Q1)

a) $T(n) = 2T(\frac{n}{4}) + \underbrace{\sqrt{n \log n}}_{f(n)}$

$a = 2$, $b = 4$ $\quad \log_4^2 = \frac{1}{2}$

$f(n) = n^{\frac{1}{2}} \cdot \log^{\frac{1}{2}} n \in \Theta(n^k \log^p n)$

$k = \frac{1}{2}$ $\quad p = \frac{1}{2}$

↳ so → $\log_b^a = k$

$\log_4^2 = \frac{1}{2}$ $\Rightarrow \Theta(n^{\frac{1}{2}} \cdot \log^{\frac{1}{2}} n)$

$= \Theta(\sqrt{n \log n} \cdot \log n)$

b) $T(n) = 9\,T(\frac{n}{3}) + \underbrace{5n^2}_{f(n)}$

(a = 9, b = 3)

① $\log_b^a = \log_3^9 = 2$

② k: $f(n) = 5n^2 \in \Theta(n^2) \in \Theta(n^k \log^p n)$

$k = 2$, $p = 0$

so $\log_b^a = k$

↳ result is $\Theta(n^k \cdot \log^{p+1} n) \Rightarrow \Theta(n^2 \log n)$

c) $T(n) = \frac{1}{2} \cdot T(n/2) + n$

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

① "a" does not satisfy this condition:

$a \geq 1$

② $f(n)$ n

So it cannot be solved by Master Theorem

So

$p \geq 0$

---

d) $T(n) = 5 \cdot T(n/2) + \log n$

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

① $\log_b^a \rightarrow \log_2^5$

② k: $f(n) = \log n \in \Theta(\log n) \in \Theta(n^k \cdot \log^p n)$

$k=0, \; p=1$

So $\log_b^a > k$ then $\Theta(n^{\log_b^a}) = \Theta(n^{\log_2^5})$

e) $T(n) = 4^n \cdot T(n/5) + 1$

It cannot be solved by Master Theorem, because "a" is not a real number

f) $T(n) = 7\, T(n/4) + n\log n$

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

① $\log_b^a := \log_4^7$

② k: $f(n) = n\log n \in \Theta(n\log n) \in \Theta(n^k \cdot \log^p n)$

$k = 1$, $p = 1$

So $\log_b^a > k$

then $\Rightarrow \Theta(n^{\log_b^a}) = \Theta(n^{\log_4^7})$

$= \Theta(n^{1.4})$

g) $T(n) = 2 \cdot T(n/3) + 1/n$

(a = 2, b = 3, $f(n) = 1/n$)

① $\log_b^a \doteq \log_3^2$

② k: $f(n) = \frac{1}{n} \in \Theta(\frac{1}{n}) = \Theta(n^k \cdot \log^p n)$

$k = -1$, $p = 0$

! It cannot be solved by master theorem because $k < 0$

h) $T(n) = \frac{2}{5} \cdot T(n/5) + n^5$

It cannot be solved by master theorem because a is smaller than 1 (not satisfy $a \geq 1$)

Q2) $A = \{3, 6, 2, 1, 4, 5\}$

1- $\{3, 6, 2, 1, 4, 5\}$ → no swapped

2- $\{3, 6, 2, 1, 4, 5\}$

$\{3, 2, 6, 1, 4, 5\}$ → 2 and 6 swapped

$\{2, 3, 6, 1, 4, 5\}$ → 2 and 3 swaped

3- $\{2, 3, 6, 1, 4, 5\}$

$\{2, 3, 1, 6, 4, 5\}$ → 1 and 6 swapped

$\{2, 1, 3, 6, 4, 5\}$ → 1 and 3 swapped

$\{1, 2, 3, 6, 4, 5\}$ → 1 and 2 swapped

4- $\{1, 2, 3, 6, 4, 5\}$

$\{1, 2, 3, 4, 6, 5\}$ → 4 and 6 swapped

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

{1, 2, 3, 4, 5, 6} → 5 and 6 swapped

so the updated array:

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

A marker (i) seperates sorted and unsorted parts of array. At the very first step, marker seperates first element and the others. After each step maker goes right. When marker arrives the end, sorting is done.

Q3)

i) Accesing the first element

- $T_{Arr} \in \Theta(1)$ → Arrays are placed memory as a block so we can calculate each elements position so that we can reach at constant time

$T_{LL} \in \Theta(1)$ → Head pointer points to the head of the linked list so this is constant time

ii) Accesing the last element

$T_{arr} \in \Theta(1)$ → We can reach each element in constant time in arrays. (explained in first question)

$T_{LL} \in \Theta(n)$ → if LL is singular linked list. We must iterate each element one by one

iii) Accesing the any element in the middle

$T_{arr} \in \Theta(1)$ → We can calculate the position of the first element. so constant time (if we know the size of array)

$T_{LL} \in \Theta(n)$ → We have to visit first $n/2$ elements. so $\Theta(n/2) \in \Theta(n)$

iv) Adding a new element to beginning

$T_{arr} \in \Theta(n)$ → We have to shift each element to the right (n shifting operation)

$T_{LL} \in \Theta(1)$ → Just change the head pointer's value and connect new node to the second one.

V) Adding a new element to the end

$T_{arr} \in \Theta(n)$ → if Arr is full we need to resize. This will linear time otherwise $\Theta(1)$

$T_{LL} \in \Theta(n)$ → iterate the last element and add

vi) Adding a new element to the middle

$T_{Arr} \in \Theta(n)$ → Shifting will take $n/2$ movements so $\Theta(n/2) \equiv \Theta(n)$

$T_{LL} \in \Theta(n)$ → Iterate to the middle element $\Theta(n/2) \equiv \Theta(n)$

vii) Deleting the first element

$T_{arr} \in \Theta(n)$ → shifting $n-1$ elements

$T_{LL} \in \Theta(1)$ → Just change what head pointer points

viii) Deleting the last element

$T_{arr} \in \Theta(1)$ → We know the last element's position

$T_{LL} \in \Theta(n)$ → The cost here is iterating to the last element. If we already in last element it is constant time.

ix) Deleting from middle

$T_{arr} \in \Theta(n)$ → We have to shift $n/2$ items to the left

$T_{LL} \in \Theta(n)$ → Iterating takes linear time

b) Analyse the space complexities:

i) $S_{Arr} \in \Theta(1)$, $S_{LL} \in \Theta(1)$ — No created, extra space

ii) $S_{Arr} \in \Theta(1)$, $S_{LL} \in \Theta(1)$ — No created extra space

iii) $S_{Arr} \in \Theta(1)$, $S_{LL} \in \Theta(1)$ — No created extra space

iv) $S_{Arr} \in \Theta(n)$ → If it is full we create extra space
$S_{LL} \in \Theta(1)$ → if we add n element it is $\Theta(n)$
if we add 1 element it is $\Theta(1)$

v) $S_{Arr} \in O(n)$ → When Array is full we will create extra space
$S_{LL} \in \Theta(1)$ → just create 1 node extra

vi) $S_{Arr} \in O(n)$ → if Array is full we will Copy it
$S_{LL} \in \Theta(1)$ → just created 1 node

vii) $S_{Arr} \in \Theta(1)$, $S_{LL} \in \Theta(1)$ — No created extra space

viii) $S_{Arr} \in \Theta(1)$, $S_{LL} \in \Theta(1)$ — No created extra space

ix)

SArr $\in \Theta(1)$
SLL $\in \Theta(1)$

] no need extra space

Q4)

```
foo ( BinarySearchTree : bst , BinaryTree : bt)
if (bt == null) then
    return
endif
foo (bst, getLeft(bt))
bst.add(bt.root)
foo (bst, getRight(bt))
return bst
end
```

Best → $\Theta(1)$

Worst → $\Theta(n \log n)$ (When we solve recurrence relation : $T(n) = 2T(n) + \Theta n$)

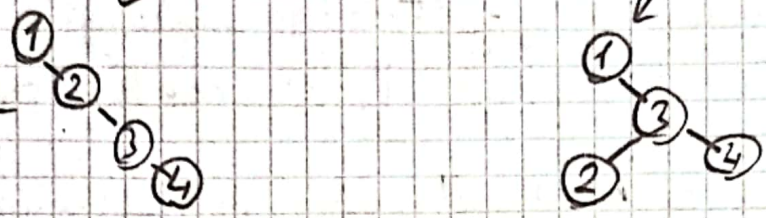
Avarage → $O(n \log n)$

Q5).

```
foo (Arrg: Arr, num)
|
| Create Hashset [ ]
|
| for i = 0 to Arr size
|   | Put i and Arr[i] to the Hashset
| endfor
|
| for i=0 to Arr size:
|   |   if num - Arr[i] in Hashset then:
|   |   |   return Arr[i], num-Arr[i]
|   |   endif
|   |
|   |   if num + Arr[i] in Hashset then:
|   |   |   return Arr[i], num+Arr[i]
|   |   endif
|   |
| endfor
| return null, null.
end
```

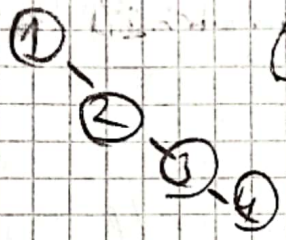
→ At the first we have to create a hashset that contains (index, value) pairs so that we'll be able reach each element at constant time. This filling operation takes linear time ($\Theta(n)$). After that we iterate the arrg and ask hashset that if it contains what we want or not. if contains return values keep going otherwise. This will take $O(n)$

Q6) a) True. [1,2,3,4] , [1,3,4,2]



So it depends the insertion order

b) True. Who BST are like this ;



(Degenerate BST)

c) False: We cannot know our current value is bigger than or not from next index's value so we have to look all of them.

d) False. It is $\Theta(n \log n)$ n comes from finding the middle one and log n comes from binary search so it is $\Theta(n \log n)$.

e) False. It is $\Theta(n^2)$. Each element (n) has to be relocated (shifted)