Süleyman Gölbol

You have 30 minutes

1. (10 pts.) Assume you have 400 elements to store in a hash table and you would like to keep the expected number of comparisons less than three for finding an item. What would be smallest table size,

a) If you use open addressing with linear probing?

$3>C=1/2(1+1/(1-L)) \rightarrow L=400/T<4/5 \rightarrow T>500 \rightarrow 501$

b) If you use chaining?

$3>C=1+L/2 \rightarrow L=400/T<4 \rightarrow T>100 \rightarrow 101$

2. (10 pts.) Explain why quadratic probing reduces collision probability and number of probes over linear probing. Give an example.

önemli

In Linear probing, we may have primary clustering since when a collision happens the value is tried to put closer places. While in quadratic probing, during probes further places from the original position is tried so, primary clustering do not happen. Thus, an empty space is found with smaller number of probes.

See the example in the book Figure 7.5 and 7.6.

You have 30 minutes

3. (20 pts.) Give the number of comparisons performed while sorting [3, 4, 7, 6, 9] using following sorting algorithms. Give details of your work.

a) Insertion Sort : 5

Insert 4, compare 4-3. Insert 7, compare 7-4. Insert 6, compare 6-7 and 6-4, Insert 9 compare 9-7

a) Shell sort with gap values 7, 3, 1 : 7

Gap 7: no comp. Gap 3: Insert 6, 6-3. Insert 9, 9-4. Gap 1: same as Insertion sort.

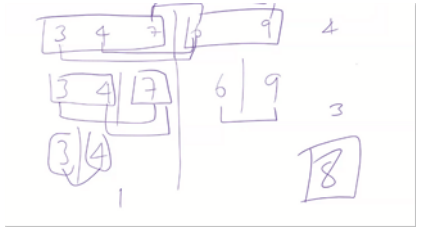
a) Quicksort using first element as the pivot : 6+5+4=15

Pivot 3: Left to Right: 3-4, Right to Left: 3-9, 3-6, 3-7, 3-4, 3-3. Pivot 4 LR: 4-7, RL: 4-9, 4-6, 4-7, 4-4,

Pivot 7: LR: 7-6, 7-9 RL: 7-9, 7-6. Pivot 6 no comp. Pivot 9 no comp.

a) Mergesort (the number of elements in the left half is equal to ceiling of n/2) : 9 tavan = 5

Merge 3x4: 3-4. Merge 3,4x7: 3-7, 4-7. Merge 6x9: 6-9. Merge 3,4,7x6,9: 3-6, 4-6, 7-6, 7-9



You have 40 minutes

4. (30pts.) Write a method **`findPred`** in the class **`BinarySearchTree`** to find predecessor of a value passed as a parameter in the binary search tree. Note that the predecessor is the largest element which is smaller than the given value. The value passed as a parameter may not be available in the binary search tree. Your function should return the predecessor or null reference if the value is not bigger than the smallest element in the binary search tree. Analyze the worst-case and best-case performance of your method and give examples for both cases.

5. (30 pts.) Write a method **`update`** in the class **`PriorityQueue`** which uses array based binary heap structure. The method takes an array index to locate the element and the new priority value as parameters. It should update the priority value of the element and perform required operations to maintain heap order property. Analyze the worst-case and best-case performance of your method and give examples for both cases.

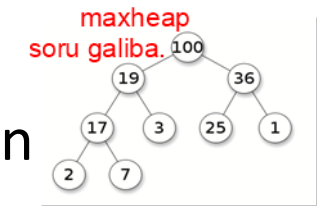
```
public E findPred(E item){
  if (root==null) return null;
  return findPred(item, root);
}
public E findPred(E item, Node<E> root){
  if (root.data.compareTo(item)>=0){
    if (root.left==null) return null;
    return findPred(item, root.left);
  }
  if (root.right==null) return root;
  E pred = findPred(item, root.right);
  if (pred==null)||(root.data.compraTo(pred)>0)
    return root
  return pred;
}
```

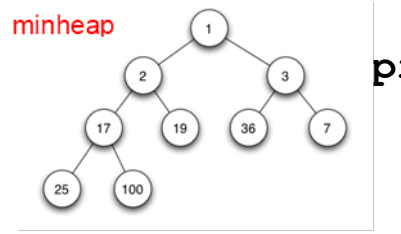
Question 4

Best case: Theta(1), when the root has no left child, and it is greater than the item.

Worst case: Theta(n), when the tree is linear, and the item is equal to the value at the only leaf.

Question






```
private void update(int index, E newVal){
  E oldVal=theData.get(index);
  theData.set(index,newVal);
  if (compare(oldVal,newVal)>0)
    down(index);
  elseif (compare(oldVal,newVal)<0)
    up(index);
}
private up(int child){
  int parent = (child - 1) / 2;
  while (parent >= 0 &&
            compare(theData.get(parent),
                theData.get(child)) > 0) {
     swap(parent, child);
     child = parent;
     parent = (child - 1) / 2;
  }
```

```
private down(int parent){
  while (true) {
     int leftChild = 2 * parent + 1;
     if (leftChild >= theData.size())
       break;
     int rightChild = leftChild + 1;
     int minChild = leftChild;
     if (rightChild < theData.size() &&
            compare(theData.get(leftChild),
              theData.get(rightChild)) > 0)
        minChild = rightChild;
     if (compare(theData.get(parent),
            theData.get(minChild)) > 0) {
        swap(parent, minChild);
        parent = minChild;
     } else
         break;
  }
}
```

Best case: Theta(1), when the newVal is smaller than oldVal but it is greater than the parent value. The node does not move at all.

Worst case: Theta(log n), when the smallest value at the root is updated with a value larger than all the values in the heap. It has to move until a leaf node.

Grading

1. a) and b) Calculation: 3

   Result : 2

   500 : -1

   100 : -1

2. Reason: 6

   not mention clustering: -2

   Example: 4

   no clear performance difference: -2

3. For each sorting method:

   Details: 3

   Number: 2

4. BST: 5

   Helper method: 3

   Predecessor at ancestor: 5

   Predecessor at descendant: 5

   Recursion: 4

   Best case + example: 2 + 2

   Worst case + example: 2 + 2

5. Update: 4

   Percolate down : 10

   Percolate up : 8

   Best case + example: 2 + 2

   Worst case + example: 2 + 2

   Build from scratch 10