Assignment 1

Rahul Motan

**1)**        (Applying L-Hopital rule)

               (Simplifying and applying L-Hopital Rule)

Applying the same principle to another set of functions,  we get,

Taking Log of numerator and denominator we get,

Now Applying L- Hopital Rule, we get

Applying limits we get

Hence

Now we compare

     (Apply L-Hopital Rule we get)

,  Simplifying & Applying the limits, we get

Hence,

**2a)**

From the given relation we can infer,

Also,

therefore from the above relations we can say that,

hence,  is TRUE

**2b)**

if this hold true then,

Taking Log on both sides we get,

,

Dividing by  we get,

So we can say this relationship is TRUE

**3. a)**

The problem here is of searching but not the exact match rather finding the number of elements lesser than the given B.M.I, additionally the exact match could be present,  so we will have to check for that as well.

In the problem statement its mentioned that at the end of every year the list is prepared for the next year. Assuming the list is sorted according to the B.M.I, we can therefor apply the algorithm of Binary-Search as my friend Polly suggested.

The catch here is that we never return “-1” when the given B.M.I we are searching for is not found, instead we return the lower bound of the deduced sub-array we are searching in at that point. This could very well be the case when the given B.M.I is not present in the sub-array or list.

**3. b)**

**Assumptions:**

1. The given sub-array & the parent array are in the non-increasing order of the B.M.I and the B.M.I in the list are of the same age as of the patient.(Using merge-sort we can sort the given B.M.I List in ) But this will be done once every year at the end of the year.
2. Let the list of patients who visited in the last year be , where
3. Let the given sub-array be , where
4. Let the given B.M.I be

sorted in non-increasing order.

***compute-Percentile(, ){***

  ***search-BMI()***

***return***

***search-BMI(){***

***if***

***if***

***if***

***return*** ***search-BMI()***

***else***

***return***

***else if***

***return*** ***search-BMI()***

***else***

***return*** ***search-BMI()***

***return***

**3. c)**We prove the correctness of our algorithm by induction on the size of sub-array

## Base case:

             if , then it will return  as

             if ,

                   it will call ***search-BMI()***

this call will return

                   it will call ***search-BMI()***

this call will return l

**Assumption**:  ***search-BMI***works correctly for any sub-array of size

**Inductive Step:**

Let be of size

                  there are 3 cases:

 1.   , it returns  assuming there’s only single instance of it in the B.M.I List.

 2.   , it will call ***search-BMI()***

                    size of sub-array now is

    a) if

        by our assumption, this returns the correct answer as our algorithm works for size

    b) if

            size of sub-array is

        by our assumption, this returns the correct answer as our algorithm works for size

          3.  , it will call ***search-BMI()***

the size of the sub-array now is

                a) if

                  by  our assumption this returns the correct answer.

                b) if

                by  our assumption this returns the correct answer .

   There can be a **special case** where the B.M.I of two or more patients in the list could be of the same value.

   For e.g.

    So in such a case we have similar scenarios as above but when  , we will call ***search-BMI()***

     and the size of the sub-array becomes

   from above we can see there will be two cases and both of them hold true.

Complexity of the above algorithm is . I have assumed the above algorithm would be run on a sorted list. The sorting would occur once and will not be done every time we call the above algorithm.