# Assignment 4

Starting with the array arr, the current index i, and the number of items n make up the first three parameters of the heapify helper function, which is defined as part of the Heapsort algorithm. Heapify checks if the subtree with index i has the max-heap attribute. To do this, it checks the present node's size against that of its left and right offspring, and if it finds that the current node is too small, it replaces it with the biggest of the three. After the swap, the max-heap structure is maintained by calling heapify on the affected subtree in a recursive fashion.  
  
Before sorting the input array, heapsort applies heapify on every non-leaf node, beginning with the last internal node and ending at the root, to convert it into a max-heap. The effective heap size is reduced by one after construction by exchanging the root (the biggest element) with the final member of the array. After that, the heapify function is used to get the heap structure back from the root. Until the whole array is sorted, this method sorts each element sequentially.  
  
Regardless of the worst, average, or best case scenario, Heapsort runs with a time complexity of O(nlogn) due to the fact that each element sorting requires n items and each insert into the heap takes O(logn) time. This approach is perfect for scenarios where memory economy is paramount since it is an in-place algorithm and it requires just 𝑖(1 ) O(1) more space independent of the input array. Elements with equal keys may not keep their original order after sorting because Heapsort is not reliable. Regardless, when reliable sorting is not needed, Heapsort is a popular alternative due to its predictable speed and simplicity. By breaking down heap formation, element extraction, and re-heapification into its component parts, the method makes it easy to understand and implement heap-based sorting principles.

A screenshot of a computer program

Description automatically generated

Each job in the priority queue is represented by a job class, which has a distinct task\_id and a degree of priority. A binary heap, internally represented as an array or list, is used to construct the priority queue by the PriorityQueue class. Essential for maintaining a priority-based task order, it enables critical heap operations such as insert and extract max.  
  
In order to maintain the max-heap property, the \_heapify\_up function is invoked whenever a new job is added to the heap. Here, we check the new job's priority against that of its parent and switch them around if the kid is more important. As long as the max-heap structure is maintained, this procedure will continue until the new job is properly positioned.  
  
The most important job, usually at the bottom of the heap, is removed and returned using the extract max function. Following the removal of the root, the heap property is restored by triggering the \_heapify\_down function and transferring the final piece in the heap to the root location. The process of \_heapify\_down entails checking the root's children for priority and replacing it with the one with the highest priority if needed. Till the heap structure is correctly re-established, this procedure will continue.  
  
The temporal complexity of both inserting and extracting data is O(logn), since up to n log log swaps are required to preserve the heap property. This priority queue solution offers rapid indexing and manipulation, making it ideal for applications that need basic heap operations. Prioritizing jobs by their importance is guaranteed by using a max-heap. Because of its effectiveness and simplicity, this method makes the fundamental operation of a priority queue easy to learn and use.

A screenshot of a computer program

Description automatically generated