

Analysis

Part I

Basis

n - size of the heap, d - heap arity, h - height of the heap.

$d \geq 2$ is assumed. For $d = 1$ the heap is degenerated, and the operations will take $\Theta(n^2)$ at the worst case.

The amount of items in depth k is at most d^k . Therefore:

$$\begin{aligned} \sum_{k=1}^{h-1} d^k &\leq n \Rightarrow \frac{d^h - 1}{d - 1} \leq n \Rightarrow \frac{d^h - 1}{d} \leq n \Rightarrow d^h \leq nd + 1 \Rightarrow \\ h &\leq \log_d(nd + 1) \leq \log_d(2nd) = \log_d(2) + \log_d(n) + 1 \leq \log_d(n) + 2 \end{aligned}$$

Analysis of heapifyDown

It takes exactly $d - 1$ comparisons to find the minimum child of a node, and 1 comparison to determine if the current node should be swapped with its minimum child. Therefore d comparisons are done at each iteration, and there are at most k iterations for a node at height k . Therefore for a call on a node of height k , there are at most dk comparisons.

Analysis of arrayToHeap (implemented with heapify)

Let x be the number of leaves in the heap. Then the number of nodes of height 1 is $\lceil \frac{x}{d} \rceil \leq \frac{x}{d} + 1$. The number of nodes of height 2 is $\left\lceil \frac{\lceil \frac{x}{d} \rceil}{d} \right\rceil \leq \frac{\frac{x}{d} + 1}{d} + 1 = \frac{x}{d^2} + \frac{1}{d} + 1$.

Similarly the number of nodes of height 3 is not greater than $\frac{x}{d^3} + \frac{1}{d^2} + \frac{1}{d} + 1$. Also, obviously $x \leq n$. It's easy to see that the number of nodes of height k is not greater than $\frac{n}{d^k} + 2$. From this, and from the analysis of heapifyDown, , the amount of comparisons done by arrayToHeap is bounded by:

$$\begin{aligned} \text{Comparisons} &\leq \sum_{k=1}^h dk \left(\frac{n}{d^k} + 2 \right) \stackrel{*}{\leq} \sum_{k=1}^h dk \cdot 2 \frac{n}{d^k} = 2nd \sum_{k=1}^h \frac{k}{d^k} \leq 2nd \sum_{k=0}^{\infty} \frac{k}{d^k} \\ &= 2nd \frac{d}{(d-1)^2} = 2n \left(\frac{d}{(d-1)} \right)^2 \leq 8n = O(n) \end{aligned}$$

Where the second transition is true for large enough n .

Also, $\Omega(n)$ is an obvious lower bound for the comparisons made by heapify, as at least $n - 1$ comparisons are required to find a minimum item in an unordered list.

Therefore arrayToHeap executes $\Theta(n)$ comparisons.

Analysis of DHeapSort (n delete-min calls)

As each delete-min simply calls heapifyDown on the root, then from the analysis of heapifyDown, the number of comparisons is $O(dh) = O(d(\log_d(n) + 2)) = O(d \log_d(n)) = O(\frac{d}{\log(d)} \log(n))$. Then the series of n calls to delete-min are executed in $O(\frac{d}{\log(d)} n \log(n))$.

This bound is tight at the following example shows:

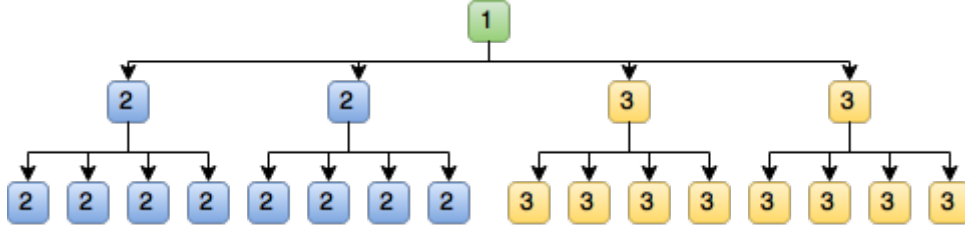
For simplicity, we'll assume the heap's last level is a full, and that d is even (the idea can be adapted to other cases).

We are given a heap where the root has key 1, and of it's d children, the left half have key 2, and the right half have the key 3. For every node that has key 2, all it's descendants have key 2, and for every node that has key 3, all it's descendants have key 3. See Figure 1 for example.

Given this scenario, the first half of the n delete-min calls will make most of the 3 nodes travel all the way from the root to the deepest level (to the left side, instead of the 2 nodes). Therefore, for $\Theta(n)$ nodes, $\Theta(d \log_d(n))$ comparisons will be made. Therefore the bound above is tight, $\Theta(\frac{d}{\log(d)} n \log(n))$ comparisons will be made at the worst case.

The entire operation will therefore make $\Theta(n) + \Theta(\frac{d}{\log(d)} n \log(n)) = \Theta(\frac{d}{\log(d)} n \log(n))$ comparisons at the worst case.

Figure 1: W.C. Heap Example



Measurements Explanation

Assuming the average case has a similar run-time complexity, the measurements can be explained:

As seen in the measurements, the amount of comparisons is linearithmic ($n \log(n)$ relation) in n , as expected.

The amount of comparisons appears to be lowest when $d = 3$. This can be explained by examining the term $\frac{d}{\log(d)}$. Looking at the function $f(x) = \frac{x}{\log(x)}$, we get $f'(x) = \frac{\log(x)-1}{\log^2(x)}$. Therefore $f'(e) = 0$, and it is clear that $x = e$ is a global minimum for $x \geq 1$. So in theory, an e -heap would have the lowest amount of comparisons. In practice 3 is the closest integer to e , so it yields the lowest amount of comparisons achievable using DHeapSort.

Part II

Analysis of heapifyUp

When heapifyUp is called on an item at depth k , then at most k comparisons and swaps are performed, as the item is only compared to its parent.

Analysis of n Insert, Decrease-Key calls

The maximum depth of an item in the heap is $h \leq \log_d(n) + 2 = O(\log_d(n))$, and each Insert performs 1 heapifyUp, therefore the amount of comparisons in n calls to Insert is bounded by $O(n \log_d(n)) = O(\frac{n \log(n)}{\log(d)})$

By the exact same reasoning, n calls to Decrease-Key are bounded by $O(n \log_d(n)) = O(\frac{n \log(n)}{\log(d)})$

The bounds are tight, as the following input shows:

n items are inserted with the keys: $2n+1, 2n, 2n-1, \dots, n+1$, and the Decrease-Key calls are done with $\delta = n+1$, in the order of insertion.

At the insertion phase, each item inserted is the minimum (at that stage), and therefore must be heapified-up from the bottom of the heap to the root. At the decrease phase, each item changes from the maximum to the minimum, and must be heapified-up from the bottom to the root.

In a heap of size n , more than $\frac{n}{2} = \Theta(n)$ items are leafs. Therefore $\Theta(n)$ of the operations, in both phases, will indeed cost $\Theta(n \log_d(n))$ comparisons, showing that the bound is tight.

Therefore the amount of comparisons of in the overall operation is at the worst case $\Theta(\frac{n \log(n)}{\log(d)})$.

Note: if $d \geq n$ then the worst case is $\Theta(n)$, because $\Omega(n)$ is an obvious lower bound.

Measurements Explanation

Assuming the average case has a similar run-time complexity, the measurements show that for $x = 100$, $x = 1000$, the number of comparisons indeed shrinks as d grows, as expected.

The lower-bound for the best case is $n - 1$ comparisons, in the case where no item changes its position (-1 because the root item never needs to be heapified-up). That bound is indeed achieved for the case where $x = 1$, because if a node's key is higher than its parent, then decreasing it by 1 will not make them swap. Because the heap is built by insertions and not via arrayToHeap, then if a node and its parent have the same key, then the parent must have been inserted earlier (because nodes are inserted to the bottom of the heap and are heapified-up). Therefore the parent's key will be decreased earlier, so there isn't a possibility of them swapping.