DATA ANALYTICS

Mohamed AbdelHalem_1180182

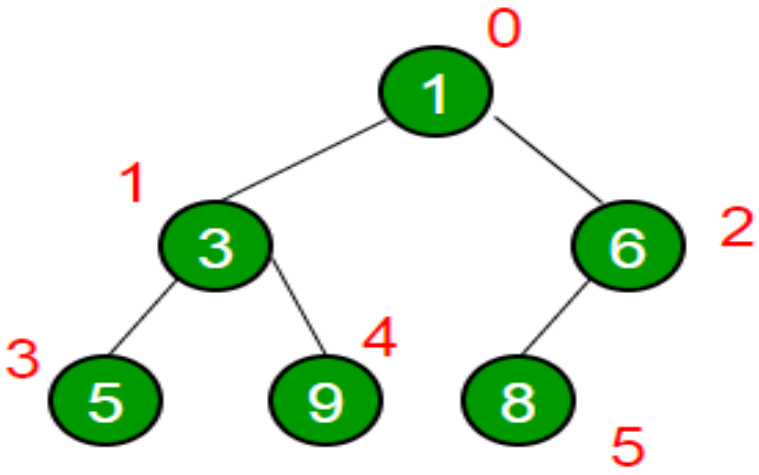
Assignment_6

Problems:

1)Heap sort

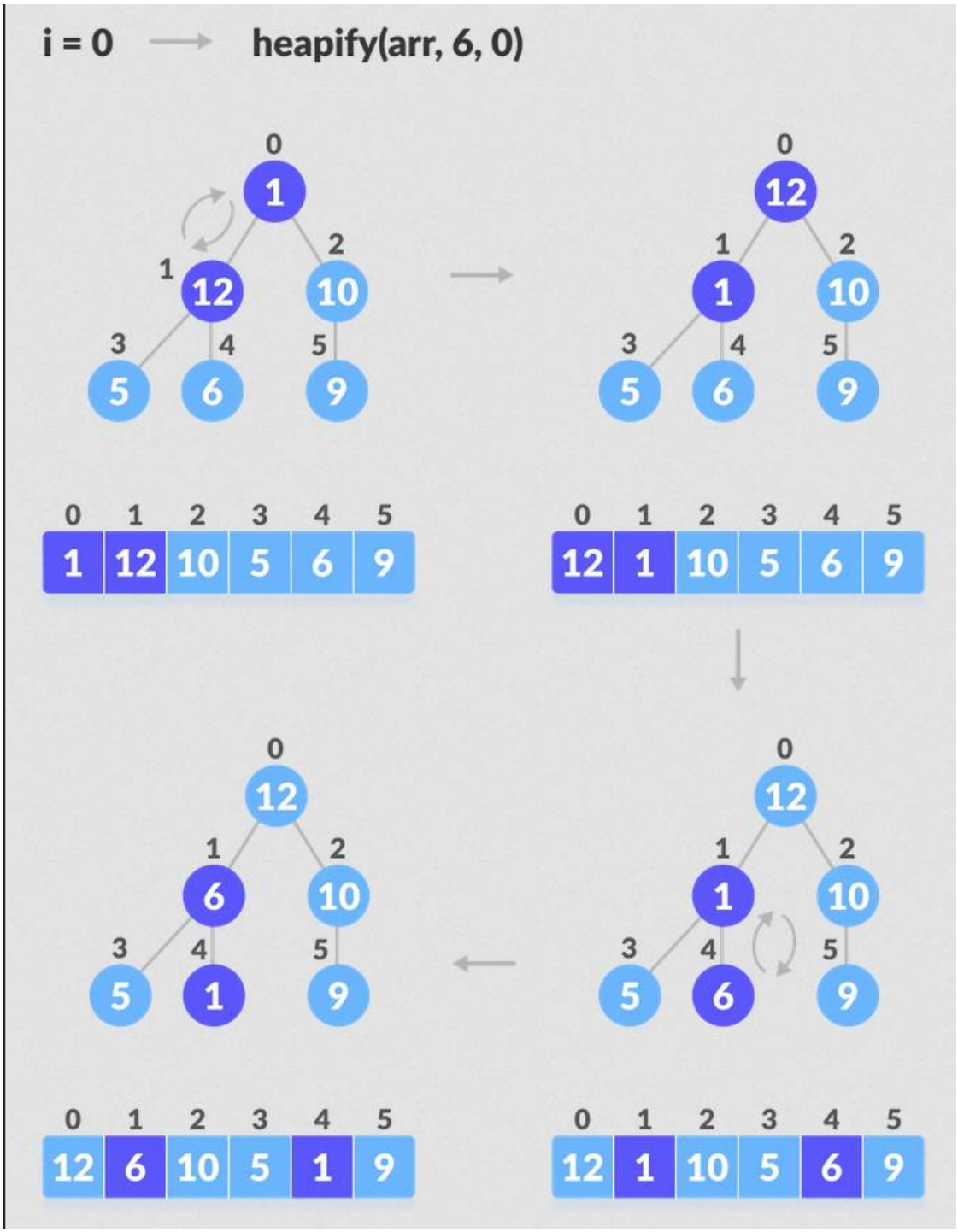
The way I solved this problem is by looping on half of the array to the first element and sifting the elements or happifying by initializing the max as root and getting left and right using the equation 2i+1 and 2i+2 respectively then we check on the length of array that it shouldn’t be exceeded and comparing root to left and right and replacing max with new max and then checking if max isn’t in root swap root to max and heapify again or keep sifting. Then we go to sort function do a for loop from N to 0 and swap root with last element then heapify and get inside the for loop again and swap again root with I and then heapify and we keep doing the process to sort it up as if we are putting max in the last each time so it becomes sorted.

(2i+1) for left and (2i+2) for right



1	3	6	5	9	8
0	1	2	3	4	5

Heapify Function Example(max heap)



Pseudo Code:

Heapify(arr,N,I):

Initialize Min as root

Left=2i+1

Right=2i+2

Check if left exceeds N and if left is less than root

    Swap left with min

Check if right exceeds N and if right is less than root

    Swap root with min

Check if min!=I

    Swap I with min

    Heapify again using new min (arr,N,min)

Heap sort(arr)

Loop from len(arr)//2 to 0

    heapify (arr,n,I)

Then loop from N to 0

Swap elements

Heapify(arr,I,0) // heapifying root element

To get output finally in array shape

Problems:

2)Build Heap

The way I solved this problem is same as problem above but the difference is on each swap in happify function we append the indices we swap together in a swaps list and we don't use last for loop as in above in heap sort function as we don't need output in array format but we need length of swaps array and each indices swapped together.

Images above is same for this example too but now we look at indices swapped to print them after printing the length of swaps array which indicates number of swaps.

Pseudo Code:

```
Heapify(arr,N,I,swaps):
Initialize Min as root
Left=2i+1
Right=2i+2
Check if left exceeds N and if left is less than root
        Swap left with min
Check if right exceeds N and if right is less than root
        Swap root with min
Check if min!=I
Swaps.append(I,min) //appending indices in swaps array
        Swap I with min
        Heapify again using new min (arr,N,min)

Heap sort(arr)
Swaps list
Loop from len(arr)//2 to 0
        heapify (arr,n,I,swaps)
```

Problems:

3)Merging Tables

The way I solved this problem is by knowing the concept which depends on checking the rank of each table by getting both destination and source parents which are given as input and we do check parent to make sure that data is in them or not as it might be added to it's parent so it's like a link checking once done check if source parent equal destination parent if so then source is destination and there is nothing to be done so we return false and then we check which one's rank is higher source or destination and the higher one becomes the parent of the other and we change the count of rows for that parent and put a zero in row count for the other one as we added it's row count to the other and we make the one that took rows from the other become it's parent now when we call it again we find a link to it's parent which contains the data of it then we check if max is changed after adding up rows to the one which was with higher row numbers and get the new max if there is one. And we do that in two cases depending on which is higher and on the third case if both ranks are same it doesn't matter which to choose to increase its rank to keep doing the process.

1row 1row

3 5

2rows link to 3

2 4 same as 3 5

2rows link to 2

1 4

link to 2 → (2rows)

2 + 1 = 3 Now (1) has 3

Now table (2) link to table 1)

5 4 → link to 2 → link to 1 (3rows)

link to 3 (2rows)

3 + 2 = 5

Now table 3 has 5 rows

5 3

link to 3 which gives 5 (samerows

Pseudo Code:

Get parent(table):

Check if table!=parent[table]

    If so get the parent of that table

Return parent[table]

Def merge(src,dst):

SrcP=getparent(src)

DstP=getparent(dst)

Check if srcP=dstP

Return false;

Check if srcP rank> dstP rank :

    Parent[dstP]=srcP

    RowC[srcP]+=RowC[dstP]

    RowC[dstP]=0

    Get max(max rowsC,RowC[srcP]

Else:

    Parent[srcP]= dstP

RowC[dstP]+=RowC[srcP]

RowC[srcP]=0

Get max(max rowsC,RowC[dstP]

If both ranks are equal of dst and src

increase dst rank by 1 or src (only one of them)

Sources used:

https://www.programiz.com/dsa/heap-sort