**HEAP SORT**

|  |  |
| --- | --- |
| **def heap\_sort(self):** | **HEAPSORT(A,n)** |
| self.build\_max\_heap()  for i in range(self.lenght-1, 0, -1):  self.v[0], self.v[i] = self.v[i], self.v[0]  self.heapsize -= 1  self.max\_heapify(0) | BUILD-MAX-HEAP(A,n)  for I = n downto 2  Exchange A[1] with A[i]  A.heap-size = A.heap-size -1  MAX-HEAPIFY(A,1) |
| **def build\_max\_heap(self):** | **BUILD-MAX-HEAP (A, n)** |
| self.heapsize =self.lenght  for i in range((self.lenght-1)//2, -1, -1):  self.max\_heapify(i) | A.heap-size = n  for i = n/2 downto 1  MAX-HEAPIFY(A,i) |
| def max\_heapify(self, i) | **MAX-HEAPIFY(A,i)** |
| l = left(i); r=right(i)  if l < self.heapsize and self.v[l] > self.v[i]:  largest = l  else: largest = i  if r < self.heapsize and self.v[r] > self.v[largest]:  largest = r  if largest != i:  self.v[largest], self.v[i] = self.v[i], self.v[largest]  self.max\_heapify(largest)  **def parent(i):**  return (i-1)//2  **def left(i):**  return 2\*i +1  **def right(i):**  return 2\*i +1 | l = LEFT(i); r = RIGHT(i)  if ≤ A.heap-size and A[l] > A[i]  largest = l  else largest = i  if r ≤ A.heap-size and A[r] > A[largest]  largest = r  if largest ≠ i  exchange A[i] with A[larges]  MAX-HEAPIFY(A,largest)  **PARENT(i)**  return i/2  **LEFT(i)**  return 2i  **RIGHT(i)**  return 2i + 1 |