21-5-25

AVL Tree insertion & deletion

main clay

Step (1):

```
Node root = null;
```

↓

root is initialized with null initially

Step (2): calling insert class

insert clay

Step (3):

i) `if (node == null)`
`return new Node(key)`

↓

Creating some memory if the node is null

Key = 9

initially root = 9

ii) 5 =>

`if (key < node.key)`

5 < 1000.key

`node.left = insert (node.left, key)`

(9) (5)

iii) 10 =>

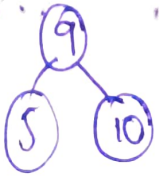
`if (key < node.key)`

10 < 1000.key => 9

`if (key > node.key)`

10 > 1000.key => 9

`node.right = insert (node.right, key)`



Step (4): Height update.

=) node.left − n.right

Step (5) returns root if there is no prob with balance factor

Step (6): If balance factor raise an issues then it will check the rotation

---

* Index of Searches for the index of the character and return the index
* If the character isn't found, it returns "-1"
→ If the given string contains multiple occurences of a specified character then it returns the index of only 1st occurence of the Specified character.

→ IndexOf (char c, int index)

→ charAt (int index)

* A charAt function return the character at the Specified index and we pass the index number inside the string. if the Specified index no doesn't Exist in the String then the fun^en throws an unchecked Exception

* Contains func^n returns true, if the string contains the specified Sequence of character values otherwise, it returns false., the parameter of contains func^n specify the Sequence of character to be Searched and it gives null pointer Exception. If the Sequence is null

* Char Sequence is an interface i.e implemented by the string class and we are using the character Sequence as an argument in the Contains method

* Sliding Window technique is used to Solve the problems using substring, Subarray or a window.
The main idea of this technique is to use the results of previous window for next window
This technique is mainly used for finding the Spec Subarrays by Specific Sum, finding the largest Subarray with unique character

```
* for(int i=0; i<=a.length-w; i++) {
      for(int j=0; j<w; j++) {
          subarrays[i][j] = a[i+j];
      }
  }
```

Bubble Sort

→ main()

→ ob - (obj create) — BubbleSort class

→ a = {64, 34, 25, 12} - declared

→ ob.bubblesort(a) is called to sort the array

→ BubbleSort() method

n = 4 → a = {64, 34, 25, 12}
1 2 3 4

→ outer loop runs n-1 = 3 times

→ Each time, the inner loop compares and swaps adjacent Elements if the left one is greater.

→ Pass 1 (i = 0):

- Compare 64 & 35
  Swap → [34, 65, 25, 12]
- Compare 64 & 25
  Swap → [34, 25, 64, 12]
- Compare 64 & 12
  Swap → [34, 25, 12, 64]

→ Pass 2 (i = 1):

C→ 34 & 25
S→ [25, 34, 12, 64]

C -> 25, 12

S -> [12, 25, 34, 64]

Now, array is sorted

-> Print the sorted array

for loop prints:

12 25 34 64

Insertion Sort

-> main():

- Creates array

arr = {12, 11, 13, 5, 6}

- calls ob.sort(arr) to sort it

-> Process:

i = 1 (key = 11)

Compare with 12 -> 12 > 11 -> Shift 12 right

-> [12, 12, 13, 5, 6]

place 11 -> [11, 12, 13, 5, 6]

i = 2 (key = 13)

Compare with 12 -> 13 > 12 -> No shift

-> [11, 12, 13, 5, 6]

i = 3 (key = 5)

Compare with 13 → Shift → [11, 12, 13, 13, 6]
Compare with 12 → Shift → [11, 12, 12, 13, 6]
Compare with 11 → Shift → [11, 11, 12, 13, 6]
insert 5 → [5, 11, 12, 13, 6]

i: 4 (key: 6)

= [5, 6, 11, 12, 13]

Merge Sort

→ main() =>

int arr[] = {12, 11, 13, 5, 6, 7};

sort(arr, 0, 5);

→ Recursive Splitting

sort(arr, 0, 5)

→ sort(arr, 0, 2)

→ sort(arr, 0, 1):

→ sort(arr, 0, 0) } base case

→ sort(arr, 1, 1) }

→ merge(arr, 0, 0, 1)

arr: [11, 12, 13, 5, 6, 7]

sort(arr, 3, 5)

⋮

arr: [5, 6, 7, 11, 12, 13]