Midterm

Data Structures and Algorithms, DatZ3168-EN
Wednesday, April 12, 2023

You must justify all your answers to get full credit

The problems can have somewhat different weights. The weights will be announced shortly.

1. For functions $f_1(n), f_2(n), f_3(n), f_4(n)$ defined below, give an asymptotic upper bound using Big-O notation. Your answer should be *a tight bound* from the following list (replacing the parameter k with appropriate value): $O(n^k)$, $O(2^n)$, $O(2^{n!})$, $O(\log^k n)$, $O(n^k \log n)$, $O(1)$, $O(n^n)$, $O(n!)$, $O(n^k \log\log n)$, $O(n^k \log n)$. $O(n^k \cdot \log n \cdot \log\log n)$ (A tight bound means that giving a very large answer such as $O(2^{n!})$ won't result in many points, if it can be improved. Your answer should be a function of n only, it must **not** contain extra parameters such as k.)

   (a) $f_1(n) = 5\log n \cdot \log\log n + 4\log(n^3)$,

   (b) $f_2(n) = n \cdot \log(n!)$,

   (c) $f_3(n) = 8^{\log_7 n} + 15n$,

   (d) $f_4(n)$ is the running time of this Python function for argument n:

```
1  def f4(n):
2      for i in range(0, n):
3          k = 1
4          while k < i:
5              k *= 2
6              print("!")
```

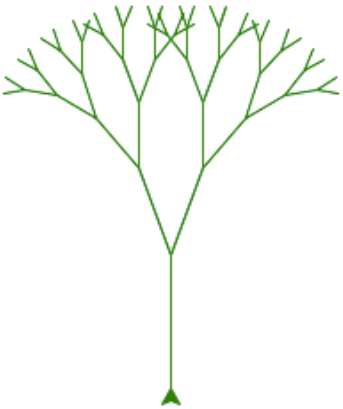
   (a) $f_1(n)$ _____

   (b) $f_2(n)$ _____

   (c) $f_3(n)$ _____

   (d) $f_4(n)$ _____

2. A very special kind of fractal tree can be drawn using the following pseudocode:



```
def drawBranch(length, angle):
    if length >= 1:
        drawLine(length);
        newLength = length / math.sqrt(2)
        newAngle1 = angle + 20
        newAngle2 = angle - 20
        drawBranch(newLength, newAngle1)
        drawBranch(newLength, newAngle2)
```

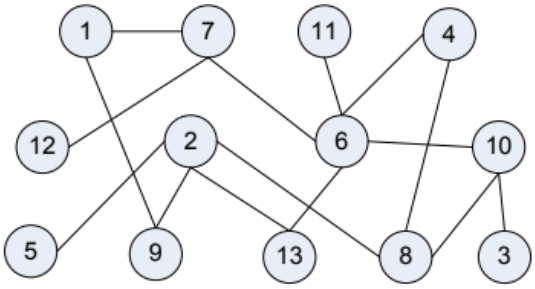
The initial call is `drawBranch(n, 90)` – namely, the initial branch length is n units long and the initial angle is 90°, i.e. vertical. At every subsequent level the branches are shorter by a factor $\sqrt{2}$, and their angle differs from the parent branch angle by $\pm 20°$.

(a) Find the maximum depth of the recursion – how many times `drawBranch(...)` will recursively call itself before the length becomes less than 1 unit. Express the maximum depth of the recursion tree with n – the parameter in the initial call.

(b) Find the number of calls to `drawLine(...)` function in terms of n.

(c) Denote the time complexity of this algorithm by $T(n)$ and find the asymptotic bound of $T(n)$ using the Master theorem.

3. In this problem a denotes the **last** digit of your Student ID. Consider the graph shown in the figure below. Perform the *Breadth-first search* on this graph, starting from the node labeled a. If $a = 0$, then start from the node 10.



If it is possible to visit several neighboring nodes, pick the node with the smallest label. Whenever you visit the node, print its label.

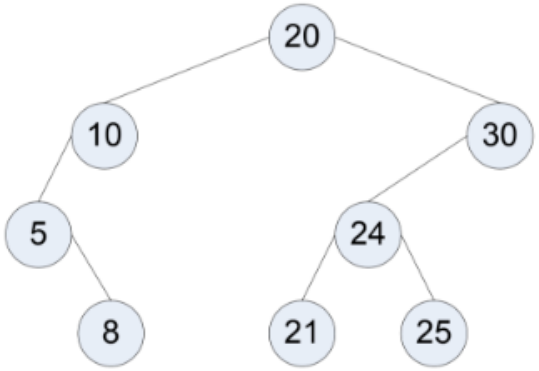
(a) Redraw the graph showing the tree edges in bold or in a different color.

(b) Write the node labels in the order they are printed by the visit function.

4. (a) Explain the purpose of Binary Search Trees (BSTs). What alternative data structures serve the same purpose? Compare their advantages and disadvantages.

   (b) Let a, b be the last two digits from your Student ID. Use them to compute values X, Y, Z, S, T, U (write them in the table).

$S = 3 \cdot b$	_____	$T = 25 + a$	_____
$U = 2 \cdot a$	_____	$X = a$	_____
$Y = (a + b) \bmod 10$	_____	$Z = 5 \cdot (a + b) \bmod 40$	_____



B.INSERT(S)
B.INSERT(T)
B.DELETE(20)
B.SHOW()
B.INSERT(U)
B.INSERT(X)
B.DELETE(25)
B.SHOW()
B.INSERT(Y)
B.INSERT(Z)
B.DELETE(S)
B.SHOW()

The initial state of a binary tree B is shown in the picture above. Run the above commands using the individualized numbers X, Y, Z, S, T, U. Whenever the command is B.SHOW(), draw the current state of the tree. (A command is ignored, if it tries to insert an existing node or to delete a non-existing one.)

5. File `input.txt` contains some undirected graph. Its first line contains number N – the number of nodes in the graph. Nodes are enumerated with numbers $\{1, 2, \ldots, N\}$. The second line contains number E – the number of edges in the graph. After that there are E lines; each line contains two space-separated numbers i j (it defines an edge $\langle i, j \rangle$ where i, j are two vertice numbers). To read numbers from `input.txt`, you can use function `readNumber()` returning a positive `int` (or -1, if the end of the file is reached).

   (a) Define a function `canNumber()` returning `bool` to determine, if the vertices in this graph can be assigned new integer numbers so that every two adjacent vertices i and j have their number total equal to an odd number. (The new integer numbers can be picked arbitrarily – they can repeat and they are unrelated with the original numbers of the nodes.) You can allocate any arrays you need and use integer variables. You can use all integer arithmetic operations including integer divison (`17 / 3 == 5`) and remainder operation (`17 % 3 == 2`).

   (b) Express the time complexity of the algorithm in part (a) in terms of N and E. When there are several possible asymptotic bounds pick the smallest one.