Fibonacci Heaps

Fibonacci is a mergeable heap with the advantage that it has very good amortized running times (only $\lg n$ for extract-min and delete) and is especially good when there are few extract-min (or max) and delete operations. The idea is to keep a root list that are all linked together. Each of these roots have their own tree. All nodes in the heap has, beside their pointers, some additional attributes, namely $x.degree$ and $x.mark$.
The potential method in amortized analysis represents the prepaid work that can be used to pay for future operations. The amortized cost of an operation $\hat{c}_i$ can be expressed as

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

We defined the potential function of a Fibonacci Heap H by $\Phi(H) = t(H) + 2m(H)$ where $t(H)$ is the number of nodes in the root list and $m(H)$ is the number of marked nodes.

We will focus on the amortized analysis of extract-min. Extract-min works by removing $H.min$ and moving all childs of the node to the root list and then we call the auxillary function `Consolidate`. `Consolidate` works by looping through the root list and if two roots have the same degree, the one with the highest key is inserted under the other and degrees are adjusted accordingly. This means that when it terminates, we have at most $D(n)$ roots, where $D(n)$ is the max degree of the nodes.
We now want to show the amortized cost is $\mathcal{O}(D(n))$ for extract-min. First we calculate the actual cost. We have that extract-min processes $\mathcal{O}(D(n))$ children (inserting them to root list). `Consolidate` also contributes $\mathcal{O}(D(n))$ when it creates an array of size $D(n)$. Now, when we call `Consolidate`, the root list is at most $t(H) + D(n) - 1$. Since we know the number of times we insert a node under another in `Consolidate` is at most $t(H)$ times, we get the work performed by `Consolidate` is at most proportional with $\mathcal{O}(D(n) + t(H))$, which is our actual cost.
Our potential before making extract-min is $t(h) + 2m(H)$. Our potential afterwards is $D(n) + 1 + 2m(H)$ since the number of roots afterwards is at most $D(n) + 1$ and the number of marked nodes has not changed, so we get the amortized cost, $\hat{c}_{em}$ is

$$\begin{aligned}
\hat{c}_{em} &= O(D(n) + t(h)) + (D(n) + 1 + 2m(H)) - (t(H) - 2m(H)) \\
&= O(D(n)) + O(t(H)) - t(H) \\
&= O(D(n))
\end{aligned}$$

Intuitively the cost of performing each link is paid for by the reduction in potential due to the link's reducing the number of roots by one. We now want to show that $D(n) \leq \mathcal{O}(\lg n)$.

We want to prove that for any node x in a Fibonacci heap with degree k, then $size(x) \geq F_{k+2} \geq \phi^k$ and $\phi = (1 + \sqrt{5})/2$.
Let s_k be the minimum size of the node x with degree k, so $s_k \leq size(x)$. Now if we consider a node z with degree k and $size(z) = s_k$ with children $y1, ..yk$ in order of when they were linked. We compute a lower bound for $size(x)$, so

$$\begin{aligned}
size(x) &\geq s_k \\
&\geq 2 + \sum_{i=2}^{k} s_{y_i.degree} \quad \text{(Counting 2 from } z \text{ itself and one for } y_1) \\
&\geq 2 + \sum_{i=2}^{k} s_{i-2} \qquad \text{(From Lemma 19.1)}
\end{aligned}$$

We now show by induction that $s_k \geq F_{k+2}$. Bases are $k = 0$ and $k = 1$, and we now assume that $k \geq 2$

and that $s_i \geq F_{i+2}$ for $i = 0, 1, .., k-1$. Now

$$\begin{aligned}
s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\
&\geq 2 + \sum_{i=2}^{k} F_i \\
&= 1 + \sum_{i=0}^{k} F_i \\
&= F_{k+2} && \text{(From Lemma 19.2)} \\
&\geq \phi^k && \text{(From Lemma 19.3)}
\end{aligned}$$

It now follows that in an n-node fibonacci heap, that $n \geq size(x) \geq \phi^k$ when we have a node x with degree k. If we take base-ϕ logarithms we get that $\lg_\phi n \geq k$. This means that the maximum degree $D(n)$ is $\mathcal{O}(\lg n)$.