Heap Sort

By:

Dustin Smith

Heap sort is a comparison based sorting algorithm. With heap sort, we divide the array into sorted and unsorted regions. We then iteratively shrink the unsorted region by extracting the largest element. Heap sort is an in place algorithm so the auxiliary space is $\theta(1)$.

1. Build max heap; building the max heap takes $\theta(n)$ operations.
2. Swap the first element of the list with the final element. Decrease the range by one.
3. Sift the new first element to its appropriate index in the heap.
4. Return to step two.

The sifting operation is $\theta(\log_2 n)$ but n times so $\theta(n + n \cdot \log_2 n)$ time complexity. This is simply $\theta(n \cdot \log_2 n)$.

```
def heap_sort(arr: List[int]) -> None:
  if len(arr) == 1:
    return

  heapify(arr, len(arr))

  end = len(arr) - 1

  while end > 0:
    arr[end], arr[0] = a[0], arr[end]
    end -= 1
    sift_down(arr, 0, end)


def heapify(arr: List[int], len_arr: int) -> None:
  start = (len_arr - 2) // 2

  while start > 0:
    sift_down(arr, start, len_arr - 1)
    start -= 1


def sift_down(arr: List[int], start: int, end: int) -> None:
  root = start

  while (root * 2 + 1) <= end:
    child = root * 2 + 1
    swap = root

    if arr[swap] < arr[child]:
      swap = child
    if (child + 1) <= end and arr[swap] < arr[child + 1]:
      swap = child + 1
    if swap != root:
      arr[root], arr[swap] = arr[swap], arr[root]
      root = swap
    else:
      return
```