# CSC2059 Data Structures and Algorithms

## Practical 8: Searching

Wednesday 1st December 2021

Launch Visual Studio. Create a new Solution Practical8 and name the first project Search1. In project Search1 create a source file test1.cpp.

## Project 1. Some simple searches

**(1)** In test1.cpp write a function

```
int find_first_not_in_range(int* pi, int N, int L, int U);
```

The function takes an array pointed by pi of size N and searches for the first number that doesn't fall in the range [L, U] with specified lower and upper bounds L, U. When successful the function returns the index of the number searched for; otherwise it returns -1.

Write a main function in which create an int array of size N filled with random numbers within the range 0-100. Test your function with variable values for N (e.g., 10, 100, …), and variable bounds [L, U] (e.g., [20, 50], [10, 60], …). At the end of the test print out the following statement:

```
cout << "The complexity of find_first_not_in_range is " << "O(XXX)" << endl;
```

where XXX is what you think the complexity is in terms of the order of N.

**(2)** In test1.cpp write another function

```
void print_duplicate (int* pi1, int N1, int* pi2, int N2);
```

The function takes two arrays, pi1 and pi2, of size N1 and N2, respectively. It finds the elements in pi1 that are also found in pi2 and prints these elements out as they are being found. Rename the above main to main1 and write a new main function, in which you take N1, N2 from keyboard and use these to create two arrays with random numbers, to test the above function. At the end of the test print out the following statement:

```
cout << "The complexity of print_duplicate is " << "O(XXX)" << endl;
```

where XXX is what you think the complexity is in terms of the order of the size of the problem.

## Project 2. Binary search for linked list

In Solution Practical8 add a new project Search2, in which add the provided ListNode.h and List.h classes, and also add a source file test2.cpp.

**(1)** In test2.cpp, write a function

```
int binarySearch(List<int>* plist, int first, int last, int key);
```

The function takes a pointer to an integer linked list, plist, and performs binary search for the given key in the list, from the position first to the position last. The function returns the position of the key in the list when it is found (position 0 for the first node of the list), or -1 if the key is not found.

In test2.cpp write a main function to test the binary search. You take the list size N from keyboard, create a list of N sorted int values by adding 0 to N – 1 into the list by using the insert_end function, and then call the above function to search over the complete list for keys that are within or outside the list.

**(2)** In test2.cpp, write code to perform linear search for the same keys for the list by using

(a) the get(int p) function

(b) the setp_first() and get_next() functions

to cycle through the list to read its node values. Compare the times between these two methods, and with the above binary search method.

## Project 3. Hash tables

**(1)** In Solution Practical8 add a new project Search3, in which add the provided HashTable.h, PhoneDir.h, PhoneDir.cpp and test3.cpp.

The hash table class (HashTable.h) is a template class created by us. We use this example to demonstrate to you the potential near O(1) performance for retrieving information from a hash table. Assume that we look for a person's phone number by giving the person's name as a searching key, from a hash table storing phone directories (simulated by the PhoneDir class).

Run test3.cpp, which allows you to search a hash table created to store persons' name-phone number records. Increase the hash table size, which indicates how many records may be in the table, from 11 to some larger numbers (e.g., 6137, 91193, 391929, …). You do not see the time taken to find the record is lengthened as the size increases (whether the person you search for is in the table or out of the table). Of course this performance may be unrealistic given that there may be hash key collisions between different names. How to design good hash functions and choose good content keys to reduce the probability of collision are some of the topics of current research.

**(2)** In test3.cpp, write a function

```
size_t hash_index(string date, string hotelname, string address, string customerref,
size_t tableSize);
```

which will be used to calculate the hash indices to implement the database for bookings made by customers on a hotel booking travel web site, where tableSize is the hash table size. To keep the average running time as small as possible, only a maximum number of ten letters from the hotel address are used. All four fields are combined in the given order to form the search key. The hash index is expected to distribute well so you should use a good hash function (e.g. the one introduced in the lecture notes).

*If you have any questions, ask a demonstrator or a teaching staff face to face in the labs, or online by using MS Teams, during the timetabled practical session time. At any other time outside the timetabled session time, you can send your questions by using the Ticketing System, or by using emails.*