/* *
* Title : Heaps
* Author : Abdul Razak Daher Khatib
* ID : 21801340
* Section : 001
* Assignment : 3
* Description : pdf file
*/
----

1)

```
(a)         1     |       1       |       1       |       1       |       1
        4     5   |   4     5     |    4     6    |   4     6     |    4       7
      6 , 7       |  7 , 6        |   5 , 7       |  7 , 5        |  5 , 6

                          1       |       1       |       1
                      4      7    |   5     4     |   5     4
                    6 , 5         |  6 , 7        |  7 , 6
```

```
(b)                               45
                             /           \
                       30                      55
                    /        \              /     \
                20              40       50        60
              /   \           /
            10     25       35

     ----------------------------------------------------------------------------------
                                  45
                             /           \
                       30                      55
                    /        \                      \
                20              35                   60
                   \
                    25
```

2)

	Insert	ExtractMin
Unsorted Array	O(1)	O(n)
AVL Tree	O(log(n))	O(log(n))
Min Heap	O(log(n))	O(log(n))
Unsorted linked list	O(1)	O(n)
Sorted linked list	O(n)	O(n)

4)

(a) The expected time is O (n), since we might need to traverse the whole heap, adding n elements to the array. The average would depend on the keys already in the heap and the element entered. If the key was greater/less than half of the elements it will take

/* *
* Title : Heaps
* Author : Abdul Razak Daher Khatib
* ID : 21801340
* Section : 001
* Assignment : 3
* Description : pdf file
*/

O(n) time too, since we will add n/2 elements to the array resulting in O(n) complexity. So the expected average time would be O(n). Since we are adding to an array the time complexity cannot be better than this for the reason given above (might need to traverse the whole heap or half of it).

(b) In this structure we use two heaps, one that is a max heap the other is a min heap. We maintain a maximum difference of 1 between the size of the two heaps and keep the element bigger than the median in the min heap and the ones smaller than the media in the max heap. We add an element according to its value compared to the max heap and min hip which takes log (n) time then we check if the difference in size property is maintained if it is not we take the top (max or min) element of the larger heap then add it to the other. For instance if we had {1 , 2 ,3 } and {5 ,6 , 7, 8} where the median is 5 and we add the element (10) then the median would be ((5 + 6) /2 = 5.5), then to maintain the property we remove the min from min heap, i.e. 5, and add it to the max heap, resulting in {1, 2, 3, 5} and {6, 7, 8, 10}, and now we take ((max + min)/2) as median. The expected time for insertion is log (n). SInce we add an element to one of the heaps. So every time we add an element it takes log(n), then afterwards we check if the median is still between the max and min (difference in sizes is not larger than 1). If not we take the max or min and we add to the other heap, which also is log(n) complexity. So, the overall complexity is O(log(n)). For finding the median, since the median is either the min of the min heap or the max of the max heap (O(1) complexity). Or the average if these two, then we have O(1) for all cases for finding the median.