Master theorem:

$x(n) = a \cdot x(\frac{n}{b}) + f(n)$ where $n = b^k$ $(k = 1, 2, \cdots)$

$x(1) = c$, $a \geq 1, b \geq 2, c > 0$

$f(n) \in \Theta(n^d)$ where $d \geq 0$

$$x(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases} \quad \forall n$$

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

can't be solved since $f(n) = \sqrt{n}\sqrt{\log n} \notin \Theta(n^d)$

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

$a = 9 \quad b = 3 \quad d = 2 \quad b^d = 3^2 = 9$

$a = b^d \Rightarrow T(n) \in \Theta(n^2 \log n)$

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

can't be solved since $a = \frac{1}{2} \not\geq 1$

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

can't be solved since $f(n) = \log n \notin \Theta(n^d)$

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

can't be solved since $a = 4^n$ is not constant

f) $J(n) = 7 \cdot J\left(\frac{n}{4}\right) + n\log n$

can't be solved since $f(n) = n\log n \notin \Theta(n^d)$

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

can't be solved since $d = -1 \not> 0$

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

can't be solved since $a = \frac{2}{5} \not> 1$

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

First element isn't checked

3	6	2	1	4	5	6 is greater than 3, continue
3	2	6	1	4	5	2 is less than 6, swap
2	3	6	1	4	5	2 is less than 3, swap, reached leftmost, continue
2	3	1	6	4	5	1 is less than 6, swap
2	1	3	6	4	5	1 is less than 3, swap
1	2	3	6	4	5	1 is " " 2, swap, reached left-most, cont.
1	2	3	4	6	5	4 is less than 6, swap
1	2	3	4	6	5	4 is greater than 3, cont.
1	2	3	4	5	6	5 is less than 6, swap
1	2	3	4	5	6	5 is greater than 4, cont

finish

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

3

a)

	Array with resize	Array without resize	Linked list
i	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
ii	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
iii	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
iv	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
v	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
vi	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
vii	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
viii	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
ix	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$

b) If resizing the array is needed for addition or deletion of an element, regardless of position, a linear space is consumed since new memory is allocated for the changed array.
If deletion is putting a NULL value instead of deleted element then there's no space allocated.
If as adding an element it is put at an index of a NULL value again no extra space is allocated.

Whenever an element is added to linked-list a Node is created for it so $\Theta(1)$ space complexity. Whenever an element is deleted from linked list there's only deallocation with zero extra allocation, can be said $\Theta(1)$.

```
func bt_traverse_and_add_to_stack (bt  stk)
  if bt.left_sub_tree exists then
    bt_traverse_and_add_to_stack (bt.left_sub_tree  stk)
  end if
  stk.add(bt.value)
  if bt.right_sub_tree exists then
    bt_traverse_and_add_to_stack (bt.right_sub_tree  stk)
  end if
end func
 ...
func bt_to_bst (bt  n)
    stk ← new stack
  Δ bt_traverse_and_add_to_stack ( bt  stk)
 ★ stk ← array_to_stack (quicksort(stack_to_array(stk  n)  n)  n)
  * bt_traverse_and_assign_from_stack ( bt  stk)
end func
```

The idea is to sort the values in the tree and putting them back in order. Function in line * is not written but it does the opposite (mirror) process that is done in line Δ, but it visits first right sub tree this is because quicksort is assumed to be sorting in ascending order.

Best, worst, and average cases are all the same since there's no control flow besides checking if left or right subtree exists, which is done to make sure every node is visited once.

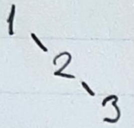
Therefore binary tree to stack, stack to array, and their inverse functions are all linear. Howeve quicksort in line ★ has $\Theta(n \log n)$ time complexity. Therefore the overall time complexity is $\Theta(n \log n)$.

```
func find_pair (arr   x)
    s ← new set
    for e in arr do
        if s.includes(x+e) then
            return (x+e, e)
        end if
        s.add(e)
    end for
end func
```

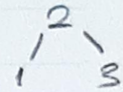
The set s is an unordered set with hashset implementation. Therefore add and includes operations are constant time. The worst case is having the only viable pair's one element at the array end making it iterate every element. Same applies for not finding any pair. So choosing any one of add or includes operation as basic operation will result in linear complexity at max since worst case is doing the operation for all elements.

Therefore it is $O(n)$

a) True. example: add elements {1, 2, 3} in this order:

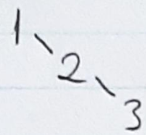


add them in order {2, 1, 3}:



they are different.

b) True. Accessing three (3) in the below BST takes linear time:



c) False. If you were given only two numbers to find the min or max one you need to check both. Since this linear time the process for more numbers can't be even divided to smaller parts to become faster therefore false.

d) False. The binary search on an array is $O(\log n)$ because indexing elements to compare is done at most $(\log n) \cdot c$ times (where c is some constant). Since indexing any element in a linkedlist takes linear time and it is done in $O(\log n)$ time complexity for BST, it makes it $O(n \log n)$ for linkedlist.

e) False. The given condition for worst case is correct but it would take quadratic time because every element (n many) is compared to every element before them ($n-1$ many). Therefore $W(n) \in \Theta(n^2)$