Name :- Harsimran Singh Dhillon.
B number = B00983136

# DAA CS575 - 2023

## Theory Assignment 2.2

Q1] We want to find the largest item in a list of n items.

a] Algorithm :-

```
int maximum (int low, int high) {
If (low == high) { // base condition
        return array [low] ——— 1 (running time)
}else {
    mid = (low + high)/2 ——— 1 (running time)
    int maxLeft = maximum (low, mid)      T(n/2) running time.
    int maxRight = maximum (mid+1, high)  T(n/2) running time

    If (maxLeft > maxRight){ // checking the maximum element
        return maxLeft ——— 1
    }Else {
        return maxRight.
    }
}
```

b] $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + T(n/2) + 1 & \text{Else} \end{cases}$

We have now

$$T(n) = 2T(n/2) + 1$$

$$T(1) = 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 1$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + 1$$

$$= 2\left[2T\left(\frac{n}{2^2}\right) + 1\right] + 1$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2 + 1$$

$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + 1$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 1\right] + 2 + 1$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^2 + 2 + 1$$

Similarly we will solve till k.

$$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} + 2^{k-2} \ldots 2^2 + 2 + 1$$

lets assume $\left(\frac{n}{2^k}\right) = 1$, because we know $T(1) = 1$

$$= 2^k T(1) + 2^{k-1} + 2^{k-2} \ldots 2^2 + 2 + 1$$

$$T(n) = 2^k + 2^{k-1} + 2^{k-2} \cdots 2^2 + 2 + 1$$

this is a G.P.

$$= \frac{1 - 2^k}{1 - 2}$$

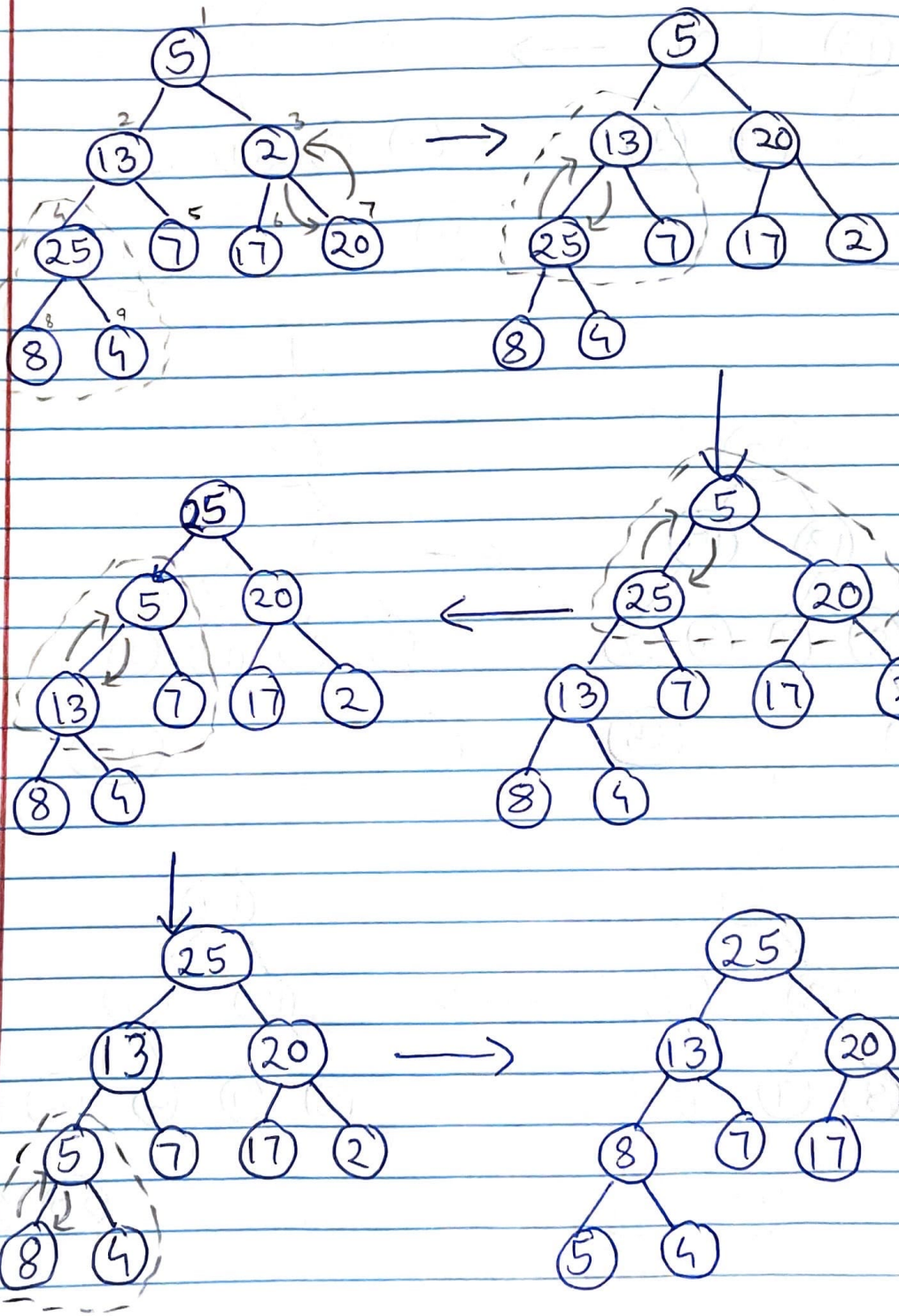
$$= 2^k - 1$$

$$\left[ \because \frac{n}{2^k} = 1 \,,\; n = 2^k \right]$$

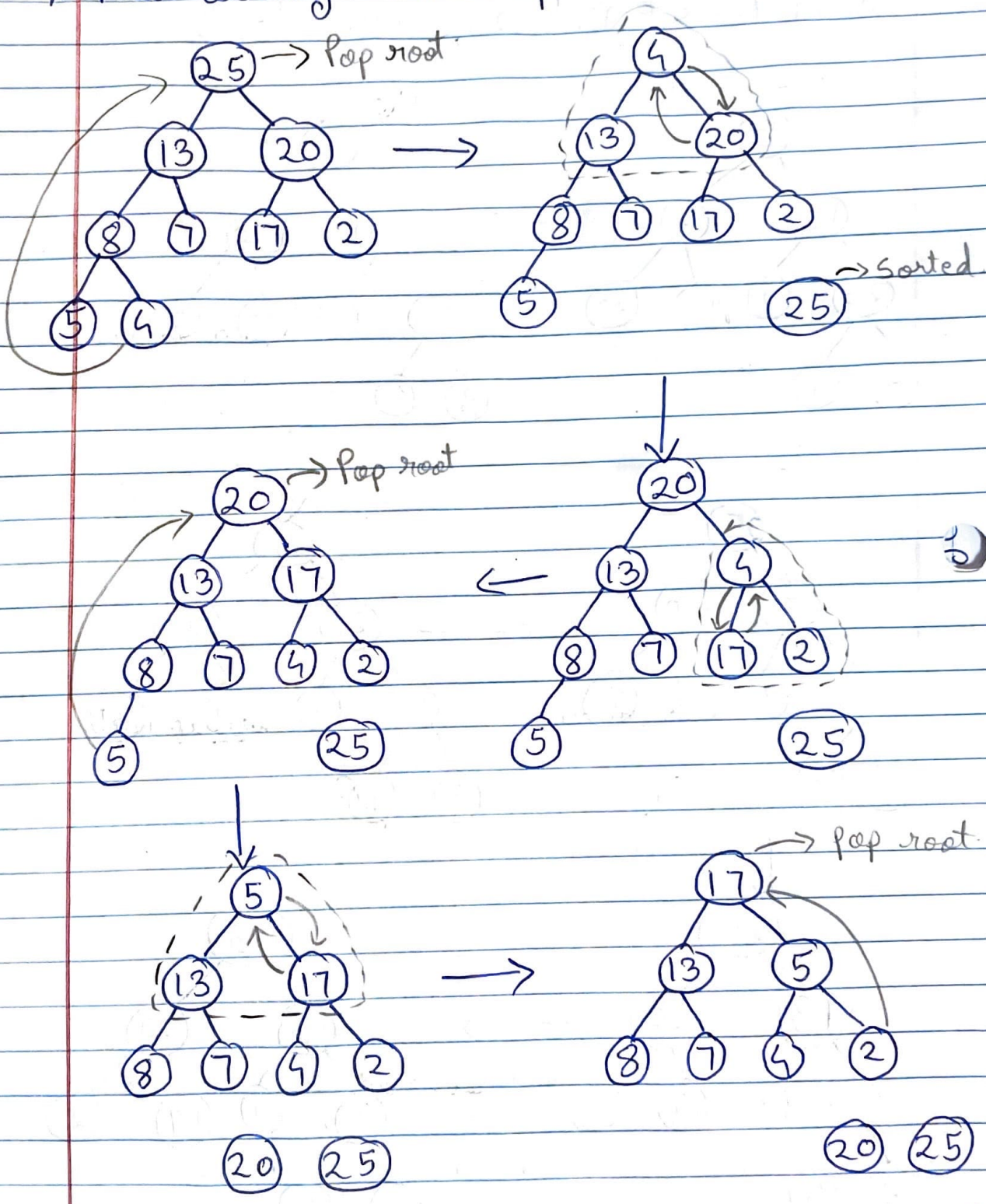
$$T(n) = n - 1$$

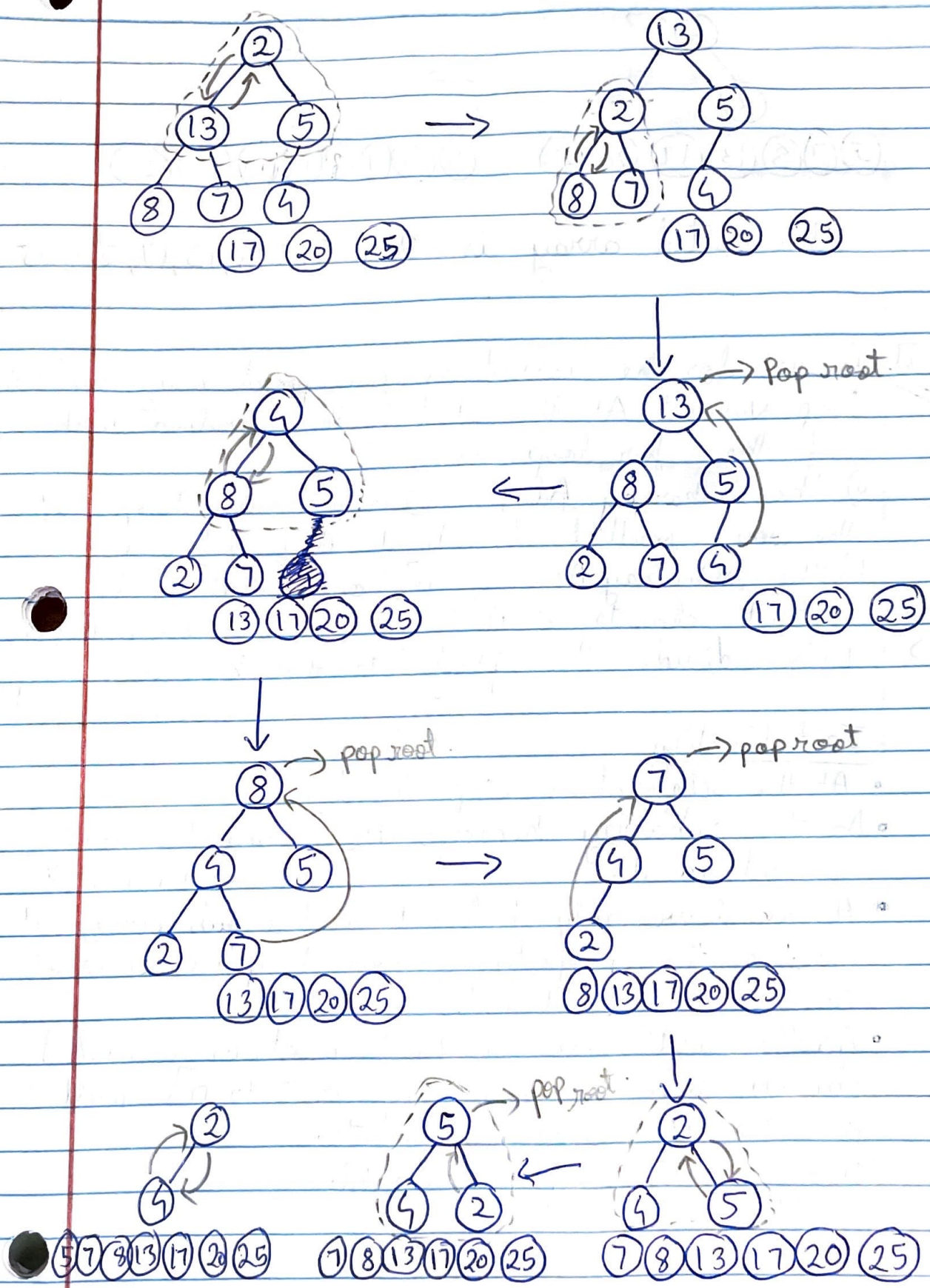
$$\boxed{T(n) = O(n)}$$

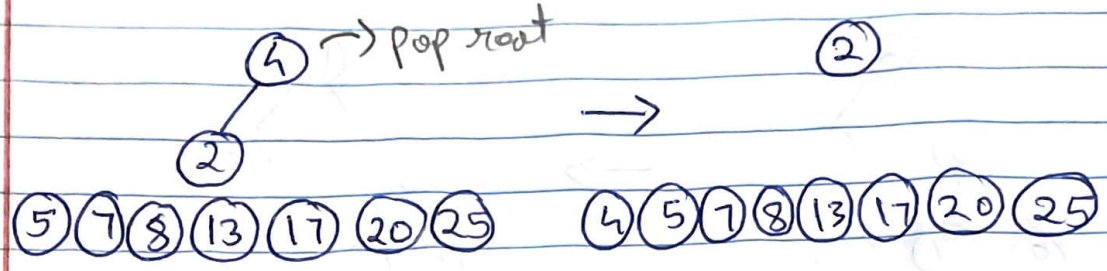
Q2] Illustrate the operation of Heapsort on the input array $A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$



* Now sorting the Heap.




2
13
5
8
7
4
17
20
25
13
2
5
8
7
4
17
20
25
Pop root
13
8
5
2
7
4
17
20
25
4
8
5
2
7
13
17
20
25
pop root
8
4
5
2
7
13
17
20
25
pop root
7
4
5
2
8
13
17
20
25
2
4
5
7
8
13
17
20
25
pop root
5
4
2
7
8
13
17
20
25
2
4
5
7
8
13
17
20
25




The sorted array is 2, 4, 5, 7, 8, 13, 17, 20, 25

Q3] Argue for the correctness of Heapsort using the following loop variant: At the start of the iteration with an i of the for loop,
a) the subarray A[1, ... i] is a max heap containing the one i smallest elements of A[1...n] and
b) the subarray A[i+1 ... n] contains the n-i largest elements of A[1...n] in correctly ordered asc.

Sol → Lets divide the proof into the three required parts.

Initialization :-

- At the initialization step i = 1
- So the subarray becomes A[1] and has only 1 element in it.
- As we have only 1 element in the sub-array, it satisfies the property of max-heap i.e root element greater than child nodes.
- As the first subarray A[1] contains 1 smallest elements, then the subarray A[2 to n] will have n-1 i.e n-1 largest elements.

Therefore this step is satisfied.

Maintenance :-

- In the maintenance, i will iterate through the loop.
- lets assume that for the $i^{th}$ element both the sub-array satisfy the condition.
- Since the above holds true till i, we can check the same for $i+1^{th}$ element.
- Now according to the (b) condition our $2^{nd}$ sub-array should have ~~to~~ ~~the~~ ~~largest~~ n-(i+1) largest elements So lets remove the first element from the $1^{st}$ sub-array a[1 to i~~+1~~]. We have removed the first element because it is the biggest element in the $1^{st}$ sub-array. According to the property of max-heap.
- After placing the first element in the $2^{nd}$ sub-array property (b) gets satisfied but property (a) is now unsatisfied.
- We now put the last element as root node i.e $1^{st}$ element of sub-array 1 and then call max-heapify to ~~sort~~ satisfy the property of max-heap.
- After max-heapify is done, property ~~b~~(a) gets satisfied. Thus the loop invariant is maintained for $i+1^{th}$ iteration.

Termination :-

- The loop will terminate when i becomes n-1.
- Considering that the loop invariant holds true till n-1 iteration.
- There will one last action step left, i.e to send the ~~max~~ element to the back of the array.
- After we remove the first element and place it as the last element. ~~We swap the smallest ele~~
- Now the entire Heap has been traversed and we have sorted all elements. Thus the loop invariant ~~is~~ holds true for termination