Computer Science 245

Final Reveiw

Do not turn in!

1. Give the $\Theta()$ running time for each of the following functions, in terms of the input parameter n:

   (a)
```
int f(int n) {
    int i;
    sum = 0;                   Theta(1)
    for (i=0; i<n; i=i+2)      executed n/2 times
       sum++;                    Theta(1)
    return sum;                Theta(1)
}
```
   $\Theta(n)$

   (b)
```
int g(int n) {
    int i;
    sum = 0;                   Theta(1)
    for (i=0; i<n; i=i+1)      exeucted n times
       sum += f(n);              Theta(n)
    return sum;                Theta(1)
}
```
   $\Theta(n^2)$

   (c)
```
int h(int n) {
    for (i=1; i<n; i=i*2)    Executed lg n times
       sum += f(n);            Theta(n)
    return sum;              Theta(1)
}
```
   $\Theta(n \lg n)$

2. For each of the following recursive functions, describe what the function computes, give the recurrence relation that describes the running time for the function, and then solve the recurrence relation.

   (a)
```
int recursive1(int n) {
    if (n <= 1)
       return 1;
    else
       return recursive1(n-2) +  recursive1(n-2);
}
```
   This function computes $2^{\lfloor n/2 \rfloor}$

   Recurrence relation:

$$
\begin{aligned}
T(0) &= c_1 \\
T(1) &= c_1 \\
T(n) &= 2T(n-2) + c_2
\end{aligned}
$$

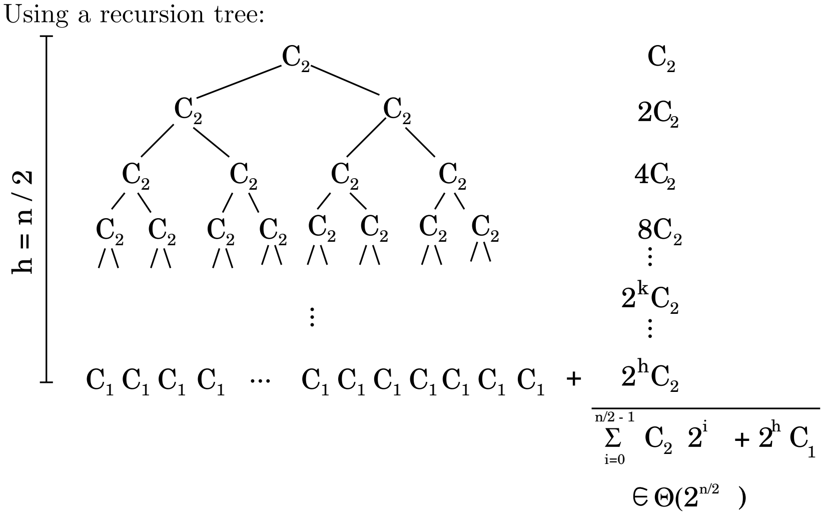
   Using the substitution method:

$$
\begin{aligned}
T(n) &= 2T(n-2) + c_2 \\
&= 2(2T(n-4) + c_2) + c_2 \\
&= 4T(n-4) + 3c_2 \\
&= 4(2T(n-6) + c_2) + 3c_2 \\
&= 8T(n-6) + 7c_2 \\
&= 8(2T(n-8) + c_2) + 7c_2 \\
&= 16T(n-8) + 15c_2 \\
&= 16(2T(n-10) + c_2) + 15c_2 \\
&= 32T(n-10) + 31c_2 \\
&= 32(2T(n-12) + c_2) + 31c_2 \\
&= 64T(n-12) + 63c_2 \\
&\ldots \\
&= 2^k T(n-2k) + (2^k - 1)c_2
\end{aligned}
$$

We set $n - 2k = 0$, or $k = n/2$ to get

$$
\begin{aligned}
T(n) &= 2^{n/2} T(n - 2n/2) + (2^{n/2} - 1)c_2 \\
&= 2^{n/2} T(0) + 2^{n/2} c_2 - c_2 \\
&= 2^{n/2} c_1 + 2^{n/2} c_2 - c_2 \\
&\in \Theta(2^{n/2})
\end{aligned}
$$

Using a recursion tree:



(b)
```
int recursive2(int n) {
   if (n <= 1)
      return 1;
   else
      return 2 * recursive2(n-2);
}
```

This function also computes $2^{\lfloor n/2 \rfloor}$ (though does it a bit more efficiently ...)

Reccurence relation:

$$\begin{aligned} T(0) &= c_1 \\ T(1) &= c_1 \\ T(n) &= T(n-2) + c_2 \end{aligned}$$

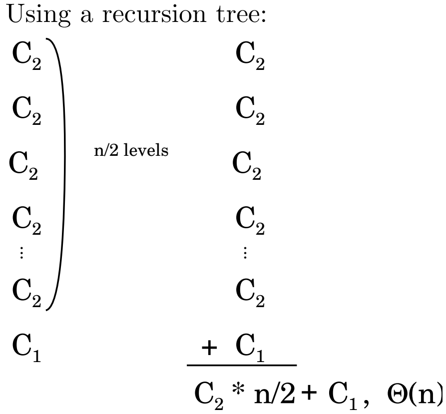
Using the substitution method:

$$\begin{aligned} T(n) &= T(n-2) + c_2 \\ &= (T(n-4) + c_2) + c_2 \\ &= T(n-4)2c_2 \\ &= (T(n-6) + c_2) + 2c_2 \\ &= T(n-6) + 3c_2 \\ &= (T(n-8) + c_2) + 3c_2 \\ &= T(n-8) + 4c_2 \\ &= (T(n-10) + c_2) + 4c_2 \\ &= T(n-10) + 5c_2 \\ &= (T(n-12) + c_2) + 5c_2 \\ &= T(n-12) + 6c_2 \\ &= (T(n-14) + c_2) + 6c_2 \\ &= T(n-14) + 7c_2 \\ &\ldots \\ &= T(n-2k) + kc_2 \end{aligned}$$

We set $n - 2k = 0$, or $k = n/2$ to get

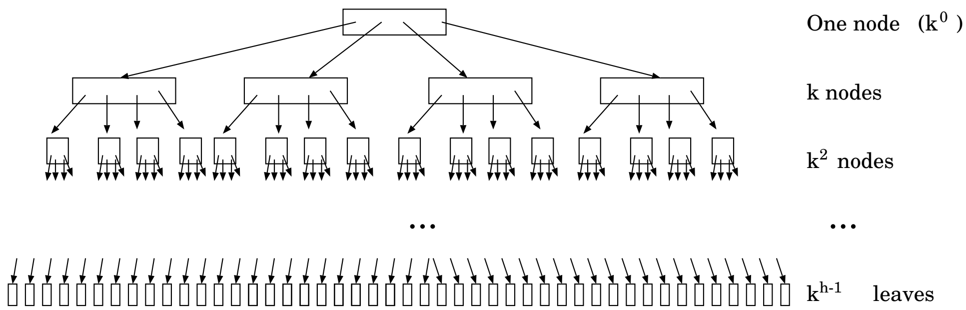
$$\begin{aligned} T(n) &= T(n-2(n/2)) + (n/2)c_2 \\ &= T(0) + (n/2)c_2 \\ &= c_1 + (n/2)c_2 \\ &\in \Theta(n) \end{aligned}$$

Using a recursion tree:



3. Consider a B-Tree with maximum degree k (that is, all interior nodes have $\lceil k/2 \rceil \ldots k$ children – a 2-3 tree is a B-Tree with maximim degree 3).

   (a) What is the largest number of keys that can be stored in a B-Tree of height h with maximum degree k? If the tree is completely full, then every internal node will have k children, and every node will have $k-1$ keys. Thus there will be 1 node at the root, k nodes at the second level of the tree, k^2 nodes at the next level, and so on – up to k^{h-1} leaves:



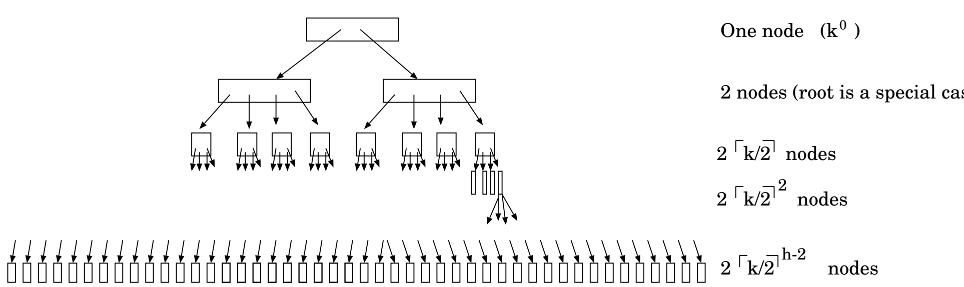
   So, the total number of nodes is:

$$
\begin{aligned}
0 + k + k^2 + k^3 + \ldots + k^{h-1} &= \sum_{i=0}^{h-1} k^i \\
&= \frac{k^{(h-1)+1} - 1}{k-1} \\
&= \frac{k^h - 1}{k-1}
\end{aligned}
$$

   Since each node has $k-1$ keys, the total number of keys in the entire tree is just (# of keys / node) * (# of nodes):

$$(k-1) * \frac{k^h - 1}{k-1} = k^h - 1$$

   (b) What is the smallest number of keys that can be stored in an B-Tree of height h with maximum degree k?
   We can use similar logic to the above, however note that the root can have as few as 2 children. Each interior node (other than the root) will have $\lceil k/2 \rceil$ children, and every node (other than the root) will have $\lceil k/2 \rceil - 1$ keys.



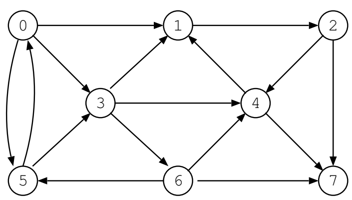
   The total number nodes is:

$$1 + 2 + 2 * \lceil k/2 \rceil + 2 * \lceil k/2 \rceil^2 + 2 * \lceil k/2 \rceil^3 \ldots 2 * \lceil k/2 \rceil^{h-1} \quad = \quad 1 + 2 * \sum_{i=0}^{h-2} \lceil k/2 \rceil^i$$

$$= \quad 1 + 2 * \frac{\lceil k/2 \rceil^{(h-2)+1} - 1}{\lceil k/2 \rceil - 1}$$

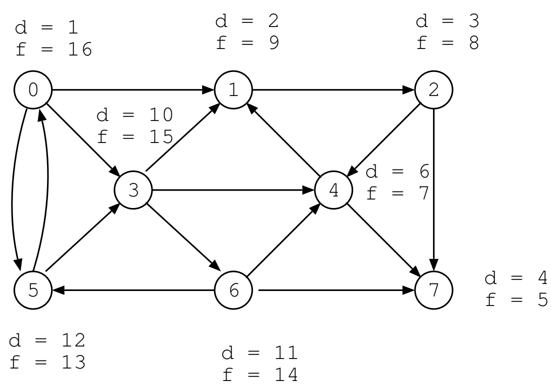
$$= \quad 1 + \frac{2\lceil k/2 \rceil^{h-1} - 2}{\lceil k/2 \rceil - 1}$$

4. Consider the following directed graph:

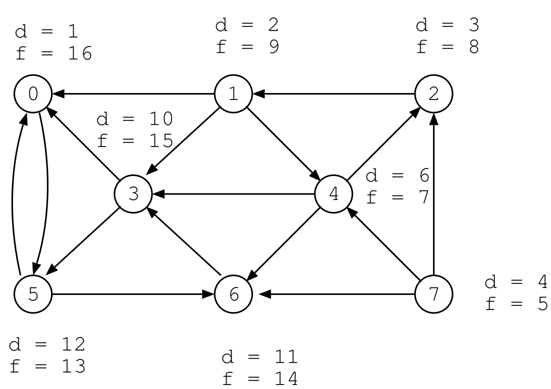


Run the connected component algorithm on this graph. Show all discovery and finish times, as well as the depth-first forrest for the final pass of the algorithm.

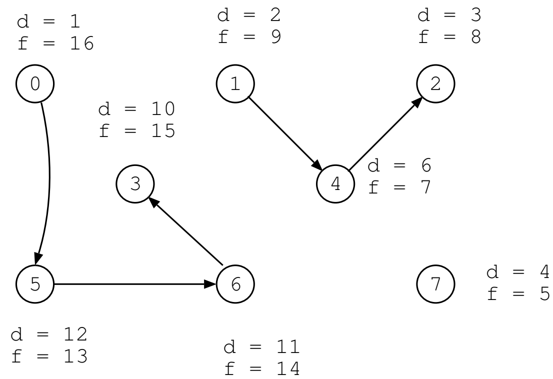
After computing discovery / finsh times (there is more than one valid set of discovery / finish times, of course):



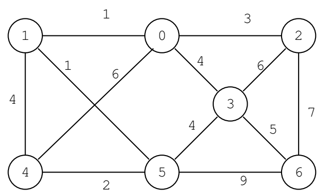
After taking transpose:



Final depth-first forrest:



5. Consider the following graph:



(a) Show the Vertex / Distance / Path table after Dijkstra's algorithm is run on this graph

Vertex	Known	Distance	Path
0	true	0	-1
1	true	1	0
2	true	3	0
3	true	4	0
4	true	4	5
5	true	2	1
6	true	9	3

(b) Show the Vertex / Distance / Path table after Prim's algorithm is run on th this graph

Vertex	Known	Distance	Path
0	true	0	-1
1	true	1	0
2	true	3	0
3	true	4	(0 or 5)
4	true	1	5
5	true	1	1
6	true	5	3

6. Look over all the visualizations for algorithms used in the class, be sure you know how all of them work.