```
bool place_in_hash_tables(char *s) {

      //copy the string to first temp-s
      strcpy(temp_s, s);

      // use a counter to detect loops
      int counter = 0;

      // start with table T1
      index = 0;

      placed = false;

      //find position of
      pos = f(temp_s, index);


      while ((!placed) && (counter < 2 * tablesize)) {
                                    1   +  1
            if (strcmp(t[pos][index], "") == 0) {
                  // hash table is available so store the string <temp_s> there
                  strcpy(t[pos][index], temp_s);

//end while loop
            }
            else {
                  // hash table is not available so

                  // Store string in t[pos][index] in temp
                  strcpy(temp, t[pos][index]);

                  //Add the new string temp_s to t[pos][index]
                  strcpy(t[pos][index], temp_s);

                  // NOW temp_s CONTAINING THE EVICTED STRING
                  strcpy(temp_s, temp);

                  // IN THE OTHER TABLE index TO INDICATE THE OTHER TABLE
                  if (counter % 2 == 0)
                        index = 1;
                  else
                        index = 0;

                  // WRITE THE CODE TO CALCULATE IN pos THE HASH VALUE FOR temp_s
                  pos = f(temp_s, index);

                  if (strcmp(t[pos][index], "") == 0) {
                        strcpy(t[pos][index], temp_s);
                        placed = true;
                  }

                  counter++;
            }
      }
      return placed;
};
```

- `bool place_in_hash_tables(char *s) {` — $O(strlen)$
- `strcpy(temp_s, s);` — $O(strlen)$
- `int counter = 0;` — 1
- `index = 0;` — 1
- `placed = false;` — 1
- `pos = f(temp_s, index);` — $O(strlen)$
- `while ((!placed) && (counter < 2 * tablesize)) {` — 4
- `if (strcmp(t[pos][index], "") == 0) {` — 1 + 1
- `strcpy(t[pos][index], temp_s);` — $O(strlen)$ } $O(strlen)$
- `strcpy(temp, t[pos][index]);` — $O(strlen)$
- `strcpy(t[pos][index], temp_s);` — $O(strlen)$
- `strcpy(temp_s, temp);` — $O(strlen)$
- $50O(strlen)+7$
- `if (counter % 2 == 0) index = 1; else index = 0;` — 1 + 1 } $\max(1,1)+2 = 3$
- `pos = f(temp_s, index);` — $O(strlen)$
- `if (strcmp(t[pos][index], "") == 0) { strcpy(t[pos][index], temp_s); placed = true; }` } $\max(O(strlen)+1, 0)+3 = O(strlen)+4$
- `return placed;` — $\max(O(strlen), 50O(strlen)+7) = 50(strlen)+7$

$Rt = 50(strlen)+7+3+20(strlen) = 70(strlen)+10$

$7strlen + 10 \leq 17strlen \quad \forall \; strlen \geq 1$ ✓

- O(length of string)

```
// compute the hash functions
size_t f(char *s, size_t index) {

    //add to function 1
    if (index == 0) {
        value = Σ_{i=0}^{len-1} (S[i] % 19) * (37 % 19);
        value = value % 19;
        return val;
    }
    //add to function 2
    else {
        value = Σ_{i=0}^{len-1} (S[len - i - 1] % 19) * (37 % 19);
        value = value % 19;
        return val;
    }
}
```

① $1+1+1+1+1$ — $\{lenStr * 5$

$1+(1+1+1)+1+1+1$ — $\{lenstr * 7$

$\max(5len+2, 7len+2) = 7len+2$

Running time of f(*string) hash function is O(strlen)

$7len+2 \leq C\,len \quad \forall\ len \geq len_0$

Let $C=9$ then $7len+2 \leq 9len \Rightarrow 2 \leq 2len \Rightarrow 1 \leq len$

$7len+2 \leq 9len \quad \forall\ len \geq 1$

```
while (not end of file)
    //Insert into hash table
    placed = place_In_hash_tables(s);
}
```

— O(strlen) + 1

len n = number of strings in file

then $n * (strlen + 1) = nstrlen + n$

$n * strlen + n \leq 2n\,strlen \quad \forall\ n \geq n_0$

let n=1 $n \leq n\,strlen \quad \forall\ n \geq n_0$

$1 \leq strlen \quad \forall\ n \geq 1$

therefore

Rt of Cuckoohash Table is $O(n * strlen)$

n = number of strings

strlen = length of i^{th} string