Q4)

a) $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n \log n}$

$a=2, b=4, f(n) = \sqrt{n \log n}$

$\sqrt{n \log n} \in \Theta(n^k \log^p n) \rightarrow k = \frac{1}{2}, p = \frac{1}{2}$

$\log_b^a = \log_4^2 = \frac{1}{2} \rightarrow p > -1 \rightarrow T(n) \in \Theta(n^{1/2} \cdot \log^{3/2} n)$

b) $T(n) = 9T\left(\frac{n}{3}\right) + 5n^2$

$a = 9$

$b = 3$

$f(n) \in \Theta(n^d) \rightarrow d = 2$

$9 = 3^2 \rightarrow T(n) \in \Theta(n^2 \log n)$

c) $T(n) = \frac{1}{2} T\left(\frac{n}{2}\right) + n$

$a = \frac{1}{2}$

$b = 2$

$f(n) \in \Theta(n^d) \rightarrow d = 1$

$\frac{1}{2} < 2^1 \rightarrow T(n) \in \Theta(n)$

d) $T(n) = 5T\left(\frac{n}{2}\right) + \log n$

$a = 5, b = 2, f(n) = \log n$

$\log n \in \Theta(n^k \log^p n) \rightarrow k = 0, p = 1$

$\log_b^a = \log_2^5 > k \rightarrow T(n) \in \Theta(n^{\log_2^5})$

e) $T(n) = 4^n \cdot T\left(\frac{n}{5}\right) + 1$

$a = 4^n$

Master theorem cannot be applied because a is not constant.

F) $T(n) = 7T\left(\frac{n}{4}\right) + n \log n$

$a = 7, b = 4, f(n) = n \log n$

$\log n \in \Theta(n^k \log^p n) \rightarrow k = 1, p = 1$

$\log_b^a = \log_4^7 > k \rightarrow T(n) \in \Theta(n^{\log_4^7})$

g) $T(n) = 2T\left(\frac{n}{3}\right) + \frac{1}{n}$

$a = 2, b = 3, f(n) = \frac{1}{n}$

$\frac{1}{n} \in \Theta(n^d) \rightarrow d = -1 \rightarrow$ Master theorem cannot be applied because $d < 0$

h) $T(n) = \frac{2}{5} T\left(\frac{n}{5}\right) + n^5$

$a = \frac{2}{5}$, $b = 5$, $f(n) = n^5$

$n^5 \in \Theta(n^d) \rightarrow d = 5$

$\frac{2}{5} < 5^5 \rightarrow T(n) \in \Theta(n^5)$

Q2) {3, 6, 2, 1, 4, 5}

First step: Compare first two element.

3 > 6 → They are in correct order, no need to change.

Second step: Compare next two element.

2 < 6 → Swap then. Move an element back.

{3, 2, 6, 1, 4, 5}

2 < 3 → Swap then. There is no element to compare.

{2, 3, 6, 1, 4, 5}

Third step: Continue from next element.

1 < 6 → Swap then. Move an element back

{2, 3, 1, 6, 4, 5}

1 < 3 → Swap then. Move an element back.

{2, 1, 3, 6, 4, 5}

1 < 2 → Swap then. There is no element to compare.

{1, 2, 3, 6, 4, 5}

Fourth Step: Continue from next element.

4 < 6 → Swap then. Move an element back.

{1, 2, 3, 4, 6, 5}

4 > 3 → They are in correct order. No need to go an element back because rest of the array is already sorted.

Fifth Step: Continue from next element

5 < 6 → Swap then. Move an element back

{1, 2, 3, 4, 5, 6}

5 > 4 → They are correct order and rest of the array contains elements less than 5. So, there is no need to go back. The array is sorted.

Q3)

a) Analyze the worst-case time complexity.

i) Accessing the first element takes constant time in both array and linked list. Because the head of the element linked list is kept. And accessing an element address known has $O(1)$ time complexity in linked list. Accessing any element in array has $O(1)$ time complexity.

ii) Accessing the last element takes constant time in array. Because in array, every element is reachable in constant time by index operator. However, in linked list accessing last element has $O(n)$ time complexity unless tail node address is not kept. Because, the address of every element is kept on previous node in linked list. So, to reach the last element every element must be visited.

iii) Accessing any element in the middle takes constant time in array. On the other hand, it has $O(n)$ time complexity because the reasons indicated previous cases.

iv) Adding a new element at the beginning takes constant time in linked list, because all it needs that changing the element address which head points at with the new element address, and adding old first element address to new element. Contrary to linked list, adding a new element at the beginning has $O(n)$ time complexity in array. Because, even we assume that array has enough space to add an element, all elements must be move by one right to place the new element at the beginning.

v) Adding a new element at the end takes constant time if the array has enough space. On the other hand, even though adding an element to linked list takes constant time, to add an element at the end linked list must be traversed until the end. So, the whole operation, going to last element and add a new elements has $O(n)$ time complexity.

vi) Adding a new element has $O(n)$ time complexity for both linked list and array. Because, moving the elements from middle to the end takes linear time in array, and reaching middle element takes linear time in linked list.

vii) Deleting the first element takes constant time in linked list for the the same reason with adding the first element. Deleting the first element has $\Theta(n)$ time complexity in array because every element must be moved one left.

viii) Deleting the last element has $\Theta(n)$ time complexity in linked list. Because, all linked list must be traversed to reach last element. On the other hand, it takes constant time in array.

ix) Deleting any element in the middle has $\Theta(n)$ time complexity in both linked list and array. Because accessing middle element in linked list and moving half of the elements one left performed in linear time.

b) Linked list need more memory than array. Because in addition to memory which array use to keep elements, linked list elements keep the address of next element. So, if the space which array use is n, the space which linked list use is at least $2n$.

Q4)

```
procedure convertBTtoArray(head, array, index)
    if head == null or head.right == null or head.left == null then
        return mergesort(array)
    endif
    call convertBTtoArray(head.left, array, index)
    array[index] = head.data
    call convertBTtoArray(head.right, array, index+1)
end

procedure convertArraytoBST(head, array, index)
    if head == null or head.right == null or head.left == null then
        return
    endif
    call convertArraytoBST(head.left, array, index)
    head.data = array[index]
    call convertArraytoBST(head.right, array, index+1)
end

procedure convertBTtoBST(head)
    create array[]
    array = convertBTtoArray(head, array, 0)
    mergeSort(array)
    call convertArraytoBST(head, array, 0)
end
```

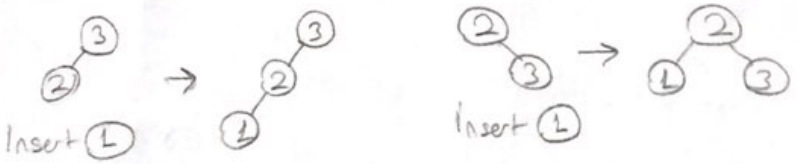
5)

```
procedure searchpair(array, x, n)
    for i=0 i<n
        if hashmap.containskey(x+array[i]) or hashmap.containskey(array[i]-x) then
            return
        endif
        else
            hashmap.insert(array[i]
        endelse
    end for
    end
```

for loop performs n times and it contains only constant time operations such as searching or adding an element to hashmap. Get ith element of array, check whether there is any value meets the condition. If there is, stop the for loop, otherwise it add the current value to hashmap and continue.

6)

a) Shape of a BST depends on insertion order. This statement is true because when adding an element to BST, root node is checked firstly. If the root node is less than the element continue same procedure on right node. Otherwise this procedure continues on left node.



b) The time complexity of accessing an element of a BST might be linear in some cases. This statement is true because if the shape of BST is pathological (every parent node only one child) accessing an element is linear. It is simply a linked list.

c) Finding an array's maximum or minimum element can be done in constant time. This statement is false because all elements must be checked to find max or min. It is impossible to find without comparing max with every element.

d) The worst-case time complexity of binary search on a linked list is $O(\log n)$ where n is the length of the list. This statement is false because if the last element is searched, reaching last element has $O(n)$ time complexity.

e) Worst-case time complexity of the insertion sort algorithm is $O(n)$ if the given array is reversely sorted. This statement is false because if the array is reversely sorted, in every step of insertion sort, it must go until the beginning of the array. So, $1+2+3+\dots+n-1 = \frac{n \cdot n+1}{2} = \frac{n^2+n}{2} \in \Theta(n^2)$

Q4 - continue)

Let convertBSTtoArray time comp be $T(n)$

$T(n) = 2T\left(\frac{n}{2}\right) + 1$

$a=2, b=2, f(n)=1$

$1 \in \Theta(n^d) \rightarrow d=0$

$a > b^d$

$2 > 2^0 \rightarrow T(n) \in \Theta(n^{\log_b^a}) \rightarrow T(n) \in \Theta(n^{\log_2^2}) = \Theta(n)$

convertArraytoBST has same mechanism with convertBTtoArray, so time comp. of convertArraytoBST is $\Theta(n)$.

convertBTtoBST time complexity depends on sorting algorithm because it takes longer then other procedures.

Best, worst and avarage time comp of mergeSort is $\Theta(n \log n)$.

mechanism: In this procedures, firstly traverse the binary tree and put all elements to the array. After all elements are visited, sorts the array by ascending order. Then traverse the binary tree preOrder, while traversing puts array elements one by one.