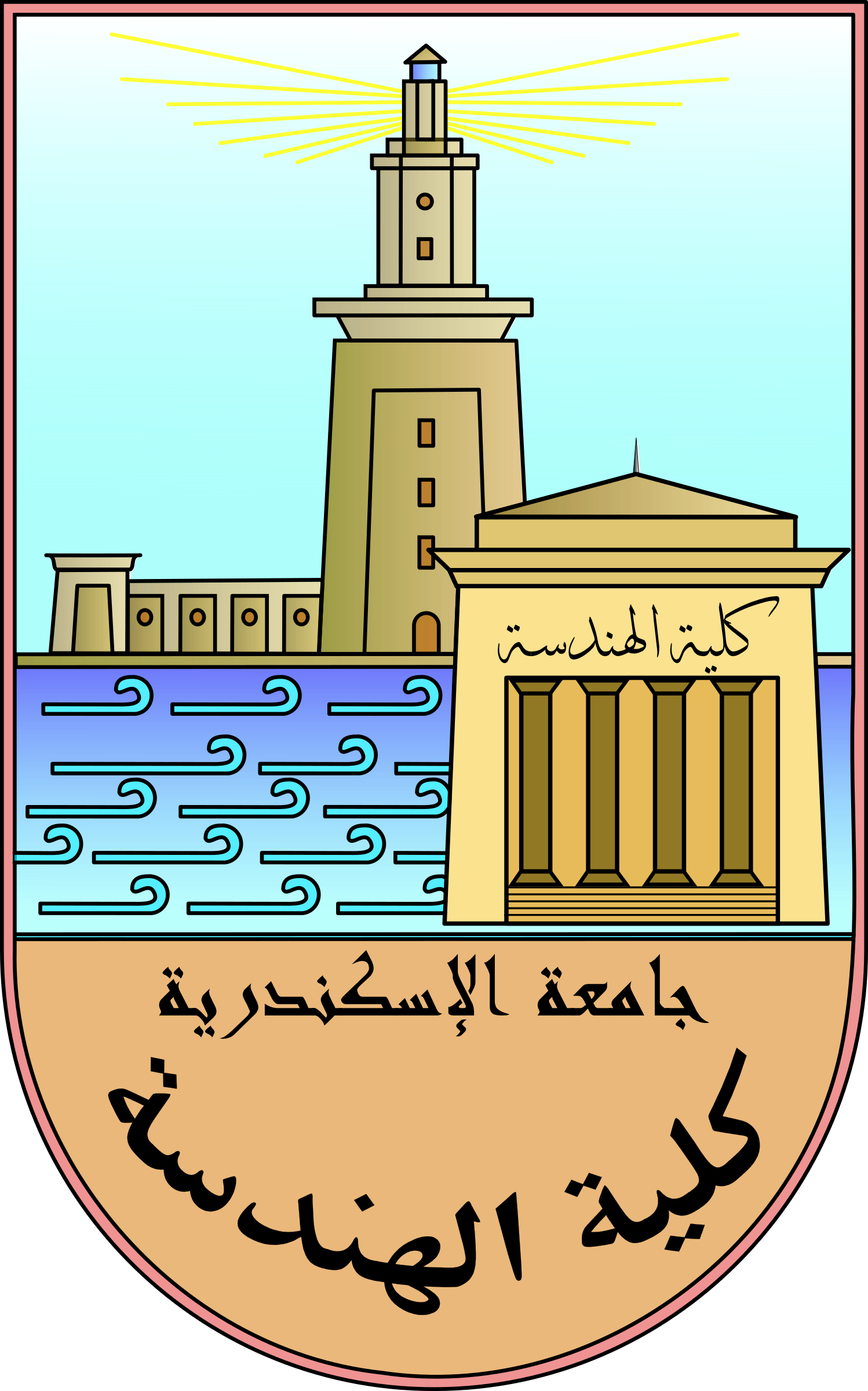
Programming Paradigms:



Phase I

Karim Tarek Ibrahem

AbdelRahman Alaa-Eldeen Idrees

Ahmed Samir

Abdullah Taman

‘

1. Features used from each paradigm.
2. Time Analysis

* **Insert:**
* **In O(N) Hash Table:**

**It took O(1) in case of no collision, but if collision happens which has a probability of half because of the similarity of this problem with the birthday-paradox problem, it takes an average of O(L(i)) which L(i) is the number of elements that share a position I in the main hash table.**

* **Delete:**
* **In O(N) Hash-Table**

**Always O(1).**

* **Search:**
* **In O(N) Hash-Table**

**Same as delete, always O(1).**

* **Batch insert:**
* **In O(N) Hash-Table**

**Building an O(n) hash-table takes an order of polynomial of degree (O(lg(n)\*lg(n)\*n)) in a static dictionary, or we could say it’s just O(n) with high probability. We are dealing with a dynamic dictionary in the lab, meaning that we could look at the batch insert as a dynamic build of a perfect O(n) hash table. We shall have again order O(lg(n)\*lg(n)\*n\*γ). Where γ a number indicated the upper bound of how many times for a slot we needed to rehash its inner hash table while inserting.**

* **Batch delete:**
* **In O(N) Hash-Table**

**O(N).**

1. Comparison between the two implementations according to the average time

We used a word generator to generate us one million word, then we manipulated them ten times and got the following results.

|  |  |
| --- | --- |
| **O(N) Space** | **O(N^2) Space** |
| - | - |