

**HAND IN**  
answers recorded  
on question paper



Section	001	002
Last Name		
First Name		
ID		

Q1		20
Q2		15
Q3		10
Q4		30
Q5		30
Q6		25
Q7		25
Q8		25
Q9		20
Total		200

Faculty of Engineering and Applied Science  
Electrical & Computer Engineering Department  
**ELEC 278**  
**Fundamentals of Information Structures**  
**Fall 2016**  
**Final Examination**  
December 19<sup>th</sup> 2016  
7:00 to 10:00pm  
David Athersych & Hesham Farahat

**Instructions:**

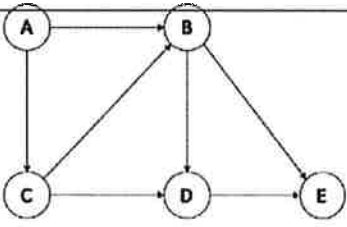
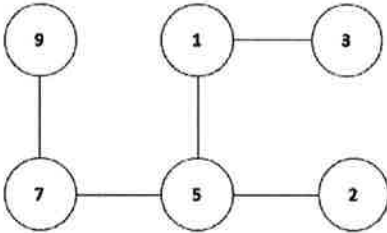
- Write your name and ID on every page. Circle your section number in the table above (1 → Hesham, 2 → David)
- Comment your code . It may help you to get partial marks if the code is not correct.
- There are 9 questions, with multiple parts. Read each question carefully, and clearly provide your solution in the space provided. If more space is required, then use the back of the page, and clearly indicate this.
- If a question is not clear to you, then write down any assumptions that you make in your solution approach.
- Casio 991 is the only approved calculator to be used in this exam.
- Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer questions as written

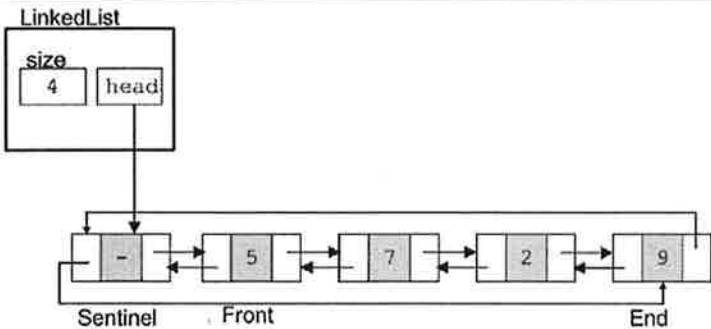
Good luck!

This material is copyrighted and is for the sole use of students registered in ELEC278 and writing this exam. This material shall not be distributed or disseminated. Failure to abide by these conditions is a breach of copyright and may also constitute a breach of academic integrity under the University Senate's Academic Integrity Policy Statement.

## Question 1

Solve the following True/False and Multiple choice questions by writing a single answer in the allocated space.

#	Question	Answer
1	Printing a minheap using inorder traversing, will print the integers in ascending order	
2	Breadth-first traversal uses a queue data structure, whereas depth-first uses a stack	
3	The degree of a vertex in a graph equals to the number of edges to connected to that vertex	
4	<pre>void f(double x) {     if (x == 0)         return;     printf("%d,",x);     f(x / 2.0); }</pre> <p>The output for f(8) is: 8,4,2,1</p>	
5	Item x is at index 569 in a minheap, at what level item x is in? Assume that the root is at level 0 A) 15 B) 10 C) 5 D) 9	
6	<p>The longest path from A to B is of length (number of edges in the path):</p> <p>A) 1 B) 2 C) 3 D) 4</p>	
7	<p>The complexity of merge sort algorithm is:</p> <p>A) <math>O(n)</math> B) <math>O(n^2)</math> C) <math>O(n \log n)</math> D) <math>O(\log n)</math></p>	
8	<p>What is this data structure?</p> <p>A) Tree B) Graph C) Tree and graph D) None of the above</p>	
9	Item x is at index 568 in a minheap, What is the index of its parent? A) 283 B) 284 C) 283.5 D) 567	

10	Stacks are LIFO, whereas Dequeues are FIFO	
11	<p>LinkedList</p>  <p>What is the complexity of adding to the end in this type of structure?</p> <p>A) <math>O(n)</math>  B) <math>O(n^2)</math>  C) <math>O(\text{long } n)</math>  D) <math>O(1)</math></p>	
12	<p>What is the complexity of adding in the middle type of structure of question 11?</p> <p>A) <math>O(n)</math>  B) <math>O(n^2)</math>  C) <math>O(\text{long } n)</math>  D) <math>O(1)</math></p>	
13	Radix sort is always faster than Quick sort	
14	Malloc operation creates a space in the memory's heap	
15	A common use of stacks is network routers to store incoming packets	
16	The complexity of insertion into an AVL tree is $O(\log n)$	
17	Hash Tables can always be used to find data in $O(1)$	
18	<p>What is the 5<sup>th</sup> probe of hashing key = 234535 into a table of size 17 using double hashing (<math>h(x) = \text{key} \bmod 17</math> and <math>h'(x) = \lfloor x/17 \rfloor</math>) :</p> <p>A) 12  B) 14  C) 3  D) 5</p>	
19	<p>Given the preorder traversal of a BST as 4, 2, 1, 3, 6, 7. What is the post order traversal?</p> <p>A) 1, 2, 3, 4, 6, 7  B) 1, 3, 2, 7, 6, 4  C) 4, 2, 6, 1, 3, 7  D) None of the above</p>	
20	Computation that can be implemented using recursion can also be implemented using iteration.	

## Question 2

Assume that a stack structure of integers is already defined and ready to be used by your code. However, the structure can only be accessed by 4 functions:

1- **void initS(Stack \*s)**

This function takes a pointer to a stack in memory and initializes it.

2- **void push(Stack \*s, int x)**

This function takes a pointer to a stack and an integer. It pushes the value x into the stack.

3- **int pop(Stack \*s, int \*res)**

This function takes a pointer to a stack, and a pointer to an integer. It pops a value from the stack and stores the value into memory location *res* is pointing to.

4- **int isEmptyS(Stack \*s)**

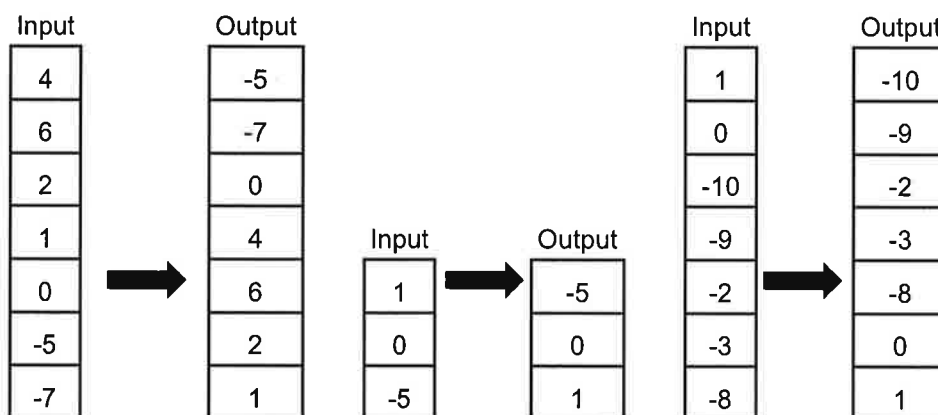
This function takes a pointer to a stack and checks if it is empty or not. If it is empty, the function will return 1 and 0 otherwise.

Using these functions only and without knowing the internal implementation of the stack, write C code for a function that takes a stack of integers with the following properties:

- a. All negative integers are at the lower half of the stack
- b. There will be at least one negative integer
- c. All positive integers are at the upper half of the stack
- d. There will be at least one positive integer
- e. There is exactly one 0 that separate the negative and positive integers

The function will take this stack and change the order by switching negative and positive halves such that all positive integers are at the lower half and all negative integers are at the upper half. The order of integers in one half should not change.

Examples:



```
void switchStack(Stack* s){
```

```
}
```

--

## Question 3

10
----

Write a recursive function `gcd(a,b)` in C that finds the Great Common Divisor of two integers A and B using Euclid's algorithm:

$$\text{gcd}(x, y) = \begin{cases} \text{gcd}(y, x), & x < y \\ x, & y = 0 \\ \text{gcd}(y, x \bmod y), & \text{otherwise} \end{cases}$$

```
int gcd(int x, int y){
```

```
}
```

Show the steps of running `gcd(54,24)`:

## Question 4

Suppose a particular problem requires that you store a set of data based on a number. For example, suppose you want to record corrosion spots on the inside of a long pipe, and you want to report a count for every centimeter. Your pipe is generally in good condition, and so most of the data is 0. A typical pipe is 4000 kilometers long, so a full set of data will be 400,000 values. A typical set of data may have only 100 values that are not 0.

You want to keep your memory usage as small as possible, so you decide to store only the non-zero values, and store them in a linked list instead of an array. Each item in your linked list will have to have the data value, along with the index associated with the value. You decide to call this a linked-list array or LArray. Your LArray will point to a linked list of LArrayData items.

The structure to describe one LArray is declared as follows:

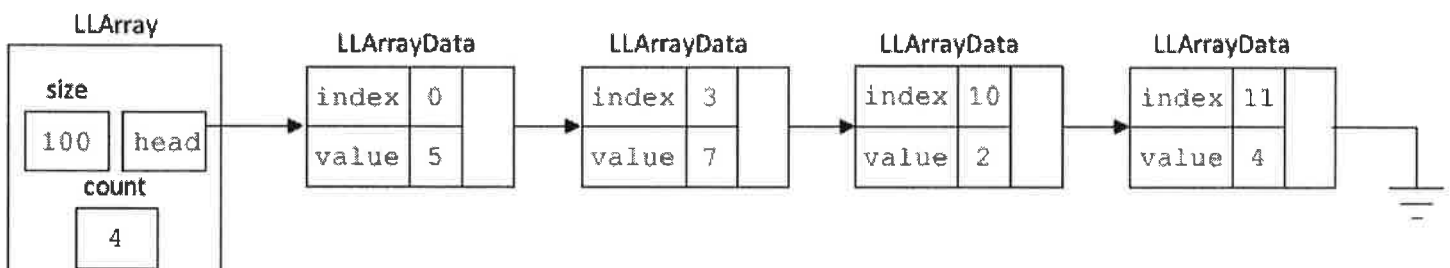
```
typedef struct LArray {
    int    count;        // how many items are actually in the linked list
    int    size;         // maximum size of the equivalent array
    LArrayData *head;    // pointer to list of actual data items
} LArray;
```

For clarity, the size field indicates the maximum number of items that can be in the linked-list array (in other words, the index is from 0 to size-1), and count just indicates how many data items are actually in the array.

A data structure to hold actual data is declared as follows:

```
typedef struct LArrayData {
    int    index;        // what array index this data would be at
    int    datavalue;    // actual data value at this array index
    LArrayData *next;    // pointer to next linked list element
} LArrayData;
```

A Linked List for 4 items will look like:



Note that:

- The next of the last item points to NULL
- If the list is empty then head points to NULL and count = 0

You decide to implement the following functions in C:

**LArray \*setup (int n)**

This function sets up (initializes) a linked-list array capable of holding n items. The function returns a pointer to the LArray that has been set up.

```
int getdata (LLArray *p, int index, int *value)
```

This function fetches the value at the position indicated by index in the linked-list array pointed to by p. It is equivalent to `*value = p[index]` if a real array was being used. If there is a data value stored for the specified index, then that value is stored in the place pointed to by value. If there is no value stored, then a 0 is stored in the place pointed to by value. The function returns 0 if the index is invalid and 1 if an assignment was made to the place pointed to by value.

```
int setdata (LLArray *p, int index, int value)
```

This function places the value at the position indicated by index in the linked-list array pointed to by p. It is the equivalent of `p[index] = value` if a real array was being used. If there is already a value for the index in the linked-list array, then the new value replaces it. If there is no value stored for the specified index, then an `LLArrayData` structure is allocated and added to the linked list. The function returns 0 if an error occurred – such as an index out of range – and 1 if everything worked. Examples of calling `setdata` on the above figure:

- If index = 1, value = 3, then insert new `LLArrayData` between nodes 0 and 3
- If index = 9, value = 3, then insert new `LLArrayData` between nodes 3 and 10
- If index = 12, value = 4 then insert new `LLArrayData` after last node
- If index = 0, value = 6 then change value of the first node to 6

```
LLArray *setup (int n){
```

```
}  
int getdata (LLArray *p, int index, int *value){
```

```
}
```



```
int setdata (LLArray *p, int index, int value){
```

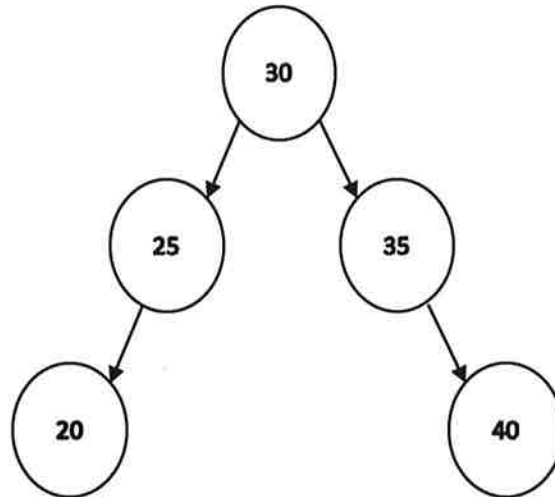
```
}
```

## Question 5

A) Insert the following integers sequentially in the below Binary Search Tree.

21 50 55 60 19 18 17 15

1) Show the final tree.



2) Print the tree using Preorder traversal

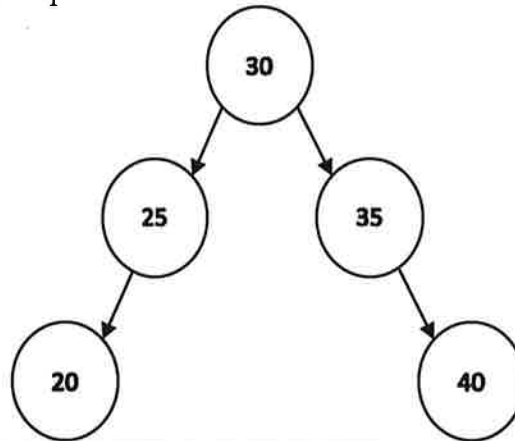
3) Print the tree using Postorder traversal

4) Print the tree using inorder traversal

5) Print the tree using breadth-first traversal

- B) Insert the following integers sequentially in the below AVL tree. After each insert, illustrate the state of the tree. If necessary, perform any required rotation, then state which node has caused the unbalancing and which of the 4 possible rotations was applied.

Note: An incorrect tree structure at any step will result in a loss of marks for subsequent steps.



<b>Insert 21</b>	Unbalanced Node
	Rotation Name
<b>Insert 50</b>	Unbalanced Node
	Rotation Name
<b>Insert 55</b>	Unbalanced Node
	Rotation Name

<b>Insert 60</b>	Unbalanced Node
	Rotation Name
<b>Insert 19</b>	Unbalanced Node
	Rotation Name
<b>Insert 18</b>	Unbalanced Node
	Rotation Name

<b>Insert 17</b>	Unbalanced Node
	Rotation Name
<b>Insert 15</b>	Unbalanced Node
	Rotation Name

## Question 6

- A) Suppose you have the following hash function, and you are using it to generate locations in an array where to store the key values. You are storing your keys in an array that can hold 1100 integers.

```
int hashCode (int key)
{
    int n;
    n = key / 100;
    n = n % 1000;
    n = n + key%100;
    return n;
}
```

You use this function to compute hash keys for the following data:

1043266, 1046636, 745048, 347029, 1050002, 644063, 540399, 640199, 1442279, 239998,  
249012, 1142082, 49220

Show what index will actually be used to store each of these key assuming that you use linear probing. Fill the probe cells when applicable.

Key	1 <sup>st</sup> Probe	2 <sup>nd</sup> Probe	3 <sup>rd</sup> Probe	4 <sup>th</sup> Probe	5 <sup>th</sup> Probe	6 <sup>th</sup> Probe	7 <sup>th</sup> Probe	Index
1043266								
1046636								
745048								
347029								
1050002								
644063								
540399								
640199								
1442279								
239998								
249012								
1142082								
49220								

- B) Suggest a modification to the hashcode function that will reduce the number of collisions. Indicate what index will be used to store each data item for your new algorithm.

Key	1 <sup>st</sup> Probe	2 <sup>nd</sup> Probe	3 <sup>rd</sup> Probe	4 <sup>th</sup> Probe	5 <sup>th</sup> Probe	6 <sup>th</sup> Probe	7 <sup>th</sup> Probe	Index
1043266								
1046636								
745048								
347029								
1050002								
644063								
540399								
640199								
1442279								
239998								
249012								
1142082								
49220								

- C) What is meant by double-hashing. Could double-hashing reduce the collisions? Why or why not?

## Question 7

A) The following code can be used to perform a bubble sort on an array of integers.

```
void swap(int *a, int g, int h)
// Parameter a points to array of integers. Routine swaps values in
// positions g and h
{
    int t = a[g]; a[g] = a[h]; a[h] = t;
}
void bubbleSort(int* a, int n) {
    int i, j, swapped;
    for (i = 0; i < n; ++i) {
        swapped = 0;
        for (j = n - 1; j > i; --j) {
            if (a[j] < a[j - 1]) {
                swapped = 1;
                swap(a, j, j-1);
            }
        }
        if (swapped == 0)
            break;
    }
}
```

Suppose you have student data stored as described by the following C structure definition:

```
typedef struct Record {
    int    student_number;
    char   name[64];
} Record;
```

The actual data is stored in an array of these structures:

```
Record dr [1000]; // Actual data held in 1000 records
```

You are also given three functions (you do not have to write these):

- **void printrecord** (Record \*p)

This function prints the contents of one data record – the one its parameter points to.

- **int compare\_records\_by\_name** (Record \*p1, Record \*p2)

This function takes pointers to two records, performs a comparison of the names in the records and returns -1 if the first Record (the one pointed to by p1) is less than the second (the one pointed to by p2), 0 if the two records are equal, and 1 if the first Record is greater than the second.



You want to be able to access the student data in student name sorted order (alphabetically) but you don't want to actually move the elements in the array. You will use a second array that contains pointers to all the elements in the data array, as follows:

```
Record *ptrs [1000];
```

This second array is initialized like this:

```
int i;  
for (i=0; i<1000; i++)  
    ptrs [i] = &dr[i];
```

Using the code given at the beginning of this question as a model, write the code for a bubble sort that will move the pointers in the ptrs array, so that it becomes an array of pointers to the data in sorted order. That is, when done, code like this:

```
for (i=0; i<1000; i++)  
    printrecord (ptrs[i]);
```

can be used to print all the records in order, while

```
for (i=0; i<1000; i++)  
    printrecord (&dr[i]);
```

will print all the records in the original order.

```
void swap(Record* a[], int g, int h){
```

```
}
```

```
void bubbleSort(Record* a[], int n) {
```

```
}
```

B) You have the following data in an integer array:

300 235 315 40 210 2 510 51 6 1 510 107 937 195 57

1) Show what the array would look like after 3 iterations of a bubble sort ( 3 iterations of the outer loop)

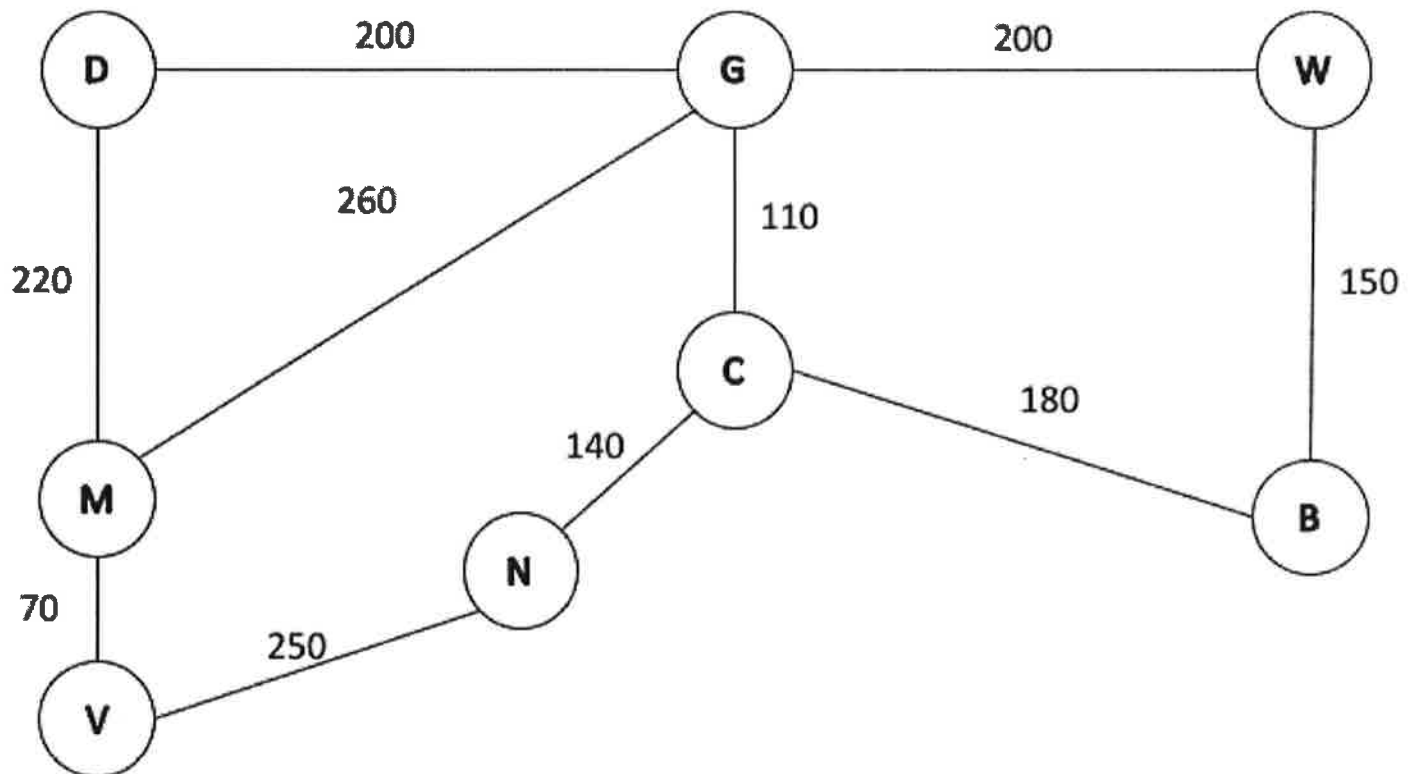
2) Sort the array using radix sort.

## Question 8

Professor X's office is at Walter Light Hall (W). He visits 7 places in his weekly schedule:

- |                  |                   |                     |
|------------------|-------------------|---------------------|
| 1- Gordon (G)    | 4- Victoria (V)   | 6- Campus Store (C) |
| 2- Dunning (D)   | 5- Nixon Field(N) | 7- Biosciences (B)  |
| 3- Mac-Corry (M) |                   |                     |

As one of his brightest students, he gave you the map of the campus and the distances between the places in meters and asked you some questions.



A) Represent the map using adjacency matrix. Order the vertices alphabetically.

- B) Identify a cycle of size 8 (consist f 8 edges):
- C) Identify all paths from C to N
- D) Find the shortest route from Walter Light Hall to all other places. Use Dijkstra to solve this problem. Order the vertices alphabetically in the table.
- E) Using the final table from Dijkstra, find the shortest path from W to V. What is the total distance traveled?

--

## Question 9

20
----

Assume that a minHeap structure of integers is already defined and ready to be used by your code.

1- Definition:

```
typedef struct Heap {  
    int* a;           //array of integers  
    int last;        //index of last element in the heap  
    int maxSize;     //maximum allowed space  
} Heap;
```

2- Heap\* **initHeap**(int size)

Creates an empty heap with maximum size and returns a pointer to the allocated space

3- **int withdrawMin**(Heap\* h)

Removes and returns the minimum value of the heap

4- **void insert**(Heap\* heap, **int** x)

Inserts new value to the heap

5- Heap\* **heapify**(**int** a[], **int** size)

Builds a heap out of an array a.

6- **void swap**(**int**\* a, **int** i, **int** j)

Swaps a[i] with a[j]

7- **void reheapUp**(Heap\* heap, **int** index)

8- **void reheapDown**(Heap\* heap, **int** i)

Develop two new functions to be added to the minHeap structure, findItem() and deleteItem().

A) Call Heapify on this array { 23, 7, 92, 6, 12, 14, 40, 44, 20, 21 } How will the array will look like after heapifying? Show your steps.

- B) **int findItem(Heap\* h, int x)** Searches the heap for the first occurrence of the argument x, and then returns the index of that location. If x is not found in the heap, return -1.

```
int findItem(Heap* h, int x){
```

```
}
```

- C) **void removeItem(Heap\* h, int i)** Removes the item at location i from the heap and fix the heap after the removal. Use the same approach used in withdrawMin function.

```
void removeItem(Heap* h, int i){
```

```
}
```