Cuckoo Hashing Pseudocode

```
main()
   char s[255] (1)
   char null_st[] (1)
   size_t i, len; (1)
   bool placed; (1)

for(i=0; i< tablesize; i++) {  [2 * ( n – 0 + 1)]
            strcpy(t[i][0], null_st);
            strcpy(t[i][1], null_st);
   }

openfile(file) (1)
   if ( file != NULL )
   {
while ( fgets ( s, 255, file ) != NULL ) {  [4 * ( n – 0 + 1)]
 len = strlen(s);
 s[len-2]='\0';

 placed = place_in_hash_tables(s);

  if (!placed) {
output( "Placement has failed")
 return 1;
}
 }
 fclose ( file ); }
   else {
            perror ( filename );
   }
   return 0;
}

bool place_in_hash_tables (char *s) {
   bool placed; (1)
   size_t pos; (1)
   int index; (1)
   char temp_s[255], temp[255]; (1)
   strcpy(temp_s, s); (1)

int counter = 0; (1)

   index = 0; (1)
   placed = false; (1)

   pos = f(temp_s, index);
   while( (!placed) && (counter < 2*tablesize) ) { [4 * ( n – 0 + 1)]
            if (strcmp(t[pos][index], "") == 0 ) {
                     string <temp_s> there
                       strcpy(t[pos][index], temp_s);
                       placed = true;
                       return placed;
            } else {

                     <temp_s> there
                       strcpy(temp, t[pos][index]);
                       strcpy(t[pos][index], temp_s);

                       strcpy(temp_s, temp);

                       if (index==0)
                                index = 1;
                       else
                                index = 0;
                       pos = f(temp_s, index);

                       counter ++;
            }
   }
   return placed;
};

size_t f(char *s, size_t index) {
size_t po, len; (1)
   int i, val, temp; (1)
   po = 1; (1)

   len = strlen(s); (1)

   if (index == 0) {
            val = s[0]; (1)

            if (len == 1){
                     val = val % tablesize; (1)
                     if (val < 0)
                       val += tablesize; (1)
            return val;
        }

        for (i = 1; i < len; i++) { [3* ( n – 0 + 1)]
            temp = s[i];
            po = pow(37, i);

            val += temp * po;
        }
        val = val % tablesize; (1)
        if (val < 0)
            val += tablesize; (1)
        return val;
    }
    else {
        val = s[0]; (1)

        if (len == 1){
            val = val % tablesize; (1)
            if (val < 0) (1)
              val += tablesize; (1)

            return val;
        }

        for (i = 1; i < len; i++) { [3* ( n – 0 + 1)]
            temp = s[len - i - 1];
            po = pow(37, i);

            val += temp * po;
        }

        val = val % tablesize; (1)
        if (val < 0)
            val += tablesize; (1)
        return val;
    }
}
```

Worst case time complexity = O(n)