Student name: Quan Hong Doan

Student ID: 986956

Assignment: Lab 11

Week: 11

Due: Dec 30, 2019

**Lab 11**

**Problem 1**:

Starting with the values 1, 2, 4, 4, 5, 6, 9, 11, 12, 12, 17, do the following:

1. Create a heap H in which these values are the keys.
2. Perform the insertItem algorithm to insert the value 7 into H. Show all steps.
3. Perform the removeMin algorithm on H and show all steps.
4. Represent H in the form of an array A.
5. Perform the array-based insertItem algorithm to insert 14 into A – show all

steps.

1. Perform the array-based removeMin algorithm on A – show all steps.

**Answer**:

1. Create Heap

Create heap from array with the same size: [1, 2, 4, 4, 5, 6, 9, 11, 12, 12, 17]

Insert 1

Insert 2

parent=1, current=2=>no swap

Insert 4

parent=1, current=4=>no swap

Insert 4

parent=2, current=4=>no swap

Insert 5

parent=2, current=5=>no swap

Insert 6

parent=4, current=6=>no swap

Insert 9

parent=4, current=9=>no swap

Insert 11

parent=4, current=11=>no swap

Insert 12

parent=4, current=12=>no swap

Insert 12

parent=5, current=12=>no swap

Insert 17

parent=5, current=17=>no swap

Heap in array: [1, 2, 4, 4, 5, 6, 9, 11, 12, 12, 17]

1. insertItem(7)

Current HeapSize = 7, which is the initial heap size.

Insert 7

Heap is full, insertion abort.

1. removeMin()

Remove minimum value

Remove root node = 1

SiftDown for current=17, minimum=2

SiftDown for current=17, minimum=4

SiftDown for current=17, minimum=11

Updated heap in array: [2, 4, 4, 11, 5, 6, 9, 17, 12, 12, 17]

1. print to array

Updated heap in array: [2, 4, 4, 11, 5, 6, 9, 17, 12, 12, 17]

1. insertItemArrayBase(14)

Insert 14

parent=5, current=14=>no swap

Updated heap in array: [2, 4, 4, 11, 5, 6, 9, 17, 12, 12, 14]

1. removeMinArrayBased()

Remove minimum value

Remove root node = 2

SiftDown for current=14, minimum=4

SiftDown for current=14, minimum=5

SiftDown for current=14, minimum=12

Updated heap in array: [4, 5, 4, 11, 12, 6, 9, 17, 12, 14, 14]

Implementation:

**private** **int** getLeftChildIndex(**int** nodeIndex) {

**return** 2 \* nodeIndex + 1;

}

**private** **int** getRightChildIndex(**int** nodeIndex) {

**return** 2 \* nodeIndex + 2;

}

**private** **int** getParentIndex(**int** nodeIndex) {

**return** (nodeIndex - 1) / 2;

}

**public** **void** insertItem(**int** value) {

**if** (heapSize == data.length) {

System.***out***.println("Heap is full");

} **else** {

heapSize++;

data[heapSize - 1] = value;

siftUp(heapSize - 1);

}

}

**private** **void** siftUp(**int** nodeIndex) {

**int** parentIndex, tmp;

**if** (nodeIndex != 0) {

parentIndex = getParentIndex(nodeIndex);

**if** (data[parentIndex] > data[nodeIndex]) {

tmp = data[parentIndex];

data[parentIndex] = data[nodeIndex];

data[nodeIndex] = tmp;

siftUp(parentIndex);

}

}

}

**public** **int** removeMin() {

**int** min = Integer.***MIN\_VALUE***;

**if** (isEmpty()) {

System.***out***.println("Heap is empty");

} **else** {

min = data[0];

data[0] = data[heapSize - 1];

heapSize--;

**if** (heapSize > 0) {

siftDown(0);

}

}

**return** min;

}

**private** **void** siftDown(**int** nodeIndex) {

**int** leftChildIndex, rightChildIndex, minIndex, tmp;

leftChildIndex = getLeftChildIndex(nodeIndex);

rightChildIndex = getRightChildIndex(nodeIndex);

**if** (rightChildIndex >= heapSize) {

**if** (leftChildIndex >= heapSize)

**return**;

**else**

minIndex = leftChildIndex;

} **else** {

**if** (data[leftChildIndex] <= data[rightChildIndex]) {

minIndex = leftChildIndex;

} **else** {

minIndex = rightChildIndex;

}

}

**if** (data[nodeIndex] > data[minIndex]) {

tmp = data[nodeIndex];

data[nodeIndex] = data[minIndex];

data[minIndex] = tmp;

siftDown(minIndex);

}

}

**Problem 2**:

Carry out the array-based version of HeapSort on the input array

[1, 4, 3, 9, 12, 2, 4]

Show steps and outputs along the way. Make sure to distinguish between Phase I and

Phase II of the algorithm.

**Answer**:

Phase 1: Build Heap

Insert 1

Insert 4

parent=1, current=4=>no swap

Insert 3

parent=1, current=3=>no swap

Insert 9

parent=4, current=9=>no swap

Insert 12

parent=4, current=12=>no swap

Insert 2

parent=3, current=2=>swap and siftUp

SiftUp between parent=3, current=2

parent=1, current=2=>no swap

Insert 4

parent=2, current=4=>no swap

Heap in array: [1, 4, 2, 9, 12, 3, 4]

Phase 2***:*** Extract minimum element from heap one-by-one, and replace it with the last element in the array. The extracted value will be put in the returning array in sequence.

Remove minimum value 1

SiftDown for current=4, minimum=2

SiftDown for current=4, minimum=3

Remove minimum value 2

SiftDown for current=4, minimum=3

Remove minimum value 3

SiftDown for current=12, minimum=4

SiftDown for current=12, minimum=9

Remove minimum value 4

SiftDown for current=12, minimum=4

Remove minimum value 4

SiftDown for current=12, minimum=9

Remove minimum value 9

Remove minimum value 12

HeapSort in Array-based: [1, 2, 3, 4, 4, 9, 12]

Implementation of sorting function (based on the problem 1 implementation).

**public** **int**[] heapSortArrayBased() {

**int** len = heapSize;

**int**[] arr = **new** **int**[len];

**for** (**int** i = 0; i < len; i++) {

arr[i] = removeMin();

}

**return** arr;

}

**Problem 3**:

The recursive version of BottomUpHeap relies on the Proposition given below. For

this exercise, prove the Proposition.

Proposition. Suppose n is a positive integer of the form for some h. Then n may

be written in the form n = 1 + m + m for some m. Moreover, m must equal .

**Answer**:

Let assume the statement is true.

Then from the form , we have

Which is the initial assumption

**Problem 4**:

Carry out the steps of the recursive algorithm BottomUpHeap for the input sequence {11, 5, 2, 3, 17, 24, 1}.

**Answer**:

Size of the array = 7, build BottomUpHeap: [11, 5, 2, 3, 17, 24, 1], start from floor(size/2), call restoreDown until node 1. During the call, we compare the node with left and right children. Which child is greater, we move up, if both are greater, move the largest child up.

Given array: [11, 5, 2, 3, 17, 24, 1]

Call Restore Down

left=17, right=24, current arr[i]=2

Right index < heapSize

Left child < right child -> swap current with right

Array updated: [11, 5, 24, 3, 17, 2, 1]

Call Restore Down

left=24, right=3, current arr[i]=5

Right index < heapSize

Left child > right child -> swap current with left

Right index < heapSize

Left child > right child -> swap current with left

Array updated: [11, 24, 17, 3, 5, 2, 1]

Call Restore Down

left=11, right=24, current arr[i]=11

Right index < heapSize

Left child < right child -> swap current with right

Right index < heapSize

Left child > right child -> swap current with left

Right index < heapSize

Current > left child && current > right child

Returning Heap: [1, 2, 5, 3, 11, 17, 24]

**Problem 5**:

1. Prove the following facts about numbers:
   1. 3 | – 1 for any x that is not a multiple of 3
   2. 5 | – 1 for any x that is not a multiple of 5

Rewrite these statements using mod notation. Then make a guess about the

General result: What general fact are these two problems special cases of?

1. 

**Answer**:

1. Proving
   1. for any x is not multiple of 3.

We know that x can be written as:

x = 3k, or x = 3k + 1 or x = 3k +2

So:

* Or
* Or

From the above statements, or or is a mod of 3.

If is a multiple of 3, then x is a multiple of 3. Therefore in the 2 remaining cases of x, we have for any x is not multiple of 3.

* 1. for any x is not multiple of 5.

For product of any 5 consecutive integer numbers is divisible by 5, we can write as

If x is not multiple of 5, then should be a multiple of 5, or

is a multiple of 5 since the factor is 5.

Therefore, is divisible by 5, in other words, we have our statement is true.

1. Proving

From

The perfect square is 1, if we don’t consider 0 is a factor to be counted as perfect square.

The 3 items of are the remaining for any x divided by 3, in other words, we have

or or

Because of the condition that , therefore x can be only 0 or 1,

where and

From , the perfect square numbers are 1 and 4, if we don’t consider 0 as perfect squre.

For any integer 0 < x < p, such that like , then for odd prime p, there are (p+1)/2 quadratic residues (counting 0), and (p-1)/2 non-residues. Those residues are in form of . To answer the question, there will be (p+1)/2 quadratic residues.