Homework Assignment 4

CS 430 Introduction to Algorithms

Name : Karthik Mahesh

CWID : A20383027

1.Solution:

In a 2-3 tree, there will be ceiling of n/3 to n/2 parent-child links when there are n items because in 2-3 tree, a parent can have 2 or 3 children.

After inserting n values, there will be ceiling of (n-1)/3 to (n-1)/2 splits.
Upper bound will be (n-1)/2 and lower bound will be (n-1)/3.

2.Solution:

We know that in chained Hashing (where collisions are resolved by adding keys into a linked list), the expected time for an unsuccessful search is "O(1 + **α**)" and for a successful search is also "O(1 + **α**)" where **α** is the load factor(as mentioned in the notes 3c).

We also know that time complexity for searching in red black tree is O(log n) where n is the total number of elements in the tree.

So when linked list is replaced by a red-black tree in chained hashing, the expected time for an unsuccessful search is "O(1 + log**α**)" and for a successful search is "O(1 + log**α**)" where **α** is the load factor.

3.Solution:

Yes, the hashing function will generate m different locations because when a location is repeated at the jth step, it will double hash and find the empty location for the key.

4.Solution:

Step (i): Characterize the structure of a coin-change solution.
• Define C[v] to be the minimum number of coins we need to make change for bill of value v.
• If we knew that an optimal solution for the problem of making change for bill v used a coin of denomination di ,
we would have: C[v] = 1 + C[v − di].

Step (ii): Recursively define the value of an optimal solution.

$$C[V] = \begin{cases} \infty & \text{if } v<0, \\ 0 & \text{if } v=0, \\ 1 + \min\limits_{1<=i<=k} \{C[v-di] & \text{if } v>1 \} \end{cases}$$

Step (iii): Compute values in a bottom-up fashion.
Avoid examining C[v] for v < 0 by ensuring that v ≥ di before looking up C[v − di].

```
COMPUTECHANGE(n,d, k )
  C[0] = 0
  for v = 1 to n do
       C[v] = ∞
       for i = 1 to k do
              if v ≥ di and 1 + C[v − di] < C[v] then
      C[v] = 1 + C[v − di]
   return c
```

Step (iv): Construct an optimal solution.
We use an additional array denom[1..n], where denom[v] is the denomination of a coin used in an optimal solution to the problem of making change for value v.

```
COMPUTE-CHANGE(n,d, k )
   C[0] = 0
   for v = 1 to n do
         C[v] = ∞
          for i = 1 to k do
               if v ≥ di and 1 + C[v − di] < C[v] then
        C[v] = 1 + C[v − di]
        denom[v] = di
    return c
```

Step (v): Print optimal solution.

```
PRINT-COINS(denom, v )
   if v > 0
     PRINT-COINS(denom, v − denom[j])
     print denom[v]
```

References used:

Analysis and design of algorithms textbook
http://ace.cs.ohiou.edu