Instructions.

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

2. You could 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 not clearly written, it might not be graded.

4. Prove the correctness of your answer. A correct answer without a proof might not be accepted.

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

6. Unless otherwise specified, all questions have same weight.

7. You could refer to a data structure studied in class, and just mentioned that

8. If your answer uses one of the data structures or algorithms that were studied in class, you could refer to it without having to repeat details studied in class. If you answer requires only modifications of one of the algorithm, it is enough to mention the required modifications, and what’s the effect (if any) on the running time and on other operations that the algorithm performs.

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

(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) High level description of the main ideas of the algorithms

(c) Pseudo code (might not be necessary if the high level description is clear).

(d) Proof of correctness (why your algorithm provides what is required).

(e) A claim about the running time, and a proof showing this claim.
1. What is the running time of the following function (specified as a function of the input value $n$) ?

```plaintext
Read(n)
  i = 0; s = 0;
  while (s <= n) {
    i ++;
    s = s + 2i;
    print("*";
  }
```

2. Assume you are given a singly connected linked list. The first node is pointed by a pointer `head`. Each node $v$ has a pointer $v\rightarrow next$, pointing to the next node. The last node's `next` pointer pointed points to NULL. There are $n$ nodes in the list. We run the following code

```plaintext
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 could use $O()$, $\Omega()$ and $\Theta()$ in your answer.

3. Assume that we need to store a stream of keys arriving to our program. We would like to store them in an array $A[1\ldots n]$. The First arriving key should be stored at $A[1]$, the second at $A[2]$, the third at $A[3]$ and so on. However, we do not have any estimation about the number of keys that are about to arrive, and our programming language (like 'C') requires that we allocate ourselves any contiguous array we are using. Hence the following solution is proposed. We start by allocating an array $A[1\ldots 4]$. We set $m = 4$, and set $i = 1$. The following code is executed upon arrival of a new key.

```plaintext
Algorithm Insert(x, A )
Input: $x$ is the $i$th key that just arrive. $A$ is an array of size $m$
($*\ \text{So far we have written keys into } A[1\ldots i-1] \text{ while } A[i,\ldots m] \text{ is still free.} \*)
1. $A[i] \leftarrow x ; \ i++$
2. if $i = m$
3. then
4. $m \leftarrow 2m.$
5. Allocate a new array $B[1\ldots 2m]$. ($* \ \text{In ‘C’, you might use the command ‘malloc’ or ‘new’} \*)$
6. for $j = 1\ldots m$, copy $B[j] \leftarrow A[j]$.
7. Free the memory from $A$, and rename $A$ to $B$ (constant time operation)
```
This solution commonly refers to as **Dynamic Table Technique**. Questions:

(a) What is the worst case running time for a single insert operation?
(b) What is the running time for a sequence of \( n \) insert operations?
(c) Let \( \alpha > 0 \) be a positive constant. Consider the process of inserting new keys into a table \( A \). Assume that we modify the algorithm such that once the current table \( A \) is full, we allocate a new table \( B \) of size \( \lceil m(1 + \alpha) \rceil \), and copy all keys from \( A \) into \( B \). Here \( m \) is the number of entries (size) of \( B \).

Note that the algorithm \( \text{Insert}(x, A) \) discussed above class is just a special case of this algorithm, when \( \alpha = 1 \).

What is the running time (as a function of \( n \) and of \( \alpha \)) of this algorithm when inserting a set of \( n \) keys into an (initially empty) table \( T \) of size 1?

(d) What would be the running time of inserting a sequence of \( n \) keys into the array, if we use the following rule: Each time that \( A \) is full, we allocate a new table \( B \) of size \( m + \alpha \) and copy all keys from \( T \) into \( T' \). Here \( m \) is the number of entries (size) of \( A \).

4. Your friend has dropped you at some point on Speedway Blvd, Tucson. You need to walk to the nearest bus station, but you are not sure if the nearest bus station is East or West of your location, so you are not sure which direction to go.

Let \( d \) be the distance to the nearest bus station. Suggest an algorithm that guarantees that the total distance you would walk is \( \Theta(d) \). Of course, you cannot use other people, cellphones, maps etc.
5. This question refers to the stable marriage problem studied in class. Assume that the preferences lists of \( n \) males and \( n \) females are given.

(a) Is it true that for the given preferences lists, there is always a stable pairing? Prove or give a counter example. You could refer to any material studied in class.

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

(c) Use this observation to suggest an \( O(n^2) \)-time algorithm that determines if is there is more than one stable pairing.

6. (a) You are given a sorted array \( A[1..n] \) 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 \( n \)).

(b) Same, but now find \( k \) in time \( O(1 + \log(\min\{k, n-k\})) \).

7. What is the running time of the following code, as a function of \( n \)?

\[
\text{Read}(n) ; \\
k = 2 ; \\
\text{while}(k \leq n) \{ \\
\quad k = k^2 ; \\
\}
\]

BTW- you might find interesting that the running time \( T(n) \) of the code of the following code

\[
\text{Read}(n) ; \\
k = 2 ; \\
\text{while}(k \leq n) \{ \\
\quad k = 2^k ; \\
\}
\]

Well, \( T(n) \) increases as \( n \) increases, but does it quite slowly and gracefully. After 3 iterations, \( k = 2^{(2^2)} = 16 \), while after the four iteration, \( k = 2^{2^{(2^2)}} = 2^{16} \). Check how large should \( n \) be so 6 iterations are needed.

There is a special name for a function that describes this behavior. It is called \( \log^* \) (pronounced “log-star”). Here \( \log^*(n) \) indicates the number of iterations needed for our code to run until \( k \) exceeds \( n \). We will get back to this function when discussing Union/Find, toward the end of the semester.

8. When we discussed in class the stable marriage algorithm, we did not discuss how exactly a women receiving proposals from two men \( b_1, b_2 \) decides which one to reject. For the algorithm to run in \( O(n^2) \), she should be able to do so in \( O(1) \) time.

Explain exactly how this part of the algorithm works. You might need some preprocessing of the data before running the algorithm itself.
9. 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 who is the most favourite man, the second favourite man and so on.

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

10. Show preferences lists of 2 men and 2 women, so that more than one stable matching exists.