HW: Hashing and Locality of Reference in C++

# 1. Hashing in C++

Hashing is a technique used to map data of arbitrary size to fixed-size values, usually for the purpose of fast data retrieval.   
The basic concept of hashing is to use a hash function to convert the key into an index in a hash table, which is an array-like data structure.  
The ideal hash function distributes keys uniformly across the hash table to minimize collisions (when two keys hash to the same index).   
The process of resolving collisions can be done using techniques like chaining or open addressing.  
  
The following code demonstrates how to implement a simple hash table using separate chaining in C++:

#include <iostream>  
#include <list>  
using namespace std;  
  
class HashTable {  
private:  
 int BUCKET;  
 list<int>\* table;  
  
public:  
 HashTable(int b) {  
 BUCKET = b;  
 table = new list<int>[BUCKET];  
 }  
  
 int hashFunction(int x) {  
 return (x % BUCKET);  
 }  
  
 void insertItem(int key) {  
 int index = hashFunction(key);  
 table[index].push\_back(key);  
 }  
  
 void deleteItem(int key) {  
 int index = hashFunction(key);  
 table[index].remove(key);  
 }  
  
 void displayHash() {  
 for (int i = 0; i < BUCKET; i++) {  
 cout << i;  
 for (auto x : table[i])  
 cout << " --> " << x;  
 cout << endl;  
 }  
 }  
};  
  
int main() {  
 int keys[] = {15, 11, 27, 8, 12};  
 int n = sizeof(keys) / sizeof(keys[0]);  
  
 HashTable ht(7);  
 for (int i = 0; i < n; i++)  
 ht.insertItem(keys[i]);  
  
 ht.displayHash();  
  
 return 0;  
}

# 2. Locality of Reference in C++

Locality of reference is a concept that refers to the tendency of a program to access a relatively small portion of its address space at any given time.   
There are two types of locality: temporal locality and spatial locality.  
- Temporal locality: When a data item is accessed, it is likely to be accessed again in the near future.  
- Spatial locality: When a data item is accessed, nearby data items are likely to be accessed soon.  
  
Locality of reference is important for optimizing cache usage in memory systems because programs that exhibit good locality of reference can make better use of caches, leading to faster performance.  
  
Here's an example in C++ that shows how locality of reference can affect performance.   
The first loop exhibits good spatial locality, while the second loop exhibits poor locality due to the way memory is accessed.

#include <iostream>  
#include <chrono>  
using namespace std;  
using namespace std::chrono;  
  
int main() {  
 const int N = 10000;  
 int array[N][N];  
  
 // Initialize array  
 for (int i = 0; i < N; i++) {  
 for (int j = 0; j < N; j++) {  
 array[i][j] = i + j;  
 }  
 }  
  
 // Measure time with good locality (row-major order)  
 auto start1 = high\_resolution\_clock::now();  
 long long sum1 = 0;  
 for (int i = 0; i < N; i++) {  
 for (int j = 0; j < N; j++) {  
 sum1 += array[i][j];  
 }  
 }  
 auto end1 = high\_resolution\_clock::now();  
 auto duration1 = duration\_cast<milliseconds>(end1 - start1).count();  
 cout << "Time with good locality (row-major order): " << duration1 << " ms" << endl;  
  
 // Measure time with poor locality (column-major order)  
 auto start2 = high\_resolution\_clock::now();  
 long long sum2 = 0;  
 for (int j = 0; j < N; j++) {  
 for (int i = 0; i < N; i++) {  
 sum2 += array[i][j];  
 }  
 }  
 auto end2 = high\_resolution\_clock::now();  
 auto duration2 = duration\_cast<milliseconds>(end2 - start2).count();  
 cout << "Time with poor locality (column-major order): " << duration2 << " ms" << endl;  
  
 return 0;  
}