantage of using open addressing for collision resolution?
   1. It requires extra memory for linked lists.
   2. **It leads to clustering.**
   3. It cannot handle large datasets.
   4. It may require frequent resizing.
2. #include <iostream>

using namespace std;

int midHash(int key, int size) {

return (key + size / 2) % size;

}

int main() {

int key = 8;

int size = 10;

int hashValue = midHash(key, size);

cout << hashValue << endl;

return 0;

}

What will be the output of the program?

* 1. 0
  2. 4
  3. **3**
  4. 8

1. Which hashing technique aims to find the next available slot when a collision occurs?
   1. **Linear Probing**
   2. Quadratic Probing
   3. Separate Chaining
   4. Double Hashing
2. #include <iostream>

using namespace std;

int midHash(int key, int size) {

return (key + size / 2) % size;

}

int main() {

int key = 12;

int size = 6;

int hashValue = midHash(key, size);

cout << hashValue << endl;

return 0;

}

What will be the output of the program?

* 1. 0
  2. 1
  3. **3**
  4. 5

1. Which collision resolution technique involves using linked lists to store multiple values in a single hash table bucket?
   1. **Separate Chaining**
   2. Linear Probing
   3. Quadratic Probing
   4. Double Hashing

#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<int> hashTable(10, -1);

hashTable[3] = 42;

hashTable[6] = 19;

hashTable[9] = 56;

cout << hashTable[5] << " " << hashTable[6] << endl;

return 0;

}

What will be the output of the program?

* 1. **-1 19**
  2. 42 19
  3. -1 -1
  4. 19 56

1. In which collision resolution technique is the next available slot (bucket) searched in a linear manner when a collision occurs?
   1. Separate Chaining
   2. **Linear Probing**
   3. Quadratic Probing
   4. Double Hashing
2. For the given hash table, in what location will the element 58 be hashed using quadratic probing?

0 | 49

1 |

2 |

3 |

4 |

5 |

6 |

7 |

8 | 18

9 | 89

1. 1
2. **2**
3. 7
4. 6
5. Which collision resolution technique involves using a hash function to probe a sequence of buckets in a quadratic manner?
   1. Separate Chaining
   2. Linear Probing
   3. **Quadratic Probing**
   4. Double Hashing
6. What are the values of h1(k) and h2(k) in the hash function?

a)

h1(k) = m mod k

h2(k) = 1+ (m’ mod k)

b)

h1(k) = 1 + (m mod k)

h2(k) = m’ mod k

c)

h1(k) = 1+ (k mod m)

h2(k) = k mod m

**d)**

**h1(k) = k mod m**

**h2(k) = 1+ (k mod m’)**

1. In double hashing, how is the step size (increment) determined for probing?
   1. It is fixed for all keys.
   2. It is the same as the hash code of the key.
   3. **It is calculated using a secondary hash function.**
   4. It is determined by the size of the hash table.
2. At what position the number 72 gets inserted in the following table?

Index Key

0 22

1 34

2

3

4

5 56

6

7 18

8 41

9

10

1. 3
2. 10
3. 4
4. **6**
5. Which collision resolution technique aims to minimize clustering by spreading out the probing sequence?
   1. Separate Chaining
   2. Linear Probing
   3. Quadratic Probing
   4. **Double Hashing**
6. Where does the number 14 get inserted in the following table?

Index Key

0

1 79

2

3

4 69

5 98

6

7 72

8

9

10

11 50

12

1. 2
2. **9**
3. 4
4. 8
5. Which collision resolution technique tends to have more efficient insertion and deletion operations when the load factor is low?
   1. **Separate Chaining**
   2. Linear Probing
   3. Quadratic Probing
   4. Double Hashing
6. What the function of give code snippet?

bool func(int arr1[], int m, int arr2[], int n)

{

set<int> hashset;

for (int i = 0; i < m; i++) {

hashset.insert(arr1[i]);

}

for (int i = 0; i < n; i++) {

if (hashset.find(arr2[i]) == hashset.end())

return false;

}

return true;

}

* 1. Check if two arrays are equal
  2. **Check if arr2 is subset of arr1**
  3. Check if arr1 is subset of arr2
  4. Merge two arrays

1. Which collision resolution technique may suffer from primary clustering?
   1. Separate Chaining
   2. **Linear Probing**
   3. Quadratic Probing
   4. Double Hashing

int inline hash1(int value){

return value%TABLE\_SIZE;

}

int inline hash2(int value){

return PRIME - (value%PRIME);

}

Above functions are used in which type of hashing?

* 1. Linear probing
  2. **Double hashing**
  3. Quadratic probing
  4. Chaining

1. In separate chaining, what is the primary advantage over other techniques?
   1. Reduced memory usage
   2. **Fast lookup for all load factors**
   3. Efficient insertions and deletions
   4. Reduced likelihood of collisions
2. What will be returned by give function in hashing?

float func()

{

return (float)(this->numOfElements + 1) / (float)(this->capacity);

}

* 1. Number of elements in hash table
  2. Capacity of hash table
  3. **Load factor**
  4. Hash key

1. Which collision resolution technique is less susceptible to secondary clustering?
   1. Separate Chaining
   2. Linear Probing
   3. Quadratic Probing
   4. **Double Hashing**
2. What will be the function of below code snippet?

int bucketIndex = this->hashFunction(key);

node<T> \*bucketHead = this->arr[bucketIndex];

while (bucketHead != NULL)

{

if (bucketHead->key == key)

{

return bucketHead->value;

}

bucketHead = bucketHead->next;

}

return -1;

* 1. Insert key in hash table
  2. Delete key from hash table
  3. Generate hash value for key
  4. **Search key in hash table**

1. In quadratic probing, how is the probing sequence determined for a key with multiple collisions?
   1. By adding a fixed offset to the hash code
   2. By adding a linear increment to the hash code
   3. **By adding a quadratic increment to the hash code**
   4. By multiplying the hash code by a fixed factor
2. What will be the function of below code snippet?

int bucketIndex = hashFunction(mp, key);

struct node\* prevNode = NULL;

struct node\* currNode = mp->arr[bucketIndex];

while (currNode != NULL) {

if (strcmp(key, currNode->key) == 0) {

if (currNode == mp->arr[bucketIndex]) {

mp->arr[bucketIndex] = currNode->next;

}

else {

prevNode->next = currNode->next;

}

free(currNode);

break;

}

prevNode = currNode;

currNode = currNode->next;

}

* 1. Insert key in hash table
  2. **Delete key from hash table**
  3. Generate hash value for key
  4. Search key in hash table