# Master Theorem.

$T(n) = a\,T(n/b) + f(n)$ → general form

$a \geq 1$ $\quad f(n) = \Theta(n^k \log^p n)$

$b > 1$ $\quad \log_b a = ?$

case 1: if $\log_b a > k$ then $\Theta(n^{\log_b^a})$

case 2: if $\log_b a = k$

(a) if $p > -1$ $\quad \Theta(n^k \log^{p+1} n)$

(b) if $p = -1$ $\quad \Theta(n^k \log\log n)$

(c) if $p < -1$ $\quad \Theta(n^k)$

case 3: if $\log_b a < k$

(a) if $p \geq 0$ $\quad \Theta(n^k \log^p n)$

$p < 0$ $\quad \Theta(n^k)$

# CSE 321 - Homework 2

Yusuf Ardan
200104004112

Q.1. "$T(n) = aT(n/b) + f(n)$"

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

$a = 2 \quad \log_b a = \log_4 2 = \frac{1}{2}$

$b = 4$

$f(n) = \Theta(n^k \log^p n) = n^{1/2} \log^{1/2} n$, then

$k = 1/2, \quad p = 1/2.$

$\rightarrow \Theta(n^k \log^{p+1} n) \rightarrow \underline{\Theta(n^{1/2} \cdot \log^{3/2} n)}$

Case 2.a

(b) $T(n) = 9T(n/3) + 5n^2$

$a = 9 \quad \log_b a = 2$ Case 2.a

$b = 3$

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

$k = 2, \quad p = 0$

$\rightarrow \Theta(n^k \log^{p+1} n) = \Theta(n^2 \log n).$

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

$a = \frac{1}{2}$ $a < 1 \rightarrow$ it is unsolvable

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

$a = 5 \quad \log_2 5 > 2$

$b = 2 \quad f(n) = \Theta(n^k \log^p n) = \Theta(\log n)$

$k = 0, \quad p = 1.$

$\rightarrow \Theta(n^{\log_b a}) = \underline{\Theta(n^{\log_2 5})}$ case 1

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

a is not a constant. Unsolvable.

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

$a = 7 \quad \log_4 7 > 1$

$b = 4 \quad f(n) = \Theta(n^k \log^p n) = \Theta(n \log n)$

$k = 1, \quad p = 1.$

$\rightarrow \Theta(n^{\log_b a}) \rightarrow \Theta(n^{\log_4 7})$

case 2.a

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

$a = 2 \quad \log_b a < 1$ and $\log_b a > 0.$

$b = 3 \quad f(n) = n^k \log^p n = n^{-1}$

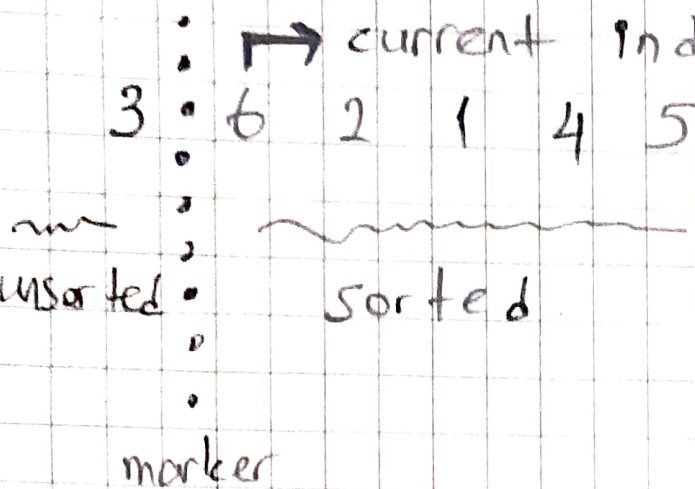
$k = -1, \quad p = 0.$

$\rightarrow \Theta(n^{\log_b a}) = \Theta(n^{\log_3 2})$ case 1

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

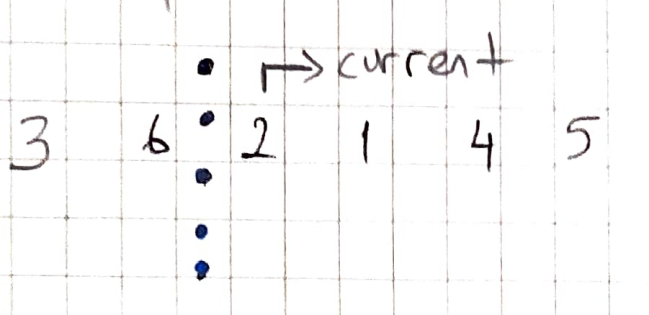
$a < 1$, unsolvable

Q.2. A = { 3, 6, 2, 1, 4, 5 }




**Step-1**

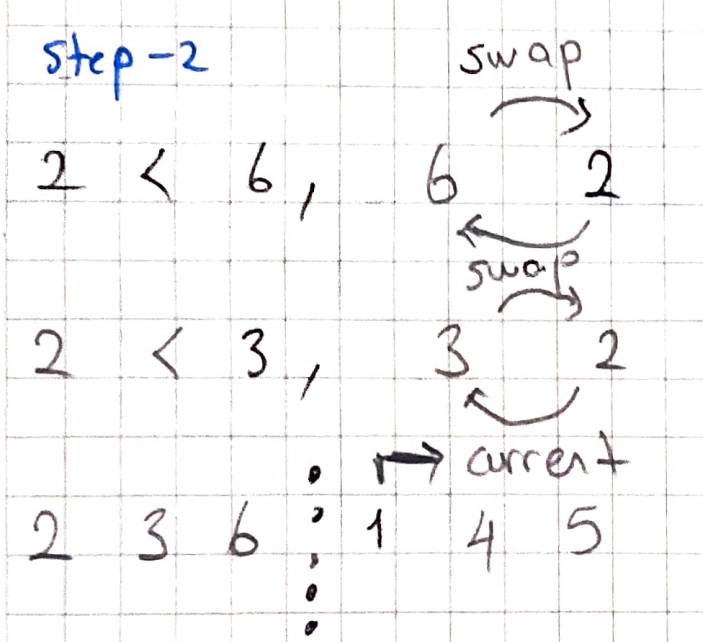
6 > 3, move marker




**Step-2**

2 < 6, swap

2 < 3, swap

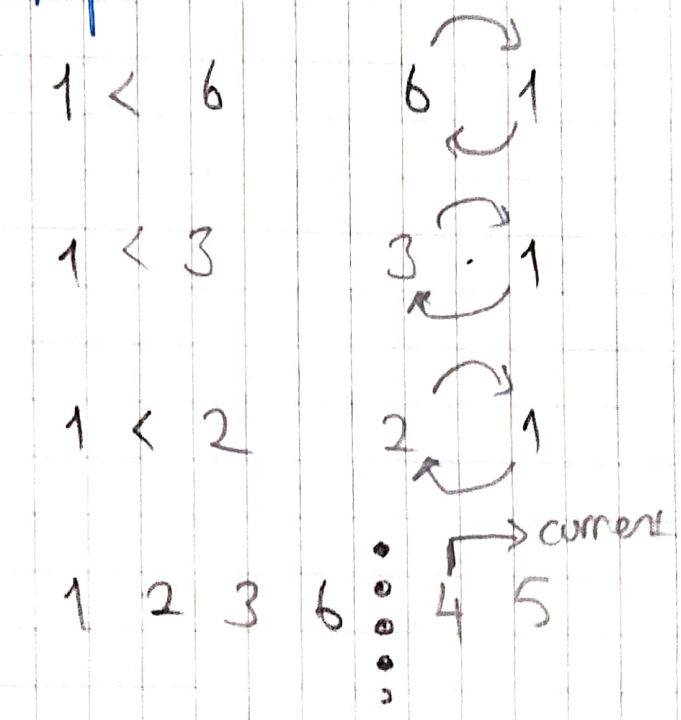



**step-3**

1 < 6

1 < 3

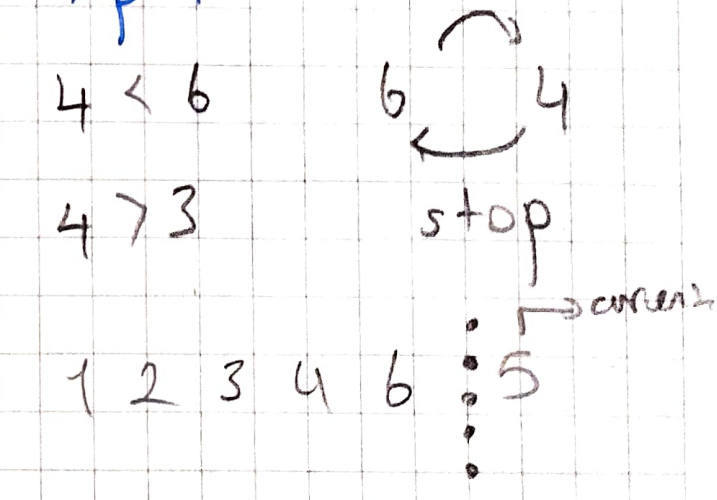
1 < 2




**Step-4**

4 < 6

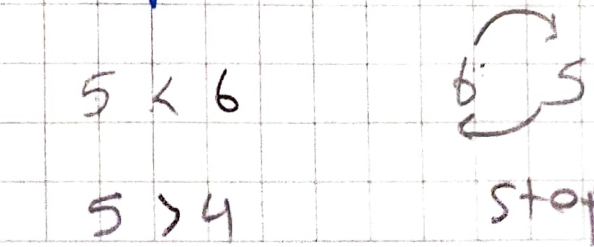
4 > 3 stop




**Step-5**

5 < 6

5 > 4 stop.



Array is sorted now → A = { 1, 2, 3, 4, 5, 6 }

Q.3.

(a) Answers can be change according implementations. But, I'll assume,

- The array is a basic array
- Linked List is a single linked list and we have only access to the first element.

i. accessing first element for both data structures (DS) is constant. $\Theta(1)$

- We have access to the first element of linked list and access in an array is $\Theta(1)$

ii. Accessing the last element.

- for the array is $\Theta(1)$
- for the linked list it is $\Theta(n)$

To access the last element, we have to move on from each node that points to the next node until reach the end.

Number of accessing operations is $(n-1)$ in a linked list.

iii. Accessing any element in the middle

- Accessing any element in an array is constant $\Theta(1)$.

- In any middle element, for worst case scenario, it will be (n-1)th element in linked list.

Therefore, accessing (n-1)th element is $\Theta(n-1)$ and it is simply $\Theta(n)$.

iv. Adding new element at the beginning.

- In array, elements are added to addressed in order. So, adding elements to the beginning requires shifting all previous elements.

number of shifting elements requires is n.
Therefore, it is $\Theta(n)$

- In a linked list, insertion is constant. It is just changing the references of a node. Since, insertion will be perform on first element, the time complexity of the operation is $\Theta(1)$.

v. Adding a new element at the end.

- In array, no shifting operation is requires. Therefore, it is $\Theta(1)$.

- In linked list, although insertion is constant, accessing on last element is $\Theta(n)$.

Therefore, time complexity is $\Theta(n)$

vi. adding a new element in the middle

- In array worst case is adding a new element to the 2nd index.
  # of shifting operations will be n-1
  Therefore, time complexity is $O(n)$.

- In linked list, if insertion will be perform on (n-1)th element is worst case.

  Therefore, it is $O(n)$.

vii. deleting the first element.

- In array, shifting operations required
  # of shifting operations are (n-1) and time complexity is $O(n)$.

- In linked list no move operation will require. It is $O(1)$.

viii deleting the last element.

- in array, just access and remove the element. $O(1)$.

- In linked list, remove also $O(1)$ but accessing to last element is $O(n)$.

1x. deleting any element in middle.

- In array, # of shifting will be (n-2) when deleting the 2nd element. Therefor it is $\Theta(n)$.

- In Linked list, deleting is constant but in worst case, # of accessing will be (n-1) when deleting the (n-1)th element.

Therefore, it is $\Theta(n)$ also.

(b) In an array only space needs to reference the first element of the array.

But, in linked-list, each node keeps the reference of the next node.

No extra space req. in array, but, in Linked list $\Theta(n)$.

In operations above, when shifting required, then also some temporary space will be require to perform shifting.

Adding, deleting in linked list is more space-efficient then an array

Q.4.

```
// Convert bt to bst.
Prosedure bt_to_bst ( bt, length)
        values[] = new[length]
        fill_array (bt.root , values).
        merge_sort ( values)
        bt_to_bst( bt.root, values)
End
// Traverse in-order and replace values sorted
Prosedure bt_to_bst(root, values)
        If root == null
            return
        End If
        bt_to_bst (root.left, values)
        root.value = values.next()
        bt_to_bst (root.right, values).
End
// Traverse Pre-order fill array with bt values
Prosedure fill_array(root, values)
        If root == null
            return
        End If
        values.add (root.value).
        fill_array (root.left)
        fill_array(root.right).
End
```

In above algorithm;

fill_array traverses binary tree and adds all values in binary tree to the values array.

merge_sort sorts the values array.

btToBST, travers binary tree in-order and adds values in array to each node in-order. So, binary tree becomes binary search tree.

(!) Addition only made for array. and it is short version of array[++i]

1- fill_array traverses binary tree visits each node once. Therefore, it is $\Theta(n)$.

2- btToBST(recursive) traverses the tree $\Theta(n)$

3- merge-sort it is $\Theta(n \log n)$.

Best, worst, average cases all $\underline{\Theta(n \log n)}$

Q.5. $A = \{a_0, a_1 \ldots a_n\}$ $x = |a_i - a_j|$

```
Procedure find_pair (A, x):
    HashMap map = new Map()
    For i from 0 to n:
        If x + A[i] is in map
            return (pair (x + A[i], A[i])
        Else if -x + A[i] is in map:
            return (pair (-x + A[i], A[i]))
        Else:
            map.add(A[i])
        End If.
    End For
End
```

In this algorithm iteration will be perform n+1 times. Other map methods are. O(1). The pair can be the first two element, or, the last element can be one of pair.

Best Case = $\Omega(1)$

Worst Case = $O(n)$ Time Com. = $O(n)$.

Q.6.

(a) TRUE

In a BST, new element will be added according to its value compare to other nodes.

The more random the values, the closer the tree will be being balanced.

(b) TRUE

If we add a sorted list to a BST, the tree will be linear. So, access can be linear.

(c) FALSE

To find the max and min element. we have to access all element so it is linear.

(d) FALSE.

The worst case is, either target is on first or last element

number of comparision will be k where $n = 2^k - 1$.

However, getting an element will be work linear. Therefor, it is $\Theta(n \log n)$

Q.6.

(a) TRUE

In a BST, new element will be added according to its value compare to other nodes.

The more random the values, the closer the tree will be being balanced.

(b) TRUE

If we add a sorted list to a BST, the tree will be linear. So, access can be linear.

(c) FALSE

To find the max and min element we have to access all element so it is linear.

(d) FALSE.

The worst case is, either target is on first or last elenet.

number of comparision will be k where $n = 2^k - 1$.

However, getting an element will be work linear. Therefor, it is $\theta(n \log n)$

(e) FALSE

It is the worst case for insertion sort and it is $\Theta(n^2)$.

Because, each new element will be swapped with its adjacent element until it reaches the last element of "sorted" part of array.