Letao Chen
260686394
9:26 PM 2017-03-16

// Comp 251 //

// Q3 //

```
given arbitrary A (an int array)

implement set of active lists
     of which we will add A[i] to

3 cases:
1. A[i] smallest compared to all end candidates
     do: create new list (length 1)
2. A[i] largest compared to all end candidates
     do: clone largest active list
          extend with A[i]
3. otherwise (A[i] value between)
     do: find list with largest end element (smaller than A[i])
          clone
          extend with A[i]
          discard all lists with same length as new list

condition maintained:
end element of smaller lists
     always smaller than end element of larger lists

-- psuedo code --

     longestIncSubseq(int seq[])

          // track predecessors of elements of each subseq
          int pred[] = new int[seq.length]

          // track ends of each subseq
          int incSubseqEnds[] = new int[seq.length + 1]

          // length of longest subseq
          int length = 0

          for i from 0 to seq.length - 1
```

```
            // do binary search
            int low = 1
            int high = length

            while low <= high
                 int middle = Ceil((low + high) / 2)

                 if(seq[incSubseqEnds[middle]] < seq[i])
                      low = middle + 1
                 else
                      high = middle - 1

            int pos = low

            // update pred for longest increasing seq
            pred[i] = incSubseqEnds[pos - 1]

            // replace / append
            incSubseqEnds[pos] = i;

            // update length of longest subseq
            if(pos > length)
                 length = pos

       // backtrack to get LIS
       int LIS[] = new int[length]

       int k = incSubseqEnds[length]

       for i from length - 1 to 0
            LIS[i] = seq[k]
            k = pred[k]

       return LIS

-- end code --
```

```
end values A[i] of subsequences
     given by length
     are in increasing order

therefore, binary search time: O(log n)
     loop runs n times
     gives O(n log n)
```

// Q4 //

Question 4.

lets use edit scores from class

matches → +1

mismatch → -1

IN/DEL → -1

1. construct Table

	-	A	A	C	T
-	0	-1	-2	-3	-4
G	-1	-1	-2	-3	-4
A	-2	0	0	-1	-2
T	-3	-1	-1	-1	0

- first row + column easy since no ~~top~~/left
  → just insert and -1 each time
- now just go thru the rest and apply the 3 cases
  → pick the best result (labelled with red arrows)

2. trace back

- use graph above to trace back

	-	A	A	C	T
-	0	-1	-2	-3	-4
G	-1	-1	-2	-3	-4
A	-2	0	0	-1	-2
T	-3	-1	-1	-1	0

A A C T
G A - T

horizontal arrow (gap)

→ optimal sequence alignment

Using alternative method, max score = 0