1.Binary search tree

Method To Insert a Value: If the tree is empty then create a new node that stores the input value, and make this the root of the tree. Otherwise compare the input value to the value stored at the root of this tree.

• If the input value is less than the value at the root, recursively insert the input value into the left subtree of the root.

• If the input value is greater than the value at the root, recursively insert the input value into the right subtree of the root.

• Finally, if the input value is equal to the value at the root, throw an exception and do not change the tree at all.

How To Delete a Value at a Node with Two Children: Let x be the node where the value to be deleted is located. Find the node y storing the smallest value in the right subtree of x by starting with the right child of x, and repeatedly moving down from a node to its left child until a node, y, with no left child has been found. The value at y can now be written into x (replacing the value to be deleted) and the right child of y can be promoted to replace y in the tree. (This will be a null node if y was a leaf.)

删除的时候情况就是1.直接删除 2.子节点替代 3. 右边分支最小值替代

2.red-black tree

height(x) is the number of nodes in a path from x down to some leaf — including the leaf but not including x, and black-height(x) is the number of black nodes in the same path from x — including the leaf but not including x, once again.

Since every node in this path is either red or black, but not both, it follows that height(x) = black-height(x) + r. where r is the number of red nodes on this path — not including x or the black leaf. Now, the children of every red node must both be black. every red node in this path has a black node immediately below it on the path. r ≤ black-height(x).

Relationship between Depth and Size of a Red-Black Tree: The depth of a red-black tree is always linear in the logarithm of its size. Colour Given to x if y was red: In this case, the colour of x is not changed.

Colour Given to x if y was black: If x was originally red then its colour is changed to red-black. Otherwise the colour of x was originally black and its colour is changed to double-black. // : This (temporarily) introduces two new “colours” for nodes in the tree. All red-black and double-black nodes must be eliminated before the resulting tree is, once again, a red-lack tree.// : A red-black node, or a double-black node that is at the root, can simply be replaced by a black node in order to restore the red-black properties. // A series of recolourings and rotations can be used to move double-black nodes closer to the root until either a red-black vertex results from the operation, or one of the simpler cases mentioned above results.

3.hash table

1.每次insertion 的时候看看里面已经有没有元素，如果有I+1来计算， i是里面的元素个数-1 h(x, i) = x + i 2 mod 10。for every integer i such that 1 ≤ i ≤ 9 — so that quadratic probing is being used。 As the above solution indicated, locations in the has table can be filled up using the special value DELETED (or “DEL”) — and this can extend the cost of all operations, because these locations must be examined along the locations storing elements of the set being represented during searches. Since h(x, 0), h(x, 1), h(x, 2), . . . , h(x, TABLESIZE - 1) is a permutation  of 0, 1, 2, …, TABLESIZE-1 (as stated in the question). Then in the worst case scenario, there are i-1 other values in the hash table already, and inserting the ith element would, therefore, require at most i probes. Therefore, the total number of probes in the worst case scenario is at most: O^2

4.binary search 给定一个升序序列 A = <A\_1, A\_2, ... , A\_n>A=<A1​,A2​,...,An​>找到这个序列里是否存在一个数 q

If high < low then a NoSuchElementException is thrown. Otherwise the value. mid = ⌊(low + high)/2⌋

This is always an integer such that low ≤ mid ≤ high. k is then compared to A[mid]. If k < A[mid] then the algorithm is called recursively; low is not changed, but high is replaced by mid − 1. The output generated (or exception thrown) by this recursive call is returned. // If k = A[mid] then mid is returned as output. // If k > A[mid] then the algorithm is called recursively: low is replaced by mid+ 1 but high is not changed. The output generated (or exception thrown) by this recursive call is returned. // mid is first set to have value ⌊(0 + 7)/2⌋ = 3, so k is first compared to A[3] = 9. • Since k = 30 > 9 the algorithm is then called recursively with inputs low = 4 and high = 7. mid is now set to have value ⌊(4 + 7)/2⌋ = 5, so k is next compared to A[5] = 25. • Since k = 30 > 25 the algorithm is called recursively with inputs low = 6 and high = 7. min is now set to have value ⌊(6 + 7)/2⌋ = 6, so k is next compared to A[6] = 30. • Since k = 30, the key has been found and no other comparisons are made. // Binary Search should be used, because the number of steps used by Binary Search is only logarithmic in the length of the input array, in the worst case, while the number of steps used by Linear Search is linear in the length of the input array in the worst case — and log2 n ∈ o(n). // Low and high are integers and are subtracted, therefore, f is an integer valued function. The value of this function either drops from 2k + 1 to k, for a non-negative integer k (so that it drops by (2k+1)−k = k+1 ≥ 1, Or it drops from 2k to either k or k − 1 for a positive integer k, so that its value drops by at least 2k − k = k ≥ 1 in both cases it drops by at least 1. If f(low, high) ≤ 0, it means that high – low ≤ 0, so high ≤ low Also low ≤ high (precondition) From above we can say low = high and in section “a” we showed that the algorithm terminates in this case. Therefore, the function terminates and does not call it self again when f(low, high) ≤ 0.

5.quick sort

Solution: The number of steps used by this algorithm to sort an input array with length n is in Θ(n 2 ), in the worst case. // Two modifications to the algorithm, can be applied together, to significantly improve the worst-case performance of this algorithm: • As described in Tutorial Exercise #19, the partitioning process can be changed so that all copies of the pivot element are clustered together and do not need to be further processed. This improves the algorithm’s performance when array entries are not guaranteed to be unique. • As mentioned in class, a deterministic linear-time to choose the median element in a subarray can be used to choose the partition element — in order to guarantee that when Quick Sort if recursively called, the subarray to be recursively sorted always has at most one-half the length of the original subarray. If both of these changed then the number of steps used in the worst case is decreased to Θ(n log2 n). // Arrays are fixed-length and therefore should not be used, at all, because eventually there will be no room to store new elements in the array, once the heap size has reached the array length.

An ArrayList should not be used, because some of the add operations require the capacity of the ArrayList to be increased the underlying array is replaced, and the cost of the operation will be linear in the size of the heap instead of only logarithmic in it.

// Since the MergeSort algorithm is already shown to be correct (see L16\_merge\_sort.pdf slide #35). All we need to do is to show that the new changes are covered by the same proof. Note that the definition of the “Merging problem” and the proof of correctness make no assumptions about the values of n\_1 and n\_2 except that they are both positive integers, so nothing about this algorithm or the proof of its correctness must be changed. Also note that nothing needs to be changed about the proof of correctness of the MergeSort algorithm when n\_1=n-1 instead of ⌈n/2⌉ and n\_2=1 instead of ⌊n/2⌋ because both new values are smaller or equal to n-1 (reduce n by at least 1), which allow us to use the same IH to establish the correctness of the recursive call.

6.heap sort

The problem is that heap order may now have been violated, because the value at the roof may now be strictly less than either, or both, of the children of this vertex. There is always at most one vertex whose value might be strictly less than either, or both, of its children. In order to correct this, if heap order is still violated, then the value at the “problem vertex” is exchanged with the value at the child whose value is larger (breaking ties arbitrarily). The “problem vertex” now becomes the child whose value was exchanged. This process continues until it is confirmed that the value at the “problem vertex” is greater than or equal to the values at any of its children — possibly because the “problem vertex” is a leaf, and there are no children at all. The number of steps used by the algorithm that fixes this problem, in this way, is in Θ(log2 n) if n is the current size of the heap. 加元素就是

加到最后然后开始往上比。删除最大值就是去掉根节点，然后根最后一个值网下比。If the array A has positive length n then heap-size is set to be 1 and, for 1 ≤ i ≤ n − 1, the entry in position i is inserted into the Maxheap. This reorders the array entries without changing them and turns the array into an array-based representation of a Maxheap with heap-size n. The deleteMin operation is then called n − 1 times: For 1 ≤ i ≤ n − 1, the element returned by the i th of these operations is written into position n − i of the array — which is now available, and the position where this element belongs. At the end of this the entries have been reordered, and the array is now sorted in non-decreasing order. // Consider an execution of this algorithm that begins with the precondition for the “Heapify” problem satisfied, so that an array (or ArrayList) A, storing values of some ordered type T, has been given as input. The while loop is eventually reached and executed. If the execution never terminates at all then neither does the algorithm, as needed to establish “partial correctness” in this case. On the other hand, if the execution of the loop terminates then the loop test must havefailed, so that i ≤ 0. However, it follows by part (c) of the loop invariant that i ≥ 0, so i = 0 when the execution of the loop, and the algorithm, ends.Since the root of the binary tree represented by A corresponds to index 0, part (d) of the loop invariant (with j = i = 0) implies the postcondition for the “Heapify” problem, as needed to establish partial correctness in this case too.

7.graph

1. 用 array和linked list来表达链接和重量或者二维数组。 2. For an adjacency-list representation it is necessary to traverse the list of neighbours of one of the vertices given as input. In this worst case the number of steps used is linear in the maximal degree of any vertex — in Θ(|V |), in the worst case. For an adjacency-matrix representation it is only necessary to examine one entry in the two-dimensional array used to represent the graph, so that Θ(1) steps are used in the worst case。// a minimum-cost spanning tree 村里通电 // The number of steps used, in the worst case, is in Θ((|V | + |E|) log2 |V |). // O n2 storage locations are needed.n2+1 if matrix used to store information about edges. // NoSuchElementException should (probably) be thrown if the input is not a pair of integers i and j such that 0≤i , j≤n-1 where n = |V|.// Otherwise, if A is the matrix used to store information about edges, then true should be returned if A[i, j] = 1, and false should be returned otherwise. // Once again, a NoSuchElementException should (probably) be thrown if the inputs are not a pair of integers i and j such that 0≤i , j≤n-1 where n = |V|. Another exception — possibly called an EdgeFoundException should be thrown if A[i, j] = 1 already. In the only remaining case A[i, j] and A[j, i] should both set to be 1. This operation can also be performed using a constant number of steps in the worst case. // If an array is used then it is necessary to make a copy of the matrix, with one more row and column each filled with 0’s. Θ(n^2) steps will be required in this case, if the graph originally had n vertices.if an ArrayList is used instead of an array then it is possible to use the add operation to extend rows, and to add a new one. It can be shown that the amortized cost will be smaller (indeed, an amortized cost linear in n can be established) but number of steps used in the worst case will still be in Θ(n^2). // If the input is not an integer between 0 and n - 1 (inclusive), where n=|V|, then a NoSuchElementException should be thrown. Otherwise the entries in row i should be checked: For 0≤j≤n - 1,  j should be listed as a neighbour if and only if A[i, j] = 1. Θ(n) steps are used in the worst case. // Assuming the uniform cost criterion and, that vertices are named 0, 1, 2, . . . n where n = |V| ..Θ(n+m) storage locations are used for this representation. One is needed to store n, n are used to represent the one-dimensional array with indices 0, 1, 2, . . . , n − 1 and storing the heads of linked lists, and (including space for references to nodes) at most another 2m locations are needed to store the linked lists of neighbours of nodes. Therefore, this representation needs n+2m+1 storage locations, which is in Θ(n+m)

.