**MaxMinHeap Implentation – Gal Nagli – OpenU – Data Structures Course 2020**

**General overview:**

In the MaxMinHeap assignment my work methodology was to implement the known structures of max and min heap, with the regular algorithms of Heapify,Insert,Delete,Extract.

The idea behind the Max-Min Heap implementation is to determine when I encounter an odd level, and therefore I need to tweak some changes to the regular structure of heap.

I have used an array which I have initialized which stores the heap nodes.

Many of the functions I have implemented are dependent on the specific node relation's with it's childs/grandchild | parent/grandparent, therefore I have created more than a few functions which will help the code look more elegant.

I have inserted into this word Document the example of running the program, each algorithm Time complexity and the connection between every function to it's sub-functions.

I have been based in my work on this paper : <http://akira.ruc.dk/~keld/teaching/algoritmedesign_f03/Artikler/02/Atkinson86.pdf>

Which shows the pseudo-code implementation of min-max heap.

**a) Finding the maximum key in O(1):**

במבנה הנתונים אשר יצרתי, ערימת המקסימום מינימום, נשמרות התכונות הנתונות:

* לכל צומת בעומק זוגי, ערכו גדול מ- או שווה לערכו של כל אחד מצאצאיו.
* לכל צומת בעומק אי- זוגי, ערכו קטן מ- או שווה לערכו של כל אחד מצאצאיו.

מכיוון שאנחנו מתייחסים לשורש המבנה להיות בעומק זוגי (0), ערך השורש גדול או שווה לערכו של כל אחד מצאצאיו, לכן בהכרח – ערך השורש הוא ערך המקסימום:

maxvalue = root (the root)

**Finding the minimum key in O(1):**

שנית, כפי שהסברתי לעיל, כל צומת בעומק אי-זוגי ערכו קטן או שווה לערכו של כל אחד מצאצאיו, לכן ניתן לקבוע כי הערך המינימאלי בערימה הינו אחד מילדיו של השורש, הילד השמאלי או הילד הימני, לכן בעזרת השוואה אחת אוכל לקבוע מהו ערך המינימום, וכך אמצא אותו בזמן קבוע.

If root.leftson > root.rightson

Return root.rightson

Else

Return root.leftson

**b)** **Images of the implementation of each function of the MaxMinHeap.**

I will be inserting the input to my program by "input.txt" which is been submitted as well in the ZIP file, which contain the following sequence:

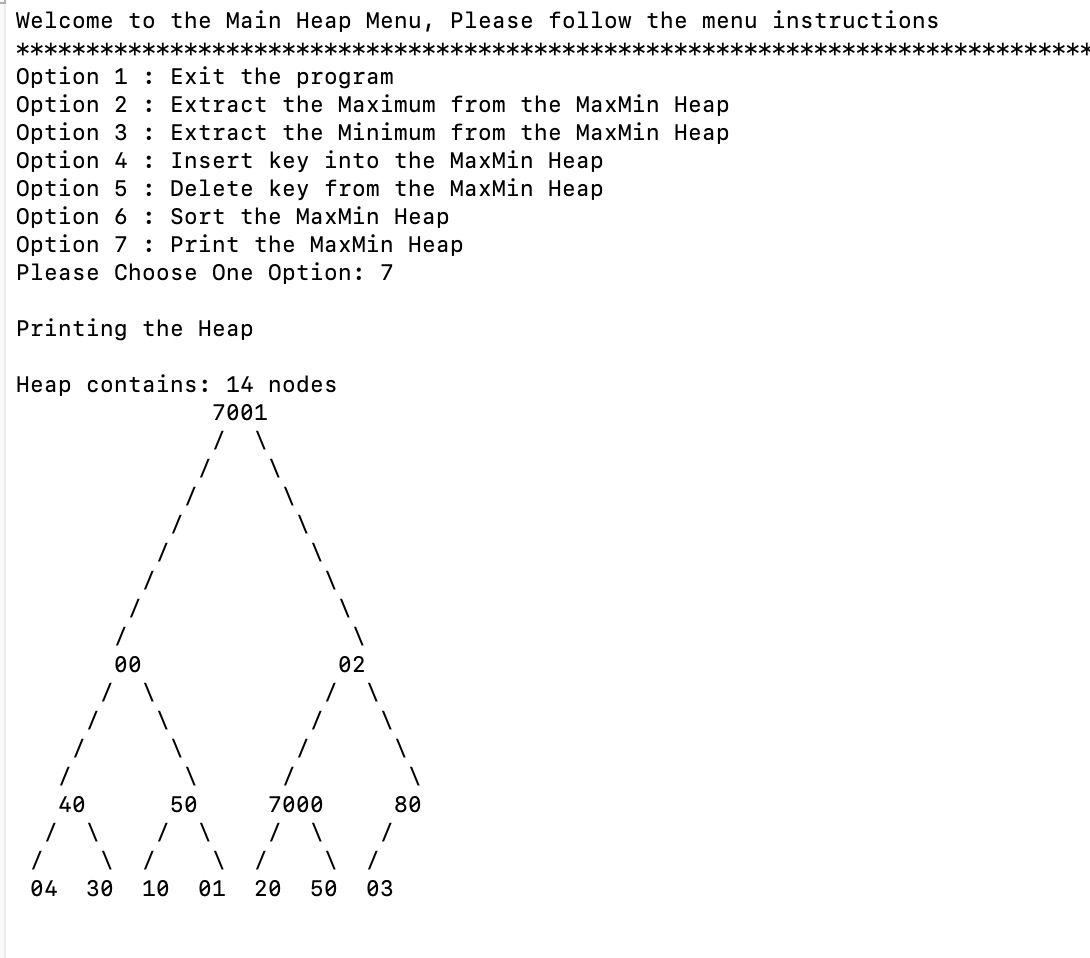
"10 20 30 40 1 2 3 4 50 7000 0 50 7001 80"

In order to run my program, type in the folder's terminal :

**python3 MaxMinHeap.py**

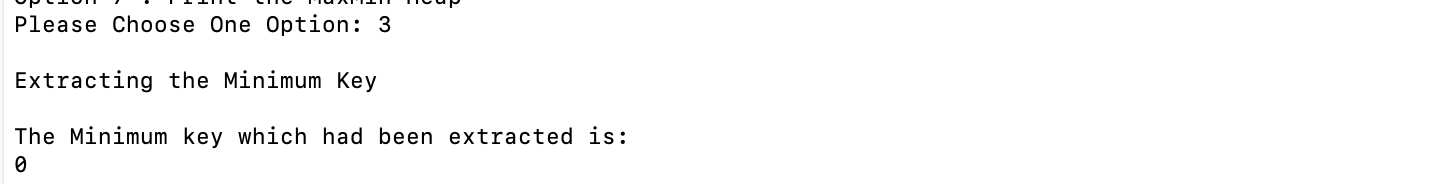
1. **Welcome Menu:** תמונה שמכילה צילום מסך

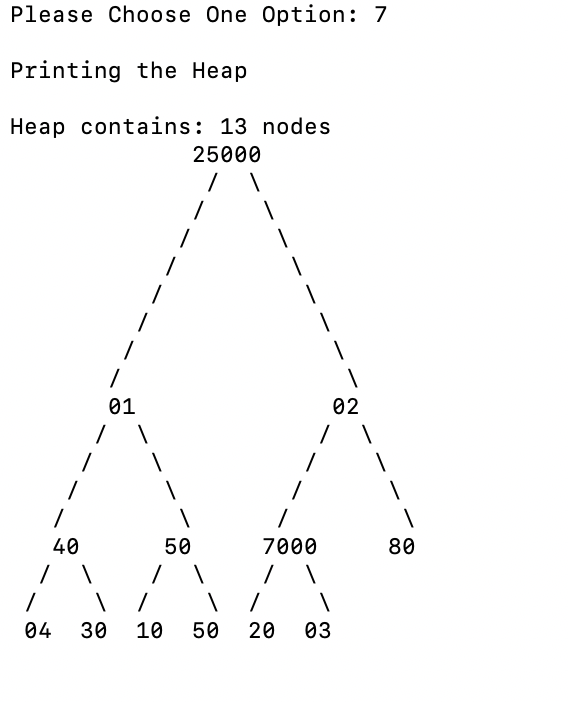
   התיאור נוצר באופן אוטומטי
2. **Building with file input : "Pressing 3", make sure you type the file name correctly:** תמונה שמכילה צילום מסך

   התיאור נוצר באופן אוטומטי
3. **Now we are introduced to the Main Menu as requested.**
4. **Pressing "7" : Printing the MaxMinHeap:** 
5. **Pressing "1" : Exiting the program:** תמונה שמכילה סכין, שולחן

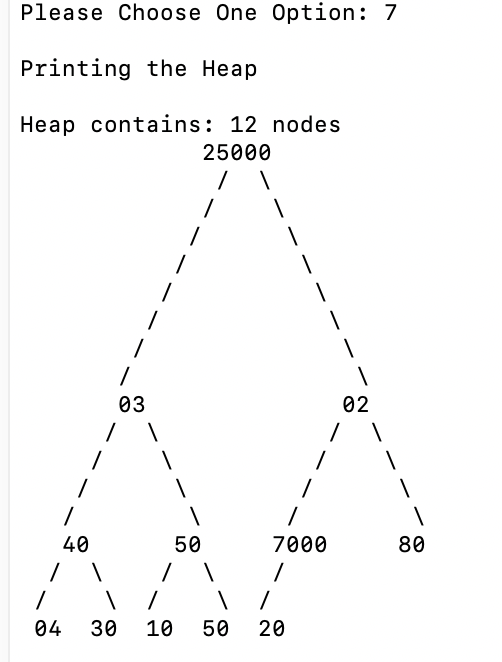
   התיאור נוצר באופן אוטומטי
6. **Pressing "2" HEAP-EXTRACT-MAX function:**



1. **Pressing "3" HEAP-EXTRACT-MIN function:** 
2. **Pressing "4" HEAP-INSERT function:** תמונה שמכילה שעון

   התיאור נוצר באופן אוטומטי
3. **Pressing "5" HEAP-DELETE function:** תמונה שמכילה ציפור

   התיאור נוצר באופן אוטומטי



1. **Pressing "6" HEAP-SORT Function:** תמונה שמכילה ציפור

   התיאור נוצר באופן אוטומטי

**c) Explaining each function implemented, and it's complexity.**

**1) HEAPIFY(self, i)**

**TIME Complexity: O(lgn):**

The way that I have implemented the Heapify function is with dividing the problem into 2, one case scenario is when I am on Odd level, and the second one is when im at Even level.

If im on Odd level, it means I need to heapify as I look to the min values, and the opposite when I'm on Even level.

I have created two functions to help with the task, maxHeapify and minHeapify, which performing almost the identical commands.

I need to identify in the process if I have grandchilds which values are bigger/smaller than the current node, if so, I would have to swap the values, in order to arrange my heap, as this is the purpose of the HEAPIFY algorithm, to arrange the heap upside down.

The time complexiry is O(lgn), because each of my sub-functions max and min heapify are almost as exact as in the book, as described in page 110.

**2) BUILD-HEAP(self)**

**TIME Complexity: O(nlgn):**

My Implementation of Build-heap algorithm is almost identical to the one written on page 111 in the book, which it describes there that the complexity is O(nlgn), because every call to heapify costs O(lgn), and it will get executed O(n) times.

**3) extract\_max(self)**

**TIME Complexity: O(lgn):**

In this function I will extract the max value, which is the root.

Afterwards I will have to determine which node should replace the root, and swap it after the extraction has occurred.

The extraction process take's O(1), and the Heapify afterwards O(lgn).

So in total the whole function will require O(lgn) time complexity.

**4) extract\_min(self)**

**TIME Complexity: O(lgn):**

In this function I will extract the max value, which is the one of the root child's (unless there is only 1 node)

Afterwards I will have to determine which node should replace the root, and swap it after the extraction has occurred.

The extraction process take's O(1), and the Heapify afterwards O(lgn).

So in total the whole function will require O(lgn) time complexity.

**5) Insert(self,key)**

**TIME Complexity: O(lgn):**

I will append the user key which will be added to the heap, I will increase the length by 1, than I will use the bubbleUp algorithm in order to rearrange the heap.

The append and increase commands are O(1), when the bubbleUp algorithm is O(lgn), so in total it the function take O(lgn) to perform.

**6)** **remove(self,i)**

**TIME Complexity: O(lgn):**

i will swap the index that the user want to remove from the heap with the last node, I will decrease the heap length by 1, and heapify it afterwards.

The swap and decrease commands will cost O(1) time complexity, when the heapify will cost O(lgn) time complexity.

In total the function will take O(lgn) to perform.

**d) FUNCTIONS and Subfunctions:**

Here I will elaborate which function use sub-function to help them perform their task.

1) **Build-Heap** : heapify, getLength

2) **heapify:** minHeapify, maxHeapify

3) **maxHeapify:** hasChildren,hasGrandChild,maxChildAndGrandChild,isGrandChild,swap, maxChild.

4) **minHeapify:** hasChildren,hasGrandChild,minChildAndGrandChild,isGrandChild,swap, SmallestChild.

5) **insert:** bubbleUp

6) **heapsort:** maxChild,swap,maxGrandChildren,heapify,insert.

7) **bubbleUp:** hasParent,swap,maxBubbleUp,minBubbleUp,getParent.

8)**maxBubbleUp:** hasGrandParent,getGrandParent,swap.

9)**minBubbleUp:** hasGrandParent,getGrandParent,swap.

10)**extract\_max:**maxChild,swap,heapify,maxGrandChildren.

11)**extract\_min:** hasChildren,smallestChild,swap,heapify,getLeftChild.

12)**remove:** swap, heapify

13)**hasLeftChild**

14)**hasRightChild**

15)**getLeftChild**

16)**getRightChild**

17)**hasChildren**

18)**hasParent:** getParent

19)**getParent**

20)**hasGrandParent:** getParent

21) **maxGrandChildren:** hasChildren,getLeftChild,maxChild,getRightChild,hasRightChild.

22)**minGrandChildren:** hasChildren,getLeftChild,smallestChild,getRightChild,hasRightChild.

23)**maxChildAndGrandChild:** maxChild,maxGrandChildren

24)**minChildAndGrandChild:** smallestChild,minGrandChildren

25)**smallestChild:** getLeftChild,getRightChild

26)**maxChild:** getLeftChild,getRightChild

27)**hasGrandChild:** hasChildren

28)**isGrandChild:** getLeftChild,getRightChild

29) **getLength**

30)**keyboardBuildHeap**: insert

31)**fileBuildHeap:** insert

e) **Time complexity for each helping function:**

I have supplied details about the sub-functions and the way I have implemented them in the python code, here I will include each function time complexity.

* **swap :** O(1)
* **getLength:** O(1)
* **isGrandChild :** O(1)
* **hasGrandChild:** O(1)
* **maxChild :** O(1)
* **smallestChild :** O(1)
* **minChildAndGrandChild :** O(1)
* **maxChildAndGrandChild:** O(1)
* **minGrandChildren :** O(1)
* **maxGrandChildren :** O(1)
* **hasGrandParent:** O(1)
* **getGrandParent:** O(1)
* **getParent:** O(1)
* **hasParent :** O(1)
* **hasChildren:** O(1)
* **getRightChild:** O(1)
* **getLeftChild:** O(1)
* **hasRightChild:** O(1)
* **hasLeftChild:** O(1)
* **printHeap:** O(n^2)
* **minBubbleUp:** O(lgn)
* **maxBubbleUp:** O(lgn)
* **bubbleUp:** O(lgn)
* **minHeapify:** O(lgn)
* **maxHeapify:** O(lgn)