Programming Assignment #4

Nicolas Benavides Levin

UIN: 127008425

Question 1:

a) Time complexity : O(n). I used the function is_sorted() to check if the vector is sorted. If so, the function returns true, and false otherwise.

b) Time Complexity: O(n^2). I created a double for loop that performs multiple passes through the vector. If the key at index j is greater than the next key, then you perform a swap. This continues until the vector is fully sorted.

c) Time Complexity: O(n^2). I created a double for loop that saves each i as the minimum index value and performs multiple passes through the vector. If the key at index j is less than the key at index minimum, then the new minimum index is equal to index j. A swap is then performed.

d) For my hybrid sort if my a.size() was less than 16 I performed a bubble sort on vector a. Otherwise I performed 4 quicksorts after separating the vector a into 4 subarrays. I then merged them together using the given merge function. I had accounted for remainders of 1, 2, and 3, but I commented these out for they were not needed to pass the test case.

e)

Array Size	1000	10000	100000	1000000
heapSort	.0003863	.0030517	.0491043	.614846
mergeSort	.0002202	.0031122	.043091	.520636
hybridSort	.003352	.0019322	.0241729	.23544
quicksort	.0001142	.0017874	.0200048	.257435
bubblesort	.0066548	.749417	77.1082	7700.405
selectionSort	.0034894	.336414	34.3101	3428.20
shellsort	.0003083	.0043394	.0684591	.934523
insertionsort	.0020951	.225096	22.7307	2265.08

Question 2:

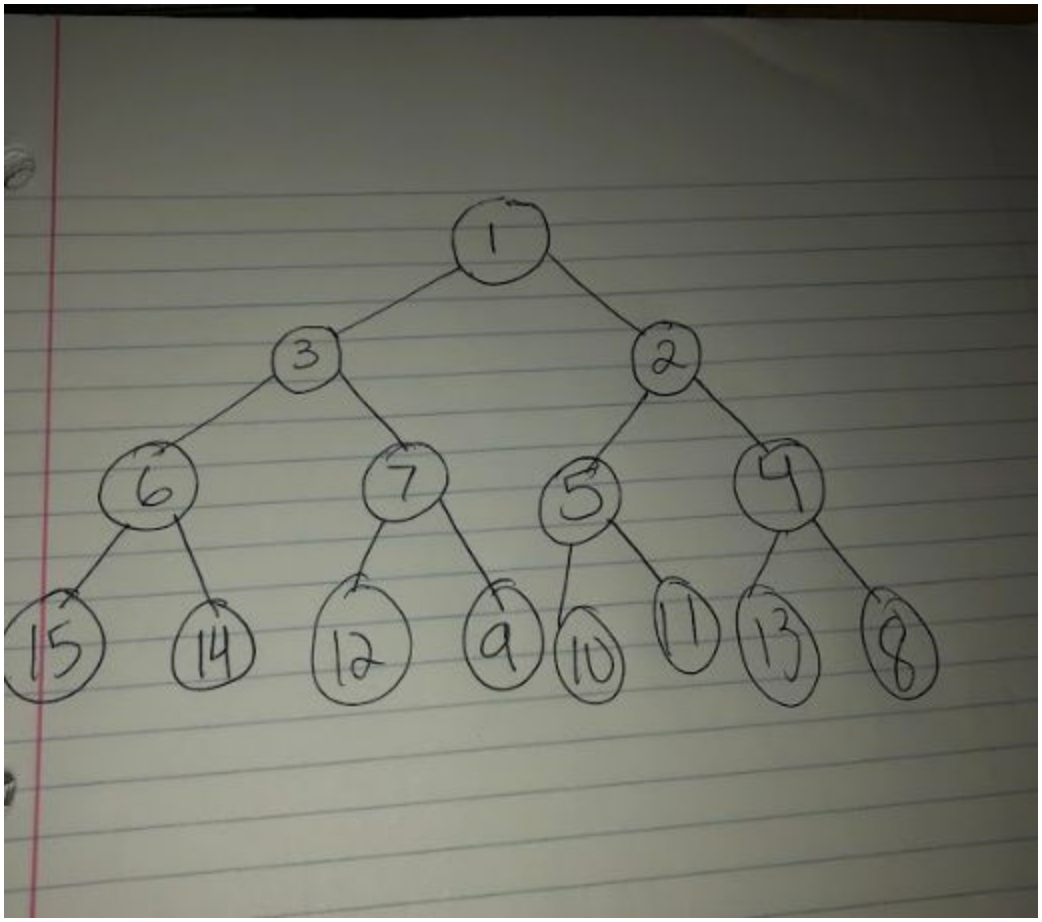
a) MinHeap.h

i) insertKey: For each insert, the heap size is increased and the value is inserted at the index heap_size-1. Then a while loop that checks for while the index !=0 and the arr[parent(index)] > arr[index]. The current index and parent(index) are then swapped, and the index is set to the parent(index).

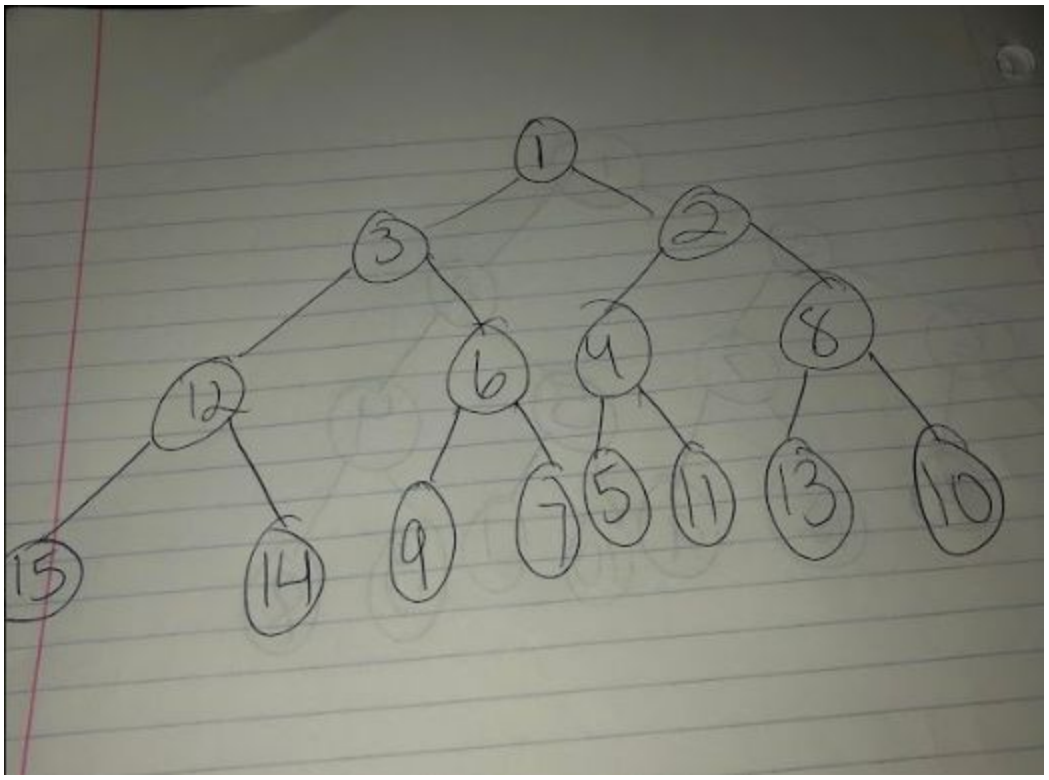
ii) extractMin: The function sets the root as arr[0]. Then arr[0] is set to the arr[heap_size-1] and heap size is reduced by one. Then index 0 is heapified and the root is returned.

iii) decreaseKey: the arr[i] is set to newVal. Then a while loop that checks for while the index !=0 and the arr[parent(index)] > arr[index]. The current index and parent(index) are then swapped, and the index is set to the parent(index).

iv) deleteKey: decreaseKey(i, INT_MIN(used to represent the most negative value)) is called to make the current key the smallest value. Then extractMin() is called to successfully delete the key.



b)



First Picture: Binary Heap
Second Picture: Linear Algorithm

c) Time Complexity is O(n+klogn). I created a new MinHeap with n capacity. I then created a for loop inserted each element of the arr[] into the MinHeap. I then created a for loop that iterates until k-1 and calls extractMin(). Once complete, I can then call getMin() and return this value for the kth smallest element.