Instructions.

1. Solution may not be submitted by students in pairs.

2. You may submit a pdf of the homework, either printed or hand-written and scanned, as long as it is easily readable.

3. If your solution is illegible not clearly written, it might not be graded.

4. Unless otherwise stated, you should prove the correctness of your answer. A correct answer without justification may be worth less.

5. If you have discussed any problems with other students, mention their names clearly on the homework. These discussions are not forbidden and are actually encouraged. However, you must write your whole solution yourself.

6. In this homework, please solve the questions yourself (without consulting other students). In the previous homeworks, I was trying to encourage brainstorming and to promote fair team work. I hope that you have found it useful. But of course then, under this model of collaboration, your homework grade reflects what you know, but is also influence by what your team-mates know.

So for hw5, do not discuss possible solutions with other students, (nor publicly on Piazza.)

7. Unless otherwise specified, all questions have the same weight.

8. You may refer to data structures or their properties studied in class without having to repeat details, and may reference theorems we have studied without proof. If your answer requires only modifications to one of the algorithms, it is enough to mention the required modifications, and the effect (if any) on the running time and on other operations that the algorithm performs.

9. In general, a complete solution should contain the following parts:

   (a) A high level description of the data structures (if needed). E.g. We use a binary balanced search tree. Each node contains, a key and pointers to its children. We augment the tree so each node also contains a field...

   (b) A clear description of the main ideas of the algorithm. You may include pseudocode in your solution, but this may not be necessary if your description is clear.

   (c) Proof of correctness (e.g. show that your algorithm always terminates with the desired output).

   (d) A claim about the running time, and a proof showing this claim.
1. Analyze the running time of the following function. Give upper bounds (using big-$\mathcal{O}()$ notation) and lower bounds (using big-$\Omega()$ notation) and tight asymptotic bounds (using big-$\Theta()$ notation).

```plaintext
NoNeed(n) {
  if (n ≤ 1) return;
  NoNeed(n-1);
  if (n mod 19 ≠ 0)
    for (i = 1; i ≤ √n; i++) print("*");
}
```

**Answer:** To show an upper bounds, we show that there are at most $\mathcal{O}(n)$ calls to 'NoNeed()', and in each, the number of '*' printed is $\mathcal{O}(\sqrt{n})$. Hence an upper bound is $\mathcal{O}(n\sqrt{n})$.

To show a matching lower bound, we note that there at least $n/2$ call to the recursion. At least in $1/2$ of them, the condition $n mod 19 ≠ 0$, is statified. And the number of '*' printed in each is

$$\geq \frac{\sqrt{n}}{2} \geq \frac{\sqrt{n}}{2}$$

So the total number of '*' printed is at least

$$\frac{n}{2} \cdot \frac{\sqrt{n}}{2} = \Omega(n\sqrt{n})$$

So we found matching upper and lower bounds, and the overall running time is $\Theta(n\sqrt{n})$.

2. Write a recursion formula for the running time $T(n)$ of the function NoNeed, whose code is below.
What is the running time of NoNeed, as a function of $n$?

```plaintext
NoNeed(n) { 
  if (n ≤ 1) return;
  n = 1;
  for (i = 1; i < n; i++) print("*");
  NoNeed(0.8n);
}
```

**Answer:**

$$\{ n + n(0.8) + n(0.8)^2 + n(0.8)^3 \ldots \} = n \{ 1 + (0.8) + n(0.8)^2 + n(0.8)^3 + \ldots \} = n \frac{1}{1 - 0.8}$$

3. Your friend has dropped you off somewhere at a point $p$ on Speedway Blvd. In order to get home, you need to walk to the nearest bus station. However, you are not sure if the nearest station is east or west of your current location, so you are unsure of which direction to go.

Let $d$ be the distance to the nearest bus station. Of course, $d$ is not known to you in advance. Let $w$ be the distance you walked (in the worst case) until reaching the station. The following algorithm has been proposed:

```plaintext
set s = 1 // 1 foot
while (not reached the bus station) {
  walk a distance of s East
  return to p
  walk a distance of s West
  return to p
  s = s + 1
}
```
Assume $d$ is much larger than 1 foot. Express $w$ (as a function of $d$). Is it true that $w = \Omega(d)$? Is it true that $w = O(d)$? What is the ratio $w/d$ in the best case? What is the ratio $w/d$ in the worst case? Prove your answers.

**Clarification** The best case occurs when the bus station is positioned such that the ratio $w/d$ is the smallest possible value. The worst case occurs when the bus station is positioned such that the ratio $w/d$ is as large as possible.

**Answer:**

Let’s consider the walk East of the junction. The total distance is $1 + 2 + 3 \ldots + d = d(d + 1)/2 = \Theta(d^2)$. The walk to the West is similar.

4. Due to reasons not to be disclosed here, but related to the very same friend from the previous question, you have found yourself at a point $p$ in a lake. Crocodiles are habituating the lake, strongly suggesting that you’d swim to the nearest point on a lake’s shore before becoming a crocodile treat. You have a GPS, but no map of the lake. Suggest your strategy, so the distance you have to swim is at most $c \cdot d$. Here $d$ is the distance from your starting position $p$ to the nearest point $q$ of the shore of the lake, and $c$ is a constant (not depending on $d$).

Note that the lake is not convex, but there are no islands in the lake, so its shore is continuous. Show the best value of $c$. You do not have a map, and you don’t know the crocodiles’ locations.

Note also that you do not have to find $q$ itself. As usual, if your solution is iterative, describe what exactly happens at the $i$th iteration (for an arbitrary $i$).

![Figure 1: The lake. You have landed at $p$, and wish to swim to the shore. The nearest point of the shore is $q$. (credit: ClipPanda.com)](image)

**Answer:** We perform a sequence of steps, until a road point is found. For $i = 1, 2, 3 \ldots$

(a) Set $r = 2^i$
(b) Walk from $N_i$, where $N_i$ is a point North of $p$ and at a distance $r$ from $p$
(c) Walk along the circle $C(p, r)$ centered at $p$ and has radius $r$. If no road point is found, you are back at $N_i$

**Analysis:**

Let $q$ be the road point which is the nearest to $p$, and let $q'$ be the point at which we meet the road. Assume we meet the road at the $i$th iteration (so $r = 2^i$). Note that $d(p, q') = r$ and $d(p, q) = d$

Lemma 1.

$$r < 2d(p, q)$$
Proof. At the previous iteration, performing a circle of radius $r/2$, no road point was found. Hence the distance to any road point (and in particular, the nearest one) is at least $r/2$.

Lemma 2. The total distance swum until (and including) the $i$th step is $\leq 13.567r$.

Proof. The distance walked along the circles $C(p,r)$ at the last iteration is $2\pi r$. The total distance walked along the perimeters of all proceeding circles is

$$\leq 2\pi r \left( \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots \right) \leq 2\pi r$$

In addition, we walked north a total distance of $r$.

Putting the two lemmas together, we obtain that the total distance walked is

$$\leq 14r \leq 28d(p,q) = O(d)$$

5. This question refers to the stable marriage problem studied in class. Assume that the preferences lists of all $n$ males and $n$ females are given.

(a) Does there exist a stable matching for every possible valid set of preference lists? Prove or give a counterexample. You may refer to any material studied in class.

(b) Show that there is only one stable matching if for every male, his optimal female and his pessimal female are the same person.

(c) Use part (b) to suggest an $O(n^2)$-time algorithm that determines if, given the preference lists of all the men and women, there exists more than one stable matching.

6. What is the running time of the following recursive function (specified as a function of the input value $n$)? First write a recurrence formula, and show (and prove) its solution.

```c
contraption(n) {
    if n <= 1 return ;
    print("*") ;
    for (i=1 ; i <= 3; i++ ) do contraption(\[n/3\]).
}
```

Here, and in the rest of the questions, you can assume that for asymptotical analysis $k = \lfloor k \rfloor = \lceil k \rceil$ for every integer $k \geq 1$.

Answer: The recursion formula is $T(n) = 1 + 3T(n/3)$. Solution

$$T(n) = 1 + 3 + 3^2 + 3^3 + \cdots + 3^{\log_3 n} = \frac{n \left( 1 + \frac{1}{3} + \frac{1}{3^2} + \cdots + \frac{1}{3^{\log_3 n}} \right)}{1 - \frac{1}{3}} = O(n)$$

Or an easier way: If we image the ternary tree (each non-leaf node has exactly children) depicting the recursion. Each node corresponds to one call of 'contraption', and prints one '*' sign. This tree has $n$ leafs. It has $n/3$ nodes at level 1, (one level above the leaves), $n/3^2$ nodes at level 2 and so on. So the total number of nodes is (with a minuscules mistake)

$$\frac{n}{3} + \frac{n}{3^2} + \cdots = n \left( 1 + \frac{1}{3} + \frac{1}{3^2} + \cdots \right) = n \frac{1}{1 - \frac{1}{3}} = O(n)$$