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.

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. 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.

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

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

2. You have just arrived at a house you have rented. Looking at the faucet in the shower (e.g. the one in Figure 1) you notice that you could turn it to one side and cold water would appear, or to the other direction for warm water. However you have no idea which direction is which. This is a cold day, and you wish for the warm water to appear as soon as possible. Let \( t_{\text{min}} \) be the time (in seconds) that you would need to let the water run until warm water appeared, had you known the correct direction. Suggest an algorithm that guarantees a waiting time of \( \leq K t_{\text{min}} \) time for the warm water. Here \( K \) is a constant that you have to compute (and show your computation), and does not depend on \( t_{\text{min}} \).

![Figure 1:](image)

For simplicity assume (despite being physically inaccurate) that if you set the tap on 'hot', waited some period of time which is less than \( t_{\text{min}} \), and then moved to 'cold', then cold water would fill all the pipes and if you were to move it to 'hot' again, your would have to wait at least \( t_{\text{min}} \).

**Answer:** Write your answer here

3. Analyze the running time of the following function. Give upper bounds (using big-\( O(\cdot) \) notation) and lower bounds (using big-\( \Omega(\cdot) \) notation) and tight asymptotic bounds (using big-\( \Theta(\cdot) \) notation).

```plaintext
NoNeed(n){
  If(\( n \leq 1 \)) return;
  NoNeed(n-1) ;
  If(\( n \mod 19 \neq 0 \))
    for(\( i = 1 ; i \leq \sqrt{n} ; i++ \)) print("*") ;
}
```

**Answer:** Write your answer here

4. Assume you are given a singly connected linked list. The first node is pointed to by a pointer \( \text{head} \). Each node \( v \) has a pointer \( v \rightarrow \text{next} \), pointing to the next node. The last node’s \( \text{next} \) pointer points to NULL. There are \( n \) nodes in the list. We run the following code
while (head→next ≠ NULL AND head→next→next ≠ NULL){
    p = head;
    while (p→next ≠ NULL AND p→next→next ≠ NULL){
        p→next = p→next→next;
        p = p→next;
    }
}

What is its asymptotic running time, as a function of \( n \)? You should use \( O() \), \( \Omega() \) and \( \Theta() \) in your answer.

5. Write a recursion formula for the running time \( T(n) \) of the function \( \text{NoNeed} \), whose code is below. What is the running time of \( \text{NoNeed} \), as a function of \( n \) ?

\[
\text{NoNeed}(n) \{
\begin{align*}
\text{if } (n ≤ 1) & \text{ return; } \\
\text{i = 1 ; } \\
\text{for}(i = 1; i < n; i + +) & \text{ print("*");} \\
\text{NoNeed( 0.8n ) ; }
\end{align*}
\}
\]

Answer: Just an example

6. Assume that we need to store a stream of keys arriving to our program. We would like to store them in an array \( A[1...n] \). The first arriving key must be stored at location \( A[1] \), the second at \( A[2] \), the third at \( A[3] \) and so on. However, we have no idea about the number of keys that we will receive, and our programming language (such as C) requires that we allocate any contiguous array we are using. Hence, the following solution is proposed (see Figure 2): Start by allocating an array \( A[1...4] \). We set \( m = 4 \), and set \( i = 1 \). The following code is executed upon arrival of a new key.

\[
\text{Algorithm \text{Insert}(x,A)} \\
\text{Input: } x \text{ is the } i\text{th key that has arrived. } A \text{ is an array of size } m. \\
\text{1. } \\
\text{(* So far we have written keys into } A[1...i-1] \text{ while } A[i,...m] \text{ is still free. *)} \\
\text{2. } A[i] \leftarrow x ; i + + \\
\text{3. if } i = m \\
\text{4. then (expanding the array *)} \\
\text{5. } \text{Allocate a new array } B[1...2m]. \text{ ( In ‘C’, you might use the command ‘malloc’ or ‘new’ *))} \\
\text{6. for } j = 1...m, \text{ copy } B[j] \leftarrow A[j]. \\
\text{7. Clear all contents of } A, \text{ and rename array } B \text{ as array } A \text{ (constant time operation)} \\
\text{8. } m \leftarrow 2m.
\]

This procedure is sometimes referred to as the Dynamic Table Technique. We will refer to it as expanding the array. . Questions:

(a) What is the worst case running time for a single \( \text{Insert} \) operation, as a function of \( |A| \), the length of \( A \)?

(b) Starting from an initially empty array \( A \), what is the worst case running time for a sequence of \( n \) \( \text{Insert} \) operations?

(c) Let \( \alpha > 0 \) be a positive constant. Assume that we modify the \( \text{Insert} \) algorithm such that once the current array \( A \) is full, we allocate a new table \( B \) of size \( \lceil |A| (1 + \alpha) \rceil \), copy all keys from \( A \) into \( B \), and rename \( B \) as \( A \). Here \( |A| \) is the previous size of the array.
You have found yourself at a point $p$ in a lake. Crocodiles are inhabiting the lake, we strongly suggesting that you’d swim to the nearest point on a the 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 $i$).

You could not use techniques proposed at www.youtube.com/watch?v=kmH0PP_zAKo.
8. To implement the TMA efficiently, we need a data structure that supports the following operations:
   Once male $i$ and male $j$ both propose to a female, the algorithm could decide in efficiently which of them is higher on her list, and reject the other.

   How much time (in the worst case) is needed for this operation if we only use the data structure as described on the slides?

   Suggest an algorithm that re-arrange the data (during a preprocessing stage, so this operation takes $O(1)$ time. The preprocessing time is $O(n^2)$).

9. You are given a sorted array $A[1 \ldots n]$ of keys, and another key $x$ which is stored somewhere in $A$. Show how to find the index $k$ so that $A[k] = x$ in time $O(1 + \log k)$. (Note $k$ is not given, and can be much smaller than $n$).

10. You have received as input the references lists of $n$ males and $n$ females similar to the way the input for the Stable Marriage Algorithm is specified in the slides. Here $n$ is an arbitrary large number.

    Show an example by which the number of rejections occurring during the TMA is as large as possible. What is this number, as a function of $n$? What is this number, as a function of $n$? Prove your answer.

    Now repeat this question but now your goal is to have as few ‘rejections’ as possible.

11. Consider again the preferences lists of the stable marriage algorithm. Assume that the ranking lists of all women by the men are the same, and analogously, the ranking of all men by the women are the same. In other words, there is a consensus between the women on who the favorite man is, who the second favorite man is, and so on.

    Prove that given this scenario, there is only one stable matching. What is it? (note that in general, the TMA finds one of possible multiple stable matching)

12. Consider the TMA algorithms when roles are switched (only females allowed to make proposals).

    Suggest a data structure so once a male receives two proposals, from women $i$ and $j$, he could determine in $O(1)$ time which one is higher in his preference list, and decide which proposal to reject. The overall running time of the algorithm, including the time for TMA, is $O(n^2)$.

---

1The legal verbal “$n$ is given to you” means that your answer is not correct if you assume, say, that $n$ is only 4. Or only for some specific value. That is, the way you build the input should be valid for any $n$. 

---

Figure 3: 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)
13. (a) Let $d$ be a fixed positive integer. Next consider a perfect SkipList constructed as follows: In order to create the $i$th level $L_i$ of the SkipList, we scan the keys of level $L_{i-1}$, and promote to $L_i$ every $d$th key. So for example, the perfect SkipList discussed in class uses the value $d = 2$. The case $d = 3$ implies that every third key is promoted, and so on.

i. What exactly is the worst case running time of $\text{find}(x)$, as a function of both $d$ and $n$?

ii. Which value(s) of $d$ will you think would lead to good performances, and which are poor choices? Why?

(b) In this question, consider a SkipList $L$ created by inserting a set $S$ of $n$ keys into an (initially empty) SkipList. As seen in class, if a key $x$ appears in level $i$, the probability that it also appears in level $i+1$ is $\frac{1}{2}$.

Assume that we re-create a SkipList by inserting the same keys, in the same order, but this time this probability $p$ is 0.01. (if a key $x$ appears in level $i$, the probability that it also appears in level $i+1$ is 0.01.) Will the expected time to perform $\text{find}(x)$ operation increase or decrease, compared to the expected time for the same operation in the original SkipList $L$?