|  |
| --- |
| PriorityQueue |
| A = [a[0]… a[n-1]] |
| Invariants:   * 0 <A.getPriorityQueueSize() * a[n] PriorityQueue * a[0].getPriority() a[i].getPriority 0 i < A.getPriorityQueueSize() * a[⌊(i-1)/2⌋] is father of: a[i] * a[i/2+1] is left son of: a[i] * a[i/2+2] is right son of: a[i] * The length of each branch must be equal to the height of the tree minus 1 |
| Operations:   * PriorityQueue 🡪 PriorityQueue * Insert PriorityQueue x element 🡪 Boolean * Maximum PriorityQueue 🡪 element * Minimum PriorityQueue 🡪 element * Extract-Max PriorityQueue 🡪 element * Extract-Min PriorityQueue 🡪 element * Increase-Key PriorityQueue, element, value of increment 🡪 boolean * Decrease-Key PriorityQueue, element, value of increment 🡪 boolean |

|  |
| --- |
| Stack |
| Stack= [Sn… S1] |
| Invariants=  0 n Size(Stack)  n top = Sn  [Si ∈ S ∧ Sn  ∈ S]  →   SI  ≠ Sn  Sn is always get out first than  S1 |
| Operations:   * Stack 🡪 Stack * Push Stack x Element 🡪 Stack * Pop Stack 🡪 Stack * Top Stack 🡪 Element * isEmpty Stack 🡪 boolean |

|  |
| --- |
| HashTable |
| HT = [(h1,..,h1xn),....,(hn,..,hnxn) ] |
| Invariant =  Element keys always have to be different |
| Operations:   * HashTable 🡪HashTable * toString HT 🡪String * insert HT x Element x Key 🡪int * search HT x Key 🡪Value * delete HT x Key 🡪void * hashFunction HT x Key x i 🡪int * containsKey HT x Key 🡪boolean * containsValue HT x value 🡪boolean * occupedSize HT 🡪int * size HT 🡪int |

|  |
| --- |
| Queue |
| Queue= [Qn… Q1] |
| Invariants=  [Qi ∈ Queue ∧ Qn  ∈ Queue]  →   QI  ≠ Qn  Qn is always get out first than  Q1 |
| Operations:   * Queue 🡪 Queue * Peek Queue 🡪 Value * Poll Queue 🡪 Value * offer Queue x Element 🡪 Element * is Empty Queue 🡪boolean * size Queue 🡪int |