National University of Computer and Emerging Sciences, Lahore Campus

Course:	Data Structures	Course:	CS 2001
Program:	BS(CS)	Semester:	Fall 2023
Due Date	06-Dec-2023 at 11:59pm	Total Marks:	40
Type:	Assignment 5		

Important Instructions:

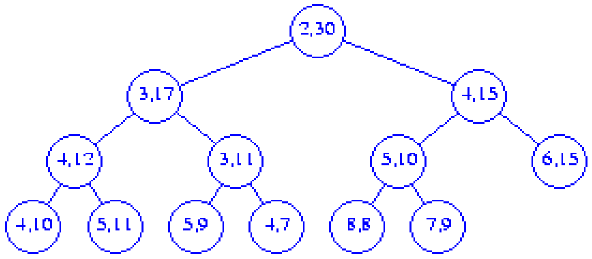
1. Submit your code in a zip file named as your roll number.
2. You are not allowed to copy solutions from other students. We will check your code for plagiarism using plagiarism checkers. If any sort of cheating is found, negative marks will be given to all students involved.
3. Late submission is not allowed.

Question 1: [Marks: 20]

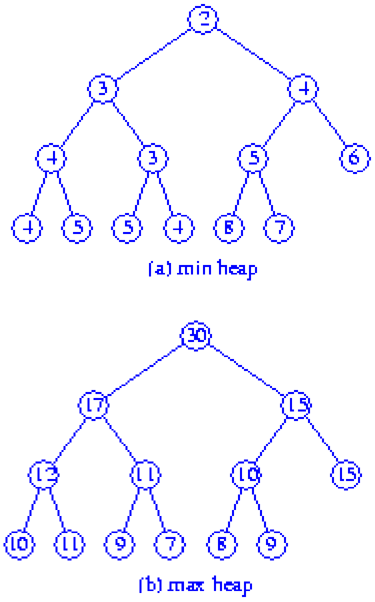
An interval heap is a binary heap in which each node contains two elements (except the last node). It is a completebinary tree in which:

- The left element is less than or equal to the right element.
- Both the elements define a closed interval.
- Interval represented by any node except the root is a sub-interval of the parent node.
- Elements on the left hand side define a min heap.
- Elements on the right hand side define a max heap.

Below is an example of interval heap with 26 elements where interval [3, 17] is a sub-interval of its parent node i.e. [2,30]



Sample Interval Heap

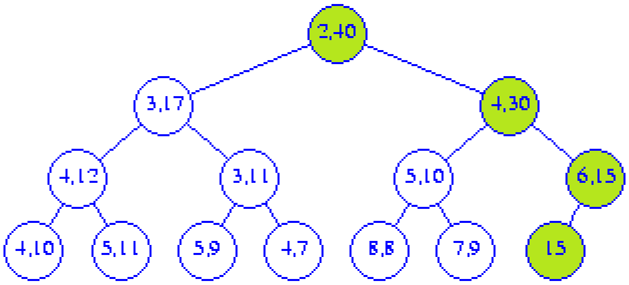


(a) min heap

(b) max heap

Corresponding Min and Max Heap

Every update operation (insert or delete) must preserve all the properties mentioned above. A new node is created in the Insertion operation if number of elements in the heap is even and no new node is created if number of elements inthe heap are odd. For example if we want to insert the element 40 in the heap given in Figure 1 then resulting heap should be as follows:



Consider the following definition of class IntervalHeap.

```
class IntervalHeap{
        int hsize;   //number of elements in heap;
        int maxsize;   //size of the array
        int **h;
public:
        IntervalHeap(int s=100){
                maxsize = s;
                hsize = 0;
                h = new int*[2];
                h[0] = new int[maxsize];//store the lower end of closed interval
                h[1] = new int[maxsize];//store the higher end of closed interval
};
```

You are required to implement the following:

1. Write a function *Insert(int d)* in class IntervalHeap that inserts data element d in the interval heap.
2. Similarly give the implementation of update, delete and display member functions for the interval heap class.

Question 2: (HashMap with Linear Probing) **[Marks :20]**

Implement a HashItem struct which represents an item in a hash array.

```
template <class v>
struct HashItem
{
	int key;
	v value;
	short status;

};
```

status variable can have 0, 1 or 2. 0 means empty, 1 means deleted, 2 means occupied. Status variable will be used by get and delete methods of HashMaps implemented in the next questions. The default value assigned to a HashItem is 0 (empty).

Now implement a HashMap class whose basic definition is as follows (You can add any helping member variables and methods):

```
template <class v>
class HashMap
{
private:
	HashItem<v>* hashArray;
	int capacity;
	int currentElements;
	virtual int getNextCandidateIndex(int key, int i)

	public:
	HashMap();
	HashMap(int const capacity);
	void insert(int const key, v const value);
	bool deleteKey(k const key) const;
	v* get(k const key) const;
	~HashMap();
};
```

1. HashMap(): constructor assigns a capacity of 10 to hashArray.

2. `HashMap(int const capacity):` an overloaded constructor that assigns the capacity of given capacity to hashArray. If capacity is less than 1 return error via assert(capacity>1)

3. `void insert(int const key, v const value):`
   a. The insert method inserts the value at its appropriate location. Find the first candidate index of the key using:

      index= key ***mod*** *capacity*

   b. To resolve hash collision, it will use the function getNextCandidateIndex(key, i) to get the next candidate index. If the candidate index also has collision, then getNextCandidateIndex will be called again with an increment in i. getNextCandidateIndex will be continued to call until we find a valid index. Initially i will be 1.
   c. If the loadFactor becomes 0.75, then it will call the doubleCapacity method to double the capacity of array and rehash the existing items into the new array.

4. `void doubleCapacity():` A private method which doubles the capacity of hash array and rehashes the existing items. Use getNextCandidateIndex method to resolve collision.

5. `virtual int getNextCandidateIndex(int key, int i):` a private and virtual method that uses linear probing to return the next candidate index for storing the item containing key k. Linear probing means that it will simply add i to the hash value of key. This method does not check whether the candidate index has collision or not.

6. `bool deleteKey(k const key) const:` this method deletes the given key. It returns true if the key was found. If the key was not found it returns false. When the key is found, simply set the status of the hashitem containing the key to deleted (value of 1). It also uses status variable to search for the key intelligently.

7. `V* get(k const key) const:` this method returns a pointer to the corresponding value of the key. If the key is not found, it returns nullptr. It also uses status variable to search for the key intelligently.

8. `~HashMap(): destructor`

Also implement the following.

- Create a class **QHashMap** which inherits the HashMap class implemented above. Override the `getNextCandidateIndex(int key, int i) method so that it performs quadratic probing, i.e., add the square of i to the hash value of key.`

- Create a class **DHashMap** which inherits the HashMap class implemented above. Override the `getNextCandidateIndex(int key, int i) method so that it performs double hashing and returns the candidate index. Double hashing will be performed as follows:`

  *first_value= key **mod** capacity*
  *second_value= (PRIME - (key **mod** PRIME))* (PRIME is any prime number.)
  *candidate index= (first_value + i*second_value) **mod** capacity*

- Create a global function populateHash which is passed a filename as parameter and a HashMap object by pointer `void populateHash(string filename, HashMap<string> *hash).` The function reads <id, name> pairs and populates the hash with those pairs. The key is id.

Now run the following main program:

```
#include <iostream>
using namespace std;
#include <string>

int main()
{

        HashMap<string> *map;
        map=new HashMap<string>;
        populateHash("students.txt", map);
        cout<<*map->get(9);
        map->deleteKey(9);
        assert(map->get(9)==nullptr);
        delete map;
        map=new QHashMap<string>;
        populateHash("students.txt", map);
        cout<<*map->get(98);
        map->deleteKey(98);
        assert(map->get(98)==nullptr);
        delete map;

        map=new DHashMap<string>;
        populateHash("students.txt", map);
        cout<<*map->get(101);
        map->deleteKey(101);
        assert(map->get(101)==nullptr);
        delete map;

        return 0;

}
```