Title: AVL Trees & Heap
Author: Atakan Akar
ID: 22203140
Section : 1
Homework : 2
Description : Analysis for Code

Question 5) If it was asked to get the least k elements from the heap instead of 5 elements than my approach would take more time. Since I used min heap insertion will take O(logk) time. And since the first question it was said that least 5 elements sorting it was asymptotically O(1) constant. However when it is said k than we must look to the elements in the first k level. Which makes 2^(k-1) elements. So sorting them with heap would take O(2^(k-1)log2^(k-1)). Which makes it not asymptotically O(1) so it won't be applicable for k and larger values.

Question 6)
Question 1 Analysis (least5.cpp): My solution is as explained. I put the elemets to a min heap in order to preserve the minimum 5 elements. Insertion into the heap is takes O(logn) time and also deletion of the top element also takes O(logn) time. In order to find the least 5 elements which are stored in the heap I stored another array which stores the least 5 elements and I got that elements from that array when it is need. I updated this five sized array when an element is added or removed. I sorted the first 5 level elements. I only compared the elements that are int the first 5 levels of the min heap. This is sufficient since in a min heap every root elements is smaller than its left and right child a least 5 elements can bot be located after the 5 levels. I sorted the elements with heap sort, although heap sort is O(nlogn) since I only considered the top 5 level elements (31 number of elements), so it takes constant time O(1) to retrieve the least 5 elements. I learned from the question that min heap gives benefit when to store the minimum elements when it is needed.

Question 2 Analysis (game.cpp): My solution is as explained. First the numbers that are in the input file are read and the card values are inserted into the array for both players. I used the binary search tree since when the binary search tree's height is balanced, the insertion and deletion operation have a time complexity of O(logn). In order to create a height balanced binary search tree first I put the items into the array and then sorted the arrays with heap sort in which I also used max heap. Heap sort has time complexity of O(nlogn) since inserting an element into a heap take logn time and I inserted the elements into the heap one by one which takes nlogn and then put the elements into the array which will lead to a sorted array nlogn 2nlogn general but O(nlogn). To create the a balanced binary search tree first I sorted the arrays. Then I used this algorithm to provide a balanced insertion order. I took the mid element as the first element to be inserted. Therefore, the left of the subtree will contain equal elements as the right subtree of the binary search tree. For the left part of the array, I also take the mid indexed element to be inserted after the root element. By recursively applying this algorithm for each sub-part of the array, the insertion order of the elements into the binary search tree becomes suitable for a balanced tree. This algorithm also takes O(n) time. Then I inserted into the binary search tree for both sequences for both players. Bobo starts the game, and both players play optimally, so Bobo must choose the greatest card from his

own hand and the greatest card that is smaller than his chosen card. If he finds it, he will win the turn. If he cannot find it, then Holosko gets the point for that turn. After Bobo's turn it time for Holosko to choose the best cards. He will use the same algorithm. This goes until they finish the cards in their hands. A balanced binary search tree here is beneficial since the retrieval operation is O(logn) and also deletion. If every player has n cards in its hands than since this operation will be done 2n times the time complexity will be 2nlogn but asymptotically O(nlogn). Also since the binary search trees height never exceeds O(logn) after the deletions it does not effects the time complexity. I learned from this question that binary search trees are beneficial in reducing the time complexity (int retravel or deletion operations) when they are balanced. Moreover, since the insertion order changes the height of a binary search tree an algorithm is required for it to be balanced.

Question 3 Analysis (prefixsubarray.cpp): My solution is as explained. The question ask for minimum L such that the subarray [1,L] for both A and B will contain minimum M elements that A is greater than B. In order find the minimum L we can make binary search in the array. If the sub array [1,L] satisfies the condition than we must reduce the L value with binary search algorithm if it does not satisfy than we must increment it since the subarray with greater size might contain satisfy it. The binary search algorithm takes O(logn). When we find a L with binary search then we must check whether the selected subarray satisfied that condition. Where I used a min heap for array A and max heap for B since I need to take the largest M elements for A and M minimum elements for B. So I took the first M elements to the both heaps between [1:L]. Then I looked to the element M+1 for if the A[M+1] is greater than the top element in the min heap of the A than I can remove the top and insert that element here min heap give me the advantage of find the maximum M elements. For array B if the M+1 element is less than the top element in the heap for B than I removed the top and inserted that element into the heap for B.Here the min heap give me advantage of finding the minimum elements of B between [1:L]. The size of heaps are equal to M so the insertion operation is O(logM) and deletion is also O(logM). Since we are not looking only the M+1 element after the first insertion to the all heaps but we are looking the all elements between [1:L] it takes O(nlogM) time. After finding the maximum elements for A and minimum elements for B between [1,L] we put them into a arrays in ascending order. The values are compared if all the elements of heap A greater than B s elements than the condition is satisfied this putting elements into a array takes O(MlogM) time but since it is less than O(nlogM).Because O(logn(nlogM + MlogM) = O(nlognlogM)) the time complexity is O(nlognlogM). If the condition is not satisfied we are incrementing L according to binary search operation on array which give rise to logn the logn in the result comes from here. I used min heap and max heap for my solution. I learned that their time complexity of insertion and deleting is O(logn) for an heap of size n. Min heap also give benefit when I am tracking the maximum object and max heap when I am tracking the minimum objects since they hold the minimum and maximum numbers in their storage respectively.

Question 4 Analysis (subarray.cpp): My solution is as explained. I stored the elements into a array for A and B. Then I start from the first index 1 I go to right of the array. I put the elements of A to a minheap for A and a max heap for B. Right pointer goes to the right. While it is iterating if it detects a element of A that is greater than the minimum elements of the heap A than it switches the elements. Also for heap B if it finds a element that is less then the maximum elements for the heap B it also switches. If it finds a good subarray than it stop

iteration for right pointer since our aim is to find the minimum length. Then left pointer is increased by one and the same continues for finding the minimum length for good array. Since there happenes n^2 iterations and deleting and putting inserting elements into the heap is O(logn) complexity total solution is O(n^2logn) time complexity. Also the heaps size for A and B are M and K respectively so if the heaps are not in their full size even if the top element of the heap A is greater than the top element of the heap B it does not says that subarray is good.