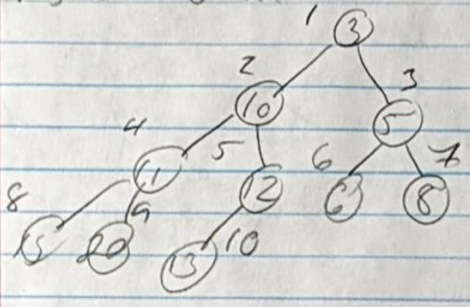
HW 6

Jeffrey Lansford

4.16 a) Example: 3, 10, 5, 1, 12, 6, 8, 15, 20, 13

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

At array position 4, we have 1.
The parent of 1 is $\lfloor 4/2 \rfloor$ which position 2 or 10. The two children are at $2i$ and $2i+1$ which is $2(4)$ and $2(4)+1$ or 8 and 9.
Position 8 and 9 or 15 and 20.



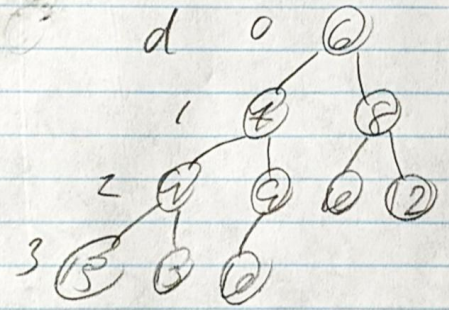
Lets say we select a node at position i and its parents are at $\lfloor i/2 \rfloor$ and i has 2 children at $2i$ and $2i+1$. If we select node i at position $2i$, then its parent is at $\lfloor 4/2 \rfloor$ or $\lfloor 2i+1/2 \rfloor$ which equals i. We can continue until the number $> n$, then that means that those nodes do not have children.

(b) a Parent Node at position j is $\left\lceil \frac{(j-1)}{d} \right\rceil$ and the children being

$\{(j-1)d+2, \ldots, \min(n, (j-1)d+d+1)\}$

with each depth describe as d^i with i being depth level. we store each level in the array with $d^i \leq j < d^{i+1}$ with i being the depth level j is at.

(c)



If we place a new element into level 0, worst case it must shift down through 3 levels. Placing new elements at lower levels reduces the number of times we need to shift down till we get to the level with leaf nodes where they do not require shift downs. We can represent the number of shifts down for each level with the above graph

$d=0 \quad 3 \times 2^0$
$1 \quad 2 \times 2^1$
$2 \quad 1 \times 2^2$
$3 \quad 0 \times 2^3$

$= \sum_{d=0}^{3} (3-d) \cdot 2^d$ total number of operations

$$\sum_{d=0}^{\log n} d \cdot 2^d = 2 \cdot \frac{(\log n + 1)2^{\log n}(2-1) - (2^{\log n + 1} - 1)}{(2-1)^2}$$

$$2 = (2^{\log n} \log n + 2^{\log n}(1) - (2^{\log n} \cdot 2^1 - 1))$$

$$2(2^{\log n} \log n + 2^{\log n} - 2^{\log n} - 2^{\log n} + 1)$$

$$2(2^{\log n} \log n - 2^{\log n} + 1)$$

$$2^{\log n + 1} \log n - 2^{\log n + 1} + 2$$

$$\log n \sum_{d=0}^{\log n} 2^d = \frac{2^{\log n + 1} - 1}{2 - 1} = (2^{\log n + 1} - 1) \log n$$

$$= (2^{\log n + 1} \log n - \log n)$$

$$(2^{\log n + 1} \log n - \log n) - 2^{\log n + 1} \log n + 2^{\log n + 1} - 2$$

$$-\log n + 2^{\log n + 1} - 2$$

$$2n - \log n - 2$$

$$O(2n - \log n - 2)$$

$O(n)$ thus the make heap procedure takes $O(n)$ time.

d.) We can adapt this algorithm to include d-ary heaps is to change minchild and bubbleup. They rely on the array position of a binary tree, so we have to change it to array position of d-ary.

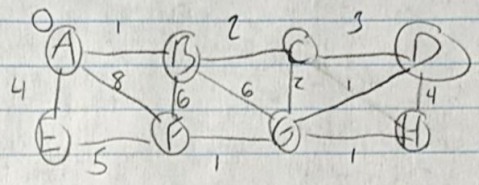
For bubbleup, we can change how it got the parent node to

$$\lceil (j-1)/d \rceil$$

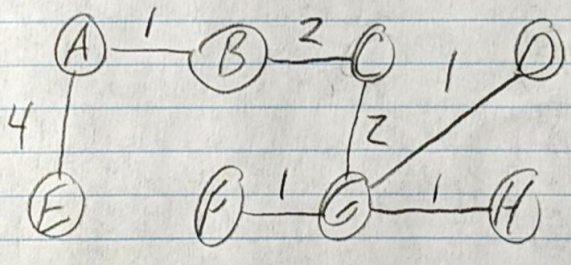
For minchild, it would have to go through each d child and return the minimum child. So you can have where it goes through each child key value d times

Jeffrey Lankford

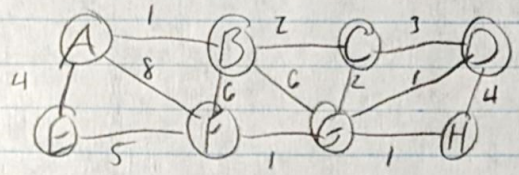
5.2 a)



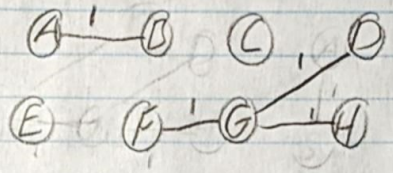
	A	B	C	D	E	F	G	H	Set
0	0	∞	∞	∞	∞	∞	∞	∞	{ }
1	/	1/A	∞	∞	4/A	8/A	∞	∞	{A}
2	/	/	2/B	∞	/	6/B	6/B	∞	{A, B}
3	/	/	/	3/C	/	/	2/C	∞	{A, B, C}
4	/	/	/	1/G	/	1/G	/	1/G	{A, B, C, G}
5	/	/	/	/	/	/	/	/	{A, B, C, G, D}
	/	/	/	/	/		/	/	



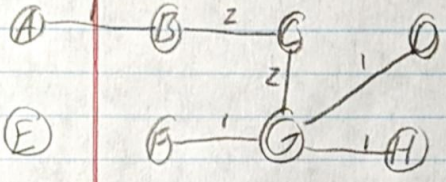
b.


A
B
C
D
E
F
G
H
1
2
3
8
6
6
2
1
4
5
1
1

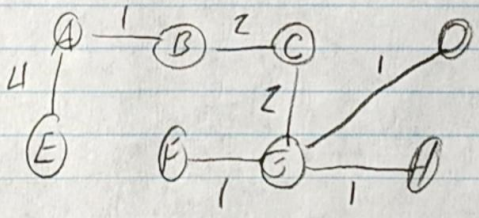
1

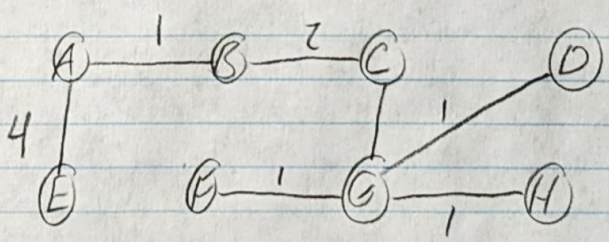

A
B
C
D
E
F
G
H
1
1
1
1

2


A
B
C
D
E
F
G
H
1
2
2
1
1
1

3


A
B
C
D
E
F
G
H
1
2
4
2
1
1
1


A
B
C
D
E
F
G
H
1
2
4
1
1
1

5.3 a) To answer the question, the edges that need to be remove are cycle edges for it to be still a connect graph. We can run DFS or BFS to find cycles, and remove them. Their runtime is $O(|V|+|E|)$

b.) To reduce the runtime, it is highly dependent on the structure of the graph. If we used it on a tree structure it would be $O(|V|)$ since the number of edges is equal $n-1$.